Simulation technologies for C^2IS development & training

Final report

Thibault, D. U.
DRDC Valcartier
Abstract

A functional overview of the High Level Architecture (HLA) distributed simulation standard Institute of Electrical and Electronics Engineers (IEEE) 1516 is provided. The Agent Oriented Software (AOS) intelligent software agent development environment, JACK, is described. Problems with the IEEE 1516 specification are identified and possible improvements outlined. A demonstration Internet Relay Chat (IRC)-like application is designed and implemented, which allows a Java application to interoperate with a JACK application through the HLA. A re-usable JACK capability is described which allows any JACK agent to participate in an HLA federation. A re-usable framework of Java classes implementing the HLA Object Model Template (OMT) encoding and decoding facilities is described. Finally, bugs are documented in JACK and the HLA Run-Time Infrastructure (RTI) used, and various design considerations and lessons learned are discussed.

Résumé

Le fonctionnement de la norme de simulation répartie High Level Architecture (HLA) Institute of Electrical and Electronics Engineers (IEEE) 1516 est décrit de façon concise. L'environnement de développement d'agents logiciels intelligents JACK d'Agent Oriented Software (AOS) est décrit. Des problèmes sont identifiés au niveau de la norme IEEE 1516 et des améliorations possibles sont décrites. Une application semblable au clavardage Internet a été conçue et implémentée, permettant de démontrer comment HLA permet l'interfonctionnement d'une application Java avec une application JACK. Une capacité JACK réutilisable permettant à n'importe quel agent JACK de participer à une fédération HLA quelconque est décrite. Un cadre de classes Java implémentant les méthodes de codage et d'encodage de l'Object Model Template (OMT) HLA est décrit. Finalement, les bogues de JACK et du Run-Time Infrastructure (RTI) HLA sont documentés, et diverses considérations et leçons apprises discutées.
Executive summary

Distributed simulation, both in its logical and geographical senses, allows one to leverage existing and future simulations to the best of their individual capabilities. Live, virtual, and constructive simulations are all susceptible to benefit from a distributed implementation. A live simulation involves real people operating real systems; the latter are modified (blanks instead of live ammunition, for example) or augmented (laser tag equipment, for example) in order to allow realistic action to occur without the actual risk of casualty or destruction. A virtual simulation involves real people operating simulated systems; the humans are in-the-loop in a central role, exercising motor control skills, decision, or communication skills. A constructive simulation, finally, involves simulated people operating simulated systems. Under certain conditions, constructive simulations are the only ones that can be run at vastly increased simulated rates of time lapse. For example, an entire large-scale operation spread over days or weeks could be simulated in a matter of seconds, given powerful enough (and smart enough) computers.

Geographical distributedness allows the simulation audience to be widely spread across actual space, as long as this does not compromise the illusion. Obviously, a real person and a real piece of equipment must be co-located for the one to operate the other, but if other entities are perceived artificially (e.g. through the tactical radio, through a radar screen, or through a simulated outside view), distributedness in principle allows them to be at arbitrarily large distances from each other (in practice, time lag problems limit the actual distances achievable). The immediate benefit is to allow exercises of any scale to occur at lesser cost in time and expense, as the participants need not move to a central location. Similarly, there is a lesser need in infrastructure. For example, instead of several sets of simulator units being installed in several locations, each one able to accommodate a full platoon, there could be only one platoon’s worth of simulators, each individual one at a separate location.

Logical distributedness occurs independently of geographical distributedness; its essence lies in breaking down responsibility for various aspects of the simulation across different simulation software entities. This allows greater fidelity to be achieved by the combined simulation than would be possible using each participating simulation by itself. Each participating simulation, by mutual agreement, becomes responsible for one or more aspects of virtuality for which it is the recognised “expert.” For example, an all-purpose simulation that includes a crude missile fly-out model could, through logical distribution, turn over responsibility for its missiles’ flight to a specialised simulation, thus improving the realism of the whole. Another example would be target detection; once the host simulation has established line of sight exists between any two platforms, it could rely on another simulation to resolve whether detection occurs or not, the latter simulation using a sophisticated atmospheric propagation model to do so.
The benefits of logical distributedness are many, not least of which is a large potential for systematic re-use of models and simulations of established validity and sophistication. Putting together a simulation for some specific need, be it training or research, then becomes a matter of determining which aspects require the most fidelity, and looking up the appropriate contributing simulations in some catalogue. Users are no longer locked into some specific investment (monolithic simulation environment), but may switch freely to improved components as they arise. Developers are free to specialise in some narrow aspect of their model, knowing any “deficiencies” their model may have in other aspects will easily be compensated, if needed, by the integration of the appropriate components.

Distributed simulation is made possible through the High Level Architecture (HLA) standard, now in its second generation: Institute of Electrical and Electronics Engineers (IEEE) 1516. This document summarises the intricacies and subtleties of the standard in what is hoped to be a complement to the standard’s defining documents themselves. Time management is given minimal treatment while ownership and data distribution management are looked at in detail. The specification, as amended by the U.S. Department of Defense (DoD), is discussed and apparent ambiguities and lacunae are identified, with solutions proposed.

Since the only fully certified Run-Time Infrastructure (RTI; the concrete implementation of the middleware standard) at this time is Pitch’s pRTI1516, that is what was used. Few problems were encountered, and these are documented. No comparisons or performance benchmarking could be conducted.

A leading software intelligent agent design environment, Agent Oriented Software’s JACK, is summarised so that a clear picture of the component blocks of an agent application emerges. The task of integrating a given JACK agent into an HLA federation is tackled, using as an example a multi-user communication application modelled after Internet Relay Chat (IRC).

Difficulties inherent in multi-threading applications were encountered, and the mechanisms required to surmount them within the HLA paradigm were implemented. Retrospection reveals how several of these could be improved, and this is discussed.

A re-usable JACK “capability” module was designed and implemented, and quickly retrofitted to the older 1.3 standard. A re-usable and extendable framework of Java classes implementing the HLA Object Model Template (OMT) encoding and decoding facilities is described and implemented. It can be used until such time as an evolved version of HLA IEEE 1516 appears which resolves the related issues. Miscellaneous other utility classes are provided as well.
It is hoped that the work will facilitate the integration of intelligent agents such as those made possible by JACK into existing or future simulations in use by the Canadian Forces, or by various Defence Research & Development Canada projects, such as those looking into cognitive effects and other human factors. The work should also be of some use to any effort involving HLA in general.

Sommaire

La simulation répartie, dans ses sens logique et géographique, permet d’exploiter au mieux de leurs capacités individuelles les simulations contemporaines et à venir. Les simulations réelles, virtuelles et constructives sont toutes susceptibles de bénéficier d’une implémentation répartie. Une simulation réelle implique des personnes réelles faisant fonctionner des systèmes réels ; ces derniers sont modifiés (p. ex. fausses balles) ou augmentés (p. ex. équipement laser) afin de permettre le réalisme sans encourir les risques habituels de destruction ou blessure. Une simulation virtuelle implique des personnes réelles faisant fonctionner des systèmes simulés ; les humains sont dans la boucle de façon centrale, exerçant leur habileté manuelle, ou leurs compétences de décision ou de communication. Une simulation constructive, enfin, implique des personnes simulées faisant fonctionner des systèmes simulés. Dans certaines conditions, les simulations constructives sont les seules à pouvoir être exécutées à un rythme fortement accéléré. Par exemple, une opération à grande échelle se déroulant pendant des jours ou des semaines pourrait être simulée en quelques secondes, si la puissance (et l’intelligence) des ordinateurs le permet.

La répartition géographique permet au personnel impliqué d’être éparpillé spatialement, du moment que l’illusion n’est pas compromise en ce faisant. Évidemment, une personne et un équipement réels doivent être au même endroit afin que l’un puisse faire fonctionner l’autre, mais si les autres entités ne sont perçues qu’indirectement (p. ex. à travers la radio tactique, un écran radar ou encore une vue extérieure simulée), la répartition permet, en principe, à ces entités d’être à une distance réelle arbitraire (en pratique, les délais de transmission limitent les distances réelles permises). Le bénéfice immédiat est que les exercices, quelle que soit leur échelle, peuvent être réalisés à moindre coût en temps et ressources, car les participants n’ont plus à se rendre physiquement à un même endroit. Similairement, il y a un besoin moindre en infrastructure. Par exemple, au lieu de plusieurs ensembles de simulateurs installés en divers endroits, chacun en mesure d’accommoder un peloton entier, il devient envisageable d’avoir un seul peloton de simulateurs, répartis en divers endroits à la pièce.

La répartition logique peut avoir lieu indépendamment de la répartition géographique : son essence consiste à distribuer la responsabilité de divers aspects de la simulation entre différentes entités logicielles. Ceci permet à la simulation combinée d’atteindre une fidélité supérieure à celle possible avec une seule des simulations participantes. Chaque participant, d’un commun accord, se charge des aspects de la virtualité pour lesquels il est reconnu être l’« expert ». Par exemple, une simulation à tout faire qui inclut un gros modèle de vol de missiles pourrait, grâce à la répartition logique, confier la responsabilité du vol de ses missiles à une simulation spécialisée, améliorant ainsi le réalisme de l’ensemble. Un autre exemple pourrait être l’acquisition de cibles, une fois que la simulation hôte a établi l’existence d’une ligne de visée entre les plateformes impliquées, elle pourrait se fier à une autre simulation (utilisant un modèle de propagation atmosphérique sophistiqué) pour déterminer s’il y a détection ou non.
Les bénéfices de la répartition logique sont nombreux, à commencer par la ré-utilisation systématique de simulations dont la validité et la sophistication sont établies de longue date. La conception d'une simulation répondant à un besoin spécifique, qu'il s'agisse d'entraînement ou de recherche, consiste alors à déterminer les aspects requérant la plus grande fidélité, pour ensuite identifier les simulations devant être mises à contribution dans un catalogue. Les utilisateurs ne sont plus prisonniers d'un investissement quelconque (un environnement de simulation monolithique), mais peuvent transitionner librement vers des composants améliorés lorsqu'ils surviennent. Les développeurs sont libres de spécialiser leur modèle, sachant que les “déficiences” que leur simulation pourrait présenter seront facilement compensées, si nécessaire, par la participation d'autres simulations.

La simulation répartie est rendue possible par la norme High Level Architecture (HLA), qui en est maintenant à sa seconde génération: Institute of Electrical and Electronics Engineers (IEEE) 1516. Ce document en explique les subtilités et les détails d'une façon qui se veut complémentaire aux documents de la norme eux-mêmes. La gestion du temps se voit accorder un minimum d'attention, tandis que la gestion de la propriété ainsi que la gestion de la répartition des données sont examinées attentivement. La norme, telle qu'amendée par l’U.S. Department of Defense (DoD), est discutée et d'apparentes ambiguïtés et lacunes sont identifiées, et des solutions sont proposées.

Puisque le seul Run-Time Infrastructure (RTI; la réalisation concrète de la norme intergicielle HLA) pleinement certifié lors de ces travaux est le pRTI1516 de Pitch, c'est ce qui fut utilisé. Quelques problèmes furent rencontrés, et ils sont documentés. Il n'y avait pas lieu de faire des comparaisons ou des étalementages de performance.

Un environnement de conception d'agents logiciels intelligents d'avant-garde, JACK d'Agent Oriented Software, est décrit avec suffisamment de détail pour qu'une idée claire puisse se faire des composants qui constituent une application faisant usage d'agents intelligents. Nous nous attelons ensuite à la tâche d'intégrer un agent JACK donné à une fédération HLA, en utilisant comme exemple une application de communication à multiples usagers, simulant de près le clavardage Internet.

Des difficultés inhérentes aux applications multi-fil ont été rencontrées, et les mécanismes permettant de les surmonter dans le contexte du paradigme HLA ont été implantés. Une rétrospective révèle comment certains de ceux-ci pourraient être améliorés, et ces options sont discutées.

Un module de “capacité” JACK réutilisable a été conçu et réalisé, et rapidement rétro-ajusté à l'ancienne norme HLA 1.3. Un cadre de classes Java réutilisable et extensible réalisant les méthodes d'encodage et de décodage de l'Object Model Template (OMT) HLA est décrit et réalisé. Il pourra être exploité jusqu'à ce qu'une version évoluée d’HLA IEEE 1516 apparaisse qui résolve cette difficulté de la norme courante. Diverses autres classes utilisitaires ont été également réalisées.
Ces travaux devraient faciliter l’intégration d’agents intelligents comme ceux rendus possibles par JACK à des simulations existantes ou futures, utilisées par les Forces canadiennes ou par divers projets de Recherche et développement pour la défense Canada, comme ceux touchant aux effets cognitifs et autres facteurs humains. Ces travaux devraient également être de quelque utilité pour tout effort impliquant HLA en général.

# Table of contents

Abstract ................................................................. i

Executive summary ................................................ iii

Sommaire ...................................................................... vi

Table of contents ...................................................... ix

List of figures ........................................................... xv

1. Introduction .............................................................. 1
   General ................................................................. 1
   Background .......................................................... 1
   Context ............................................................... 2
   Aim ..................................................................... 5
   Objectives ......................................................... 6
   Scope ............................................................... 6
   Outline ............................................................. 6

2. High-Level Architecture (HLA) .................................... 9
   Historical Perspective ........................................... 9
   Framework and Rules ............................................ 10
   Synthetic Environment ......................................... 13
   HLA Interface .................................................... 15
   HLA Services ..................................................... 16
      Federation Management ...................................... 16
      Declaration Management ................................. 22
      Object Management ........................................ 23
      Ownership Management ................................... 24
      Time Management .......................................... 28
      Data Distribution Management ......................... 29
      Support Services .............................................. 36
Threads and Process Models ........................................................................................................ 37
Synchronization Issues .................................................................................................................. 38
Pitch pRTI1516 ........................................................................................................................... 39
   Bugs ..................................................................................................................................... 41
Visual OMT 1516 .......................................................................................................................... 44
   Bugs ..................................................................................................................................... 44
3. JACK™ Intelligent Agents ........................................................................................................... 45
   Overview of JACK ................................................................................................................... 46
      Events ................................................................................................................................. 46
      Belief Sets ......................................................................................................................... 47
      Plans ................................................................................................................................. 48
      Capabilities ....................................................................................................................... 49
      Multi- Threading ............................................................................................................... 49
   JACK-HLA interface ............................................................................................................... 50
      HLA services, callbacks and JACK events ................................................................. 50
      HLA1516 JACK Capability ............................................................................................. 51
      HLA13 JACK Capability ................................................................................................. 55
      JACK Integrated Development Environment (IDE) ..................................................... 55
      JACK Bugs ...................................................................................................................... 56
      JACK Pre-Processor Bugs .............................................................................................. 56
4. Chat Application ........................................................................................................................ 59
   JACK Chat Demonstration ..................................................................................................... 60
   HLA Chat Demonstration ....................................................................................................... 65
   HLA Chat Federation Object Model .................................................................................... 66
   Assertions .............................................................................................................................. 77
   Federate Life Cycle ............................................................................................................... 78
5. Conclusions ................................................................................................................................ 83
Internal distribution (DRDC Valcartier) ............................................ 1361
External distribution ........................................................................ 1361

Note: The Annexes B through F are included only in the CD-ROM version of the report.
This page intentionally left blank.
List of figures

Figure 1. The Topographic Map Display (ToMaDi) ................................................ .4
Figure 2. The ImmersaDesk ...................................................................................... 5
Figure 3. High Level Architecture ........................................................................ 11
Figure 4. The intersections of the object models ..................................................... 12
Figure 5. A subset of the RPR-FOM version 2.0 object class hierarchy ................. 13
Figure 6. The interfaces between a federate and the HLA RTI ............................... 16
Figure 7. The global federate states and their transitions ................................. 17
Figure 8. The joined federate states ..................................................................... 19
Figure 9. The federate save and restore states ...................................................... 20
Figure 10. Interchangeability of federates of the same type ................................. 22
Figure 11. Known/Unknown transition state diagram ............................................ 23
Figure 12. Owned/Unowned transition state diagram ............................................. 26
Figure 13. Not Acquiring/Acquisition Pending transition state diagram ............. 27
Figure 14. Dimensions, ranges, regions, and region sets ..................................... 30
Figure 15. Transitions between region templates, specifications and realizations . 31
Figure 16. Relationship between dimension sets .................................................. 33
Figure 17. An example of overlapping publication and subscription region sets .. 34
Figure 18. Another example ................................................................................. 35
Figure 19. How an agent uses the HLA capability .............................................. 54
Figure 20. A high-level view of a chat application .............................................. 60
Figure 21. The JACK Chat Server window ....................................................... 61
Figure 22. The JACK Chat Client window at start-up ....................................... 61
Figure 23. The JACK Chat Client window after logging in ................................ 63
Figure 24. The JACK Chat demonstration .............................................................. 64
Figure 25. The Java Chat demonstration ................................................................. 65
Figure 26. The Java Chat Client window after start-up ........................................ 66
Figure 27. The Java and JACK Chat Client windows after both have joined ......... 67
Figure 28. The Chat federation in action .............................................................. 67
Figure 29. The Participant objects and the ChatRoomSlots dimension ............... 76
Figure 30. The Java Chat client in the disconnected state .................................... 997
Figure 31. The Java Chat client in the joined state .............................................. 997
1. Introduction

General

Modelling and simulation (M&S) are recognised as tools of paramount importance throughout the military endeavour. Physical M&S can be used to evaluate engineering solutions to specific problems (e.g. Which shape of helmet best protects against a projectile impact? Which layering of materials offers the best armour protection against a specific threat? Which disposition of controls best suits the ergonomics of the cockpit?). Logistics and demographics simulations can be used to evaluate force deployability and sustainability solutions. Virtual and real simulations are constantly used in training at all levels, ranging from small arms individual training, through simulators (aircraft, armoured vehicles, ship bridges) all the way up to command post exercises. Simulations are also extensively used in the design phases of acquisition projects, to refine requirements, identify errors before they result in expenses, and identify potential synergies. During real operations, simulations can be used to put to the test proposed friendly (or suspected enemy) courses of action, identifying overlooked opportunities or threats. And so on.

Faced with this overwhelming breadth of applicability and abundance of diverse applications, simulation interoperability quickly emerged as a lynchpin. The High Level Architecture (HLA) is the still-evolving standard that strives to make this interoperability happen.

Background

In the Research and Development (R&D) domain, it is necessary to explore a variety of issues with a minimum of effort and expenditure of resources. Both to ensure the validity and applicability of the research and to avoid duplicate development and maintenance, the simulations used in R&D ought to be the same as (or at least have much in common with) those used in the military community. However, in the military application domain simulation is overwhelmingly used for training purposes. Command and control (C²) training simulations perforce do not automate or simulate the C² process, since that is precisely the expected trainee benefit. One can oversimplify the situation like this: for training purposes, the more humans are involved, the better. New equipment is introduced in the training scenarios only shortly before it is actually deployed. The preparation and execution of the scenarios is a complex, labour-intensive affair. Hence the conundrum: simulations designed for training purposes have manpower requirements which make them poorly suited to the R&D environment.
Consequently, R&D tends to minimise the scope of simulation vignettes used to explore new concepts, in order to make the running of the simulations achievable with a minimum of personnel. This approach is quite successful in the “material” domains, but what if one wishes to study the complex interactions of multiple systems in the overall C^2 process? The number (and nature) of entities to be simulated cannot be reduced without voiding the purpose pursued. There is a need to provide a reasonable level of “intelligence” to the entities. This is, of course, a much harder problem than physical simulation, but artificial intelligence has made some progress over the decades since its inception.

One of the paradigms that holds a fair amount of promise nowadays is the so-called intelligent agent approach, which strives to provide software entities with deliberative and pro-active qualities.

**Context**

The 12sd *Simulation technologies for C^2 information system (C^2IS) development & training* project was known as 12kr before 2003. Briefly designated 2kq when created in 2000, it was at first called *Army C^2IS development and training through simulation*. The project was an administrative means of regrouping a number of disjointed work units. The original aim was “To demonstrate an HLA simulation federation involving a simulation engine (with emphasis on OOTW support), command agents, an LFCS stand-in, an intelligence CCIS subset (ASIP) and decision aid tools.” The objectives were:

- **Command Agents:** Explore models of the decision making process of military command and formalise a command model (below battalion, e.g. section, platoon, company and battalion) derived from contemporary artificial intelligence. The models must be flexible enough to represent doctrinally different allies and enemies as well as non-military entities (insurgents, refugees, organized crime, non-governmental organizations, etc.).

- **Tools:** Explore development and implementation environments and tools applicable to HLA object models (simulation and federation), HLA federation development process management, and command agents.

- **Analysis:** Evaluate the efficiency of command agents in land tactical CCIS over a distributed simulation environment. Evaluate several different mixed CORBA/HLA distributed environments. Evaluate desirable levels of HLA compliance on the part of the CCIS.

- **Demonstration:** Demonstrate the usefulness of the federation by using it to test advanced land intelligence tools in a fully interactive peace-keeping simulated exercise.

- **Interfaces:** Explore 3D visualisation and multi-modal interfaces for exercise/experiment preparation and execution.
With the eventual termination of one of the work units, the splitting off of another, and unexpected difficulties due in part to the immaturity of the domain and shifting standards, considerably less was achieved than hoped at the outset.

This report concerns the 12sd11 work unit, *Simulation technologies*. Besides the work being reported here, there was also another report, *Conformation à et exploitation de HLA par ASIP (ASIP Compliance to and Exploitation of HLA)*, written by Catherine Daigle ([1] Daigle 2001), who later became a member of DRDC Valcartier’s Decision Support Systems (DSS) Section. This report is included alongside this one on the CD (Compact Disc).

The 12kr12 work unit, *Command agents in simulation*, was the responsibility of Kenneth N. Ackles and Dr. Benoit Jean Fugère of the Royal Military College (RMC) Kingston. The goal was originally stated as the “exploration of models of computer-generated forces (CGF) under command, which must react to orders and the enemy in a realistic, doctrinally sound manner.” This was tightly integrated with the RMC graduate programs in Military Modelling & Simulation and RMC’s implication in the High Performance Computing Virtual Laboratory (HPCVL). A prototype armoured squadron set of command agents (using JACK) was allegedly completed. Alternate architectures involving neural networks and fuzzy logic were investigated. No reports were ever delivered to Defence R&D Canada (DRDC), although a number of theses were written and papers were published ([2] Liang and Fugère 2000, [3] Liang et al. 2001, [4] Robichaud 2001, [5] Liang and Fugère 2002, [6] Jaillet et al. 2002). The decision was made to terminate the work unit in early 2003.

The 12kr13 work unit, *New media*, was the responsibility of Roger Fortin of DRDC Valcartier. It concerned itself with a novel interface device, a Topographical Map Display (ToMaDi) ([7] Fortin 2001a). Second and third generations of the prototype device were realised and field tested in three major military exercises. ToMaDi MkII ([8] Fortin 2001b, [9] Fortin 2001c) consists of a tiltable 4 by 4 matrix of sixteen colour Thin Film Transistor - Liquid Crystal Displays (TFT-LCDs, essentially portable computer screens) combined with a touch screen. It presents so far the only practical alternative to topographical paper maps for C² applications. With ToMaDi, it is possible to physically step back and view the entire map at once, and step up for the detail picture, without zooming in and out nor restricting access to one person at a time. Presently, two technologies are being investigated to make the ToMaDi’s mullions vanish: aspheric lenses and retroprojection (reports in preparation). ToMaDi contributes to the digitization of the battlefield.
The 12kr14 work unit, *Urban simulation*, was the responsibility of François Létourneau of DRDC Valcartier. It split off from 12kr in fiscal year 2001-2002 to become 12kx, which later became project 15al *Geospatial technologies for information decisions (GEO-TIDE)*. The work focused on the development of three-dimensional models of urban areas, specifically Québec City. This entailed the purchase of 3D modelling software and visualisation hardware (e.g. an ImmersaDesk). Various methodologies of 3D urban model creation were examined in order to gauge the effort required and identify the challenges, stakes and technical and methodological considerations involved. Two reports documented this work ([10] Létourneau et al. 2003, [11] Martel and Létourneau 2003).

The 3D models of Québec City were used for technology demonstration purposes during the Summit of the Americas in 2001 by the RCMP headquarters. The *Soldier information requirements (SIREQ)* Technology Demonstration (TD) project asked 12kr14 to produce several versions of the Québec City 3D model which were used in an experiment that studied how different geospatial representations affected the capacity of soldiers to orient themselves in an urban operations field (reports in preparation).
The 12sd15 work unit, *Display assessment and enabling technology research for new military displays*, was also the responsibility of Roger Fortin of DRDC Valcartier, and a Defence Industrial Research (DIR) project. It was intended to provide enhanced display design capabilities through the application of commercially available display components via the development of enabling technology elements. The work focused on investigations into Active Matrix Liquid Crystal Display (AMLCD), Field Emission Display (FED), and Organic Light-Emitting Diode (OLED) displays and into the associated core technologies necessary to implement military displays. Solutions to a wide variety of display-related problems were identified and the basic technology foundations developed ([12] Thomas 2004). An OLED display was built and demonstrated.

**Aim**

The aim of this document is to provide a number of insights into the technical aspects of the current HLA standard, Institute of Electrical and Electronics Engineers (IEEE) 1516, and into the technical issues relating to its interface with one particular software intelligent agent development environment, Agent Oriented Software’s JACK.
Intelligent agents model reasoning behaviour according to the theoretical Belief Desire Intention (BDI) model of artificial intelligence. They are autonomous software components that have explicit goals to achieve or events to handle ("desires"). To achieve their goals, the agents follow plans, which are high-level descriptions of expert behaviour. Set to work, the agents pursue their given goals (desires), adopting the appropriate plans (intentions) according to their current set of data (beliefs) about the state of the world. This combination of desires and beliefs initiating context-sensitive behaviour is a central characteristic of BDI agents.

Objectives

The specific objectives of this report are to:

- Provide an overview of HLA IEEE 1516, focusing on a subset of the services it provides;
- Provide an overview of Agent Oriented Software’s JACK;
- Suggest improvements to the IEEE 1516 specification, building on those already promulgated by the U.S. Department of Defense;
- Document bugs in the only commercially available RTI that fully implements IEEE 1516, Pitch’s pRTI 1516;
- Document bugs in the current version of JACK; and
- Document various design considerations, tips, tricks and lessons learned in implementing a specific federation.

Scope

This document is intended for those who are familiar with the high-level concepts of HLA and software engineering.

Outline

The sections examine, in turn, the HLA, the JACK intelligent agents, and the demonstration federation and Chat application. In the annexes, we comment on the IEEE 1516 specification, provide Javadoc for the HLA 1516 Application Programming Interface (API), document our re-usable supporting classes, provide a closer look at the Java Chat Client and provide its code, document the re-usable JACK capHLA1516 capability, and provide the JACK Chat Client code.
This document is also distributed in electronic form, as a CD-ROM (Compact Disc –
Read-Only Memory). The CD’s directory structure is as follows:

- **Java**
  - bin
  - chat
  - doc
  - doccheck
  - lib
  - src

- **JACK**
  - chat
  - capHLA1516
  - capHLA13

The Java source files are rooted at Java\src; for example, 
`hla.rti1516.AttributeHandle.java` is found at
Java\src\hla\rti1516\AttributeHandle.java. The corresponding class
files are rooted at Java\lib. Java\bin contains jar files that provide an alternate
representation of the class files. The doccheck directory contains the results of
running the DocCheck doclet on the source files, while the doc directory contains
the generated Javadoc.

The code supplied in the annexes is found here:

- The `hla.rti1516 IEEE 1516 Java API with Javadoc added` (Annex B);

- The `ca.gc.drdc_rddc.hla.rti1516 Integer64 family of LogicalTime[Interval][Factory] implementations` called for by the specification but missing from the Pitch pRTI1516 product;

- The `ca.gc.drdc_rddc.hla.rti1516.omt` supporting classes (Annex C);

- The `ca.gc.drdc_rddc.hla.rti1516.FedAmb` supporting classes (Annex D);


The JACK source files are in the **JACK** directory:

- The `chat` project (Annex F Part Two);

- The `capHLA1516 re-usable capability` (Annex F Part One); and

- A `capHLA13 re-usable capability for HLA 1.3NGv6` (bonus).
2. High-Level Architecture (HLA)

Historical Perspective

Computer simulation was developed hand-in-hand with the rapid growth of the computer. One of the very first large-scale efforts was the World War II Manhattan Project’s computer modelling of the process of nuclear detonation. In the military domain, simulators had been used for individual training purposes long before the computer was invented. Likewise, the “war game” as a command training tool dates back at least to the Napoleonic era. Computers, naturally, were at first used to enhance these disparate tools on a piecemeal basis. Thousands of different computer applications were eventually developed to fulfill an immense variety of training and research requirements. Simulations grew from the bottom up, features being added every time they were felt necessary, leading to repeated reinvention of the same wheels. Interoperability was virtually non-existent. By the early 1980s, it was generally considered impossible to build an affordable, large-scale, collective, free-play, force-on-force, interactive, worldwide networked war fighting system ([13] Cosby 1999).

That is precisely what the U. S. Defense Advanced Research Projects Agency (DARPA) SIMulator NETwork (SIMNET) programme set out to disprove (SIMNET later became SIMulation NETwork, to reflect its widening span). Initiated in 1983, it was the first step in moving from individual task-based simulators to a network of low-cost simulators that fought on a virtual battlefield to provide "synchronized execution of collective war fighting skills in a combined arms and joint arena." Eventually, the University of Central Florida’s Institute for Simulation and Training (IST) was contracted by DARPA to undertake research in support of SIMNET. The first draft of the resulting Distributed Interactive Simulation (DIS) protocol appeared in 1992 ([14] DIS 1992), and it quickly became an IEEE standard, IEEE 1278 ([15] IEEE 1993). DIS enjoyed considerable success and is still used in some contexts.

In 1989, DARPA started extending the SIMNET DIS principles to aggregate level constructive training simulations. The effort led in 1991 to the Aggregate Level Simulation Protocol (ALSP) which was first used to support major exercises in 1992 ([16] MITRE 1993). ALSP laid the theoretical foundations for the High Level Architecture (HLA) while becoming a multi-service program focusing on the support and execution of the Joint Training Confederation (JTC). By 1998, the ALSP Confederation would start transitioning to HLA.
Starting in 1995, the U. S. Defense Modeling and Simulation Office (DMSO) developed the High Level Architecture as a standard for distributed simulation that could support any kind of simulation (live, virtual or constructive, real-time or not) while encouraging re-usability as much as possible by not imposing semantics on the participating simulations ([17] Kuhl et al.). First published and adopted as the U.S. Department of Defense de facto standard architecture for distributed simulation on 10 September 1996 ([18] DoD 1996), HLA underwent several revisions, eventually becoming the Object Management Group (OMG) standard Facility for Distributed Simulation Systems 1.0 in November 1998 ([19] OMG 1998a). The DMSO demonstration Run-Time Infrastructure (RTI, the implementation of the middleware allowing HLA distributed simulations to take place) underwent successive revisions in parallel, eventually stopping with version 1.3NGv6. It was provided free-of-charge to encourage experimentation and development. In 2002, distribution was shut down to avoid competing with the burgeoning commercial implementations of HLA.

Simultaneously with the OMG ratification process, the standard was submitted to the IEEE, partly in order to rely on a non-governmental standard, since American law requires the U.S. Department of Defense (DoD) to use such non-governmental standards when consistent with mission, priorities and budget resources. The IEEE gave it a thorough going over, resulting in IEEE 1516 in 2000-2003 ([20] IEEE 2000). The IEEE standard deliberately avoided including a Common Object Request Broker Architecture (CORBA) ([21] OMG 1998b) Interface Definition Language (IDL) specification (unlike HLA 1.3), leaving this up to the OMG. The result was the OMG Facility for Distributed Simulation Systems 2.0 in February 2002 ([22] OMG 2002).

Version 2 of IEEE 1516 has just begun preparation and is expected to be released within the next few years.

The most recent edition of DoD 4120.24-M, Defense Standardization Program (DSP) Policies and Procedures, is dated 9 March 2000 and does not yet formally adopt either HLA 1.3 or IEEE 1516 ([23] DoD 2000). The situation is similar on the North Atlantic Treaty Organization (NATO) side, where Standardisation Agreement (STANAG) 4574, Standardized Modelling and Simulation Information for High Level Architecture (HLA), ([24] NATO 1999) has yet to replace STANAG 4482, Standardized Information Technology Protocols for Distributed Interactive Simulation (DIS) ([25] NATO 1995). Nevertheless, the trend is clear: HLA is the way of the future.

Framework and Rules

HLA strives to be an interface definition which is flexible enough to accommodate any combination, in number and kind, of simulations. Live, virtual and constructive simulations can, in principle, be made to interoperate using HLA. HLA is fully described by the IEEE 1516 documents (see the Bibliography for a full list), so we do not reproduce them here. Instead, a brief overview and summary is provided.
The functional components are of two kinds: the federate applications and the runtime infrastructure (RTI). The federate applications are the individual simulation applications. The RTI plays a role similar to an Object Request Broker (ORB) under the CORBA; it is the (very likely distributed) set of software services which links the federates together (see Figure 3).

![Figure 3. High Level Architecture](image)

Here several federates participate in two different federation executions (red and green pipes) through the RTI. An unjoined federate can also be seen.

Each federate application has a Simulation Object Model (SOM): a software document written according to the rules laid out as part of the standard (the Object Model Template (OMT)). This SOM describes the objects, interactions, dimensions and other features of the simulation relevant to interoperability. Each version of the simulation application will likely have a different SOM, unless the changes are entirely transparent to the other federates. That is to say, implementation details can change freely, as long as the interface and the semantics do not. The actual operation, over time, of a federate application is referred to as a federate.

A federation is a named set of federate applications and a common Federation Object Model (FOM) that are used as a whole to achieve some specific objective. The FOM is best thought of as the relevant intersection of the SOMs of the federates involved (see Figure 4). The FOM is concretely represented by the FOM Document Data (FDD), although the two terms are essentially interchangeable. A federation execution is the actual operation, over time, of a set of federates interconnected by an RTI.
In principle, a FOM exists only for the duration of the "specific objective", typically a scheduled exercise. When that is the case, it does indeed make sense to generate the FOM as the intersection of the SOMs of the federates involved. However, in practice FOMs tend to lead a continuing existence of their own, either because the exercise is repeated at regular intervals, or because an all-inclusive FOM is desired in order to avoid having to edit it despite a changing roster of federates. In particular, a FOM greater than the intersection of the *actively* participating federates makes sense when *passive* federates are used to capture (for analysis logs or for after-action reviews, for example) what happens during the federation execution. In such a case, the passive federate needs to be able to subscribe to objects and interactions (see Synthetic Environment, below) that spend their entire lifespan (i.e. are created, managed and destroyed) within a single federate’s purview. Even though no collaborative simulation occurs, the objects of interest must nevertheless appear in the FOM for the passive federate to accomplish its task.

All communication and interaction between the federates goes through the RTI; private direct connections are not allowed. The representations of the objects in the federation execution are all held by the federates: the RTI does not cache the data that transits through it.

**Figure 4.** The intersections of the object models

*The intersections of the simulation and federation object models (SOM and FOM) dictate the possible collaborations. The three-colour area represents the object model understood by both simulations and the federation; only those objects are susceptible to collaborative simulation by the simulations during the federation execution.*
**Synthetic Environment**

The HLA synthetic environment is populated by durable entities called *objects*. In a military context, the simulation objects are usually representations of real-world objects: soldiers, vehicles, buildings and so on. More abstract objects are possible and indeed often desirable. For example, a simple tactical constructive simulation could have a "Weather" object, which could be little more than a holder for a handful of meteorological parameters such as temperature, humidity, wind speed and wind direction. Other objects could represent more abstract concepts, such as groups of entities (organized or not): a military unit or a crowd of protesters.

The object models define *classes* of objects, which are nothing more than a collection of *attributes* that serve to define the object's state. In other words, object classes are *data structure templates*. In object-oriented software design, objects are normally defined as encapsulations of data and operations (*methods*); in HLA, object classes have no methods: they are defined entirely by their identifying attributes. The behaviours and operations that affect the values of HLA object attributes are kept resident in the federates. The HLA supplies the *HLAObjectRoot* class, from which all other object classes descend in hierarchical fashion. Subclasses inherit their superclass’s attributes, naturally.

*Figure 5. A subset of the RPR-FOM version 2.0 object class hierarchy. The attributes are not shown.*
During a federation execution, instances of the various object classes may be created and deleted by each federate. A federate declares its intention to publish and subscribe to an object class’s attributes (or a subset thereof). When an object instance is created by a federate, the RTI considers that federate as owning the instance attributes that it publishes for that class (any remaining attributes are unowned). Ownership of an instance attribute means the owner is responsible for keeping that instance attribute’s value up to date, and for supplying that value when it is requested by the RTI. Subscribers discover object instances when they are pertinent to their subscription interests, and may, as needed, request that the RTI supply them with the instance attribute values. The RTI truly earns its title of middleware in this circumstance, since all it does is request the values from their respective owners and then dispatch the owners’ subsequent value update.

The RTI also mediates ownership transfers between federates. Divestiture of an owned instance attribute can be initiated by the owning federate with varying degrees of assertiveness (If Wanted, Negotiated and Unconditional) or it can be requested by the RTI. Conversely, acquisition of an unowned instance attribute may be requested by the federate with varying degrees of forcefulness (If Available, Negotiated) or it can be offered by the RTI. The section entitled Ownership Management (below) discusses this delicate matter in detail.

Interaction is a concept closely related to the object. Interactions are also described as a hierarchy of classes, descending from HLAInteractionRoot. Their attributes are called parameters, and the key difference is that instead of creating instances, the federates send occurrences. Interactions have no duration: they are created at a given instant, distributed like an object instance attribute update, and then cease to exist. They are meant to represent simulation events such as a lightning flash or a direct-fire engagement. Note that SOMs can disagree on what should and should not be an interaction. Let’s take the example of a burst of machine gun fire aimed at an infantry platoon. A simpler simulation would likely model this direct-fire engagement as an interaction, whereas a more detailed simulation could choose to create object instances for each bullet.

Despite their name, interactions don’t necessarily involve more than one object instance. A FOM could very well include interactions which are nothing more than running commentaries, destined for eventual consumption by event loggers. For example, an event could be designed that corresponds to the decision by one entity to engage another. This “event” has no immediate physical consequences since it occurs “inside the head” of the entity, and could be of interest only while debugging the entity’s target selection algorithm.
Each simulation is free to have object models of varying degrees of detail. Indeed, a sizable part of the federation development process (FEDEP) consists of deciding what level of detail is required and which federate is most suited to the task, as well as reconciling object model differences. To facilitate this process, the Real-time Platform Reference FOM (RPR-FOM) is published by the Simulation Interoperability Standards Organization (SISO). SISO-STD-001-1999, RPR-FOM v1.0 ([26] SISO 1999a, [27] SISO 1999b) captures the functionality of DIS (IEEE 1278.1-1995), and facilitates interfacing a DIS simulation with an HLA federation. RPR-FOM version 2.0 is intended to add the functionality of DIS 2.0 (IEEE 1278.1a-1998) and to update version 1.0 to the IEEE 1516 HLA standard; it is in its seventeenth draft as this is being written ([28] SISO 2003). RPR-FOM version 3.0 is in early draft form and is intended to include features that were originally considered for DIS 3.0 (now abandoned).

A federate could subscribe to an object or interaction class for which there is no publisher; if another federate publishes a superclass of the subscribed class, that is what the first federate will discover. In that sense, HLA is polymorphic. All it really means is that you may not get all of the attributes or parameters you subscribed to, since successive subclasses can only differ by the monotonic accretion of attributes/parameters.

Dimension rounds out the set of key HLA concepts. It will be discussed in the section on data distribution management below.

**HLA Interface**

The application programming interface between a federate and an HLA RTI consists of two software objects: the RTI Ambassador and the Federate Ambassador. The RTI Ambassador instance is supplied by the RTI and exposes a number of methods, called services, to the federate (documented in IEEE 1516.1-2000). The federate sends “commands” to the RTI by invoking the RTI Ambassador services. Some services return a value immediately (a Boolean, an attribute handle set, etc.) but most respond asynchronously through the Federate Ambassador. The IEEE 1516 specification, unlike the older specifications (1.3 and predecessors), does not prescribe how the RTI Ambassador instance is to be procured. This allows multiple RTIs to cohabit in the same name space, but it does mean a federate application will probably need a thin adaptor class in order to be able to fetch whichever RTI is installed without needing to be recompiled.
The interfaces are specified in terms of services (methods), designators (names and handles) and abstract data types only. The mappings to specific programming languages are supplied in three annexes, covering respectively Ada 95, C++ and Java. The mapping used by a given federate has no consequences on the rest of the federation; for example, a designator encoded using one mapping will be decodable using any other mapping. Since the JACK Agent Language is an extension of Java, we naturally chose to restrict ourselves to Java in what follows.

**HLA Services**

For convenience, the HLA interface methods are grouped into seven *service groups*, which we will discuss in varying degrees of detail below.

**Federation Management**

These services deal with the creation, dynamic control, modification, and deletion of a federation execution. Create/Destroy Federation Execution are used to create and destroy federation executions. Join/Resign Federation Execution are used by the federate to join a federation execution and withdraw from it when the execution ends.
Federation Execution does not exist

Create Federation Execution

Federation Execution exists

Join Federation Execution

Destroy Federation Execution [no joined federates]

Joined Federate

Figure 7. The global federate states and their transitions
This diagram and the following ones use the ([29] Harel 1987) statechart notation and are derived from the HLA IEEE 1516 state diagrams. The dot is the initial state. The bracketed term is a Boolean guard expression; here it shows the federation execution may be successfully destroyed only if there are no joined federates.
Synchronization points may be declared by the Register Federation Synchronization Point service. Optionally, knowledge of the synchronization point’s existence can be restricted to a subset of the joined federates. The RTI responds with either a Synchronization Point Registration Succeeded or a Synchronization Point Registration Failed callback (the latter includes a reason for failure parameter, because the second form of the Register Federation Synchronization Point service can fail in two distinct ways). The semantics of the synchronization points are documented in the FDD but the RTI does not (indeed cannot) enforce them. A typical use would be a “StartingLine” synchronization point (to use a racing metaphor). As federates join the federation execution and complete their initialization process (which may include interactions between the federates, mediated by the RTI), they would be advised by the RTI of the synchronization point’s existence through the Announce Synchronization Point callback, and would be expected to eventually invoke the Synchronization Point Achieved service. Once all required federates have thus acknowledged the synchronization, the RTI announces this state of affairs to each federate through the Federation Synchronized callback, in this case firing the “starting gun.” Once the Federation Synchronized callbacks have all been issued, the synchronization point ceases to exist (and may therefore be registered anew).

Federation Synchronization points can also be used whenever some form of synchronization is desirable between multiple joined federates. They can be used to let the RTI arbitrate potential race conditions between federates. Whenever synchronization is required between the multiple threads of an application, programming languages or frameworks provide a variety of synchronization and exclusion classes and mechanisms, variously named semaphores, mutexes (contraction of “mutual-exclusion”), or monitors. In broad terms, the protected object or block of code has an associated lock which can be owned by only one thread at a time. When a race condition occurs, one thread succeeds on its lock operation while the others block on it. When the winner concludes the protected block and relinquishes the lock, the system awards it to one of the blocked threads, and so on.

The synchronization services can mimic this. Obtaining a lock corresponds to the successful registration of a synchronization point: if the operation succeeds, the federate “has the lock.” The other federates are expected to achieve the synchronization as soon as it is announced to them (acknowledging the lock). Other federates that were simultaneously attempting to register the lock (presumably they would not try if they were already aware of the lock’s existence) are expected to put themselves in whatever wait mode is appropriate. The lock’s owner relinquishes the lock by achieving the synchronization in its turn (most likely it will be the last federate to do so, but this is not a requisite). When this occurs, the RTI invokes Federation Synchronized for all federates, which signals that the lock has been freed up (i.e. the synchronization point no longer exists). By varying this pattern slightly, semaphores can also be mimicked (e.g. announcement of the synchronization point’s existence is the signal being awaited).
Figure 8. The joined federate states
Transitions marked with a dagger (†) are FederateAmbassador callbacks. The synchronization point services do not change the state of the federate itself, although they do change the federation state.

*Federation saves* complete this set of services. A federation save stores a complete description of the federation execution in such a way that the federation execution may be restored later. Saves are identified by a label, and can be generated at an optional specific logical time. It can be presumed to consist of a set of files distributed between the joined federates on the one hand, and the RTI on the other (the form the storage takes is left as an implementation issue for the RTI and federates). Besides the save label, the RTI stores the number of federates joined at the time the save is created, and the types of the various joined federates (these types are simply labels supplied by the federates when they join the federation execution; their semantics are left to the federation). Unlike synchronization points, saves always affect the entire set of joined federates. A federate is expected to use the save label, the name of the federation execution (and, implicitly, the FDD), its joined federate designator and its federate type to distinguish the saved state information.
Any federate may Request Federation Save. The RTI sends the Initiate Federate Save callback to each federate, including the requestor. Each federate then complies by invoking Federate Save Begun and later Federate Save Complete (specifying if it was successful or not). As soon as the federate enters one of the “Federate Save in Progress” states, the RTI withholds most callbacks and refuses to accept the overwhelming majority of the various service invocations from the federate—those that would change the state of the federation in some way. As the save progresses, a federate may Query Federation Save Status, to which the RTI responds with the Federation Save Status Response callback. Eventually the RTI announces to the federates that the save is complete (successful or not) through the Federation Saved callback, and normal execution can then resume.
The reverse of the save process is the federation restore process. This begins with a Request Federation Restore, which the RTI acknowledges with the Confirm Federation Restoration Request callback (specifying if it was successful or not). Using the Federation Restore Begun callback, the RTI signals each federate (including the requestor) that it has been put in the “Federate Restore in Progress” state, which restricts federate activity just like during a federation save. Note that the requestor enters the suspended state as soon as it receives confirmation of the request. Next, the RTI requests of each federate that it begin to restore its state as previously saved, using the Initiate Federate Restore callback. The federate signals the completion of its restore process through Federate Restore Complete (specifying if it was successful in doing so or not) and may Query Federation Restore Status, receiving a Federation Restore Status Response callback in response. Finally, the RTI advises each federate of its return to normal operation through the Federation Restored callback.

Because federation saves are implementation-specific, there is no requirement that a save created by one vendor’s RTI be restorable by another vendor’s RTI. Also, federates that share the same federate type must be able to handle each other’s save documents. For example, if federates A, B, and C were all of the same type when the save was created, when the RTI processes a restore request the only guarantee is that it will assign each of the federates one of those three roles (see Figure 10). Federate A could therefore end up restoring its own save data, or B’s or C’s. This strongly suggests a certain level of storage sharing for save/restore.
Figure 10. Interchangeability of federates of the same type

Top: the federation is saved and the save data is spread between the stores (in grey). When the restore occurs (bottom), the federate instances $B_1$ through $B_3$, all of the same type, may be assigned any of each other's original roles.

### Declaration Management

These services deal with the publication of and subscription to object and interaction classes. This is accomplished by the [Un]Publish/Subscribe Object Class Attributes and [Un]Publish/Subscribe Interaction Class services. Note that publication is at the object class attribute level in the former case, but at the interaction class level in the latter: you cannot subscribe to or publish just a subset of an interaction's parameters.

The only (optional) RTI callbacks are Start/Stop Registration For Object Class and Turn Interactions On/Off. These are known as the Object Class and Interaction Relevance Advisories, respectively. They are used to allow a federate to more intelligently manage its publication activities. In effect, the Object Class Relevance Advisory lets the federate know whether or not any other federates are actively subscribing to a published set of object class attributes. If there are none, there is no point in registering new object instances (or sending new interactions). This can help with federation execution performance.

A small subtlety here is the possibility of passive subscriptions; these do not trip the advisories and are designed to allow special federates like loggers or viewers, which do not contribute actively (i.e. do not own anything or send interactions), to join a federation execution freely without modifying its behaviour.
Object Management

These services deal with the registration, modification, and deletion of object instances and the sending and receipt of interactions. The bulk of the "action" of a federation execution is expected to occur here.

![Diagram of state transitions](image)

**Figure 11.** Known/Unknown transition state diagram

Part of the state diagram for object instance attributes, focusing on the known/unknown transitions. The details of the owned/unowned transitions (*) are given in Figure 12. Here i stands for an object class, j for an object attribute and k for an object instance.

The first service in this group is Reserve Object Instance Name, to which the RTI responds through the Object Instance Name Reserved callback. All object instances, as they are registered, are given federation-wide unique names automatically by the RTI. The name reservation service allows a federate to reserve a name, something which is typically done for special object instances that have been deemed necessary for the federation execution design.

Next is Register Object Instance, which may trigger Discover Object Instance at other federates. This is how object instances are injected into the federation execution. The converse service is Delete Object Instance, which will trigger Remove Object Instance at the federates that had previously discovered the object instance. A non-owning federate can also invoke Local Delete Object Instance, whose purpose is simply to "forget" about the object instance. This could be because it has gone out of scope and is no longer of any interest to the federate, or it could be because the federate wants to shift its subscription interests for already-discovered object instances (e.g. an object was previously discovered as one of its superclasses and the federate wants to rediscover it as a more specific subclass, or vice versa).
Then we have Update Attribute Values, which is how an owning federate apprises the RTI of the new values of instance attributes. The update policy for each class attribute is documented in the FDD but is not enforced by the RTI. The updated values are distributed to the subscribing parties through the Reflect Attribute Values callback. A subscriber may invoke Request Attribute Value Update, which will lead the RTI to prompt the publisher with a Provide Attribute Value Update callback. Note that a request for a value update for an unowned instance attribute won't be answered, since no custodian of the instance attribute value exists. This is because the RTI does not cache the "last known value."

Interactions, because of their transient nature, combine Register and Update into a single service, Send Interaction. Subscribers receive the Receive Interaction callback.

The RTI supports two built-in transportation types: "reliable" and "best effort", typically mapped to Transmission Control Protocol/Internet Protocol (TCP/IP) and User Datagram Protocol (UDP), respectively. New ones may be defined if needed. The transportation type is specified by the FDD and may be changed at the instance attribute or interaction class level through the Change Attribute/Interaction Transportation Type services.

Finally, there are two sets of advisories associated with this service group. They serve to let publishers and subscribers know whether or not they have counterparts in the federation. Like the declaration management advisories, they serve to let federates optimise their use of bandwidth—there is no point in sending instance attribute updates regularly if there are no subscribers currently listening. The Attribute Relevance Advisory causes the Turn Updates On/Off For Object Instance callbacks at the publisher. Its converse, the Attribute Scope Advisory, causes the Attributes In/Out Of Scope callbacks at the subscriber. A published attribute goes out of scope if it has no subscribers; a subscribed attribute goes out of scope if it has no publisher (i.e. is unowned). This notion of scope is of limited usefulness until the data distribution management services are considered, as we will see later.

Ownership Management

These services deal with the transfer of ownership of instance attributes among joined federates. HLA departs from conventional object-oriented design in allowing the attributes of an object instance to be owned in a distributed fashion, rather than being under the exclusive responsibility of the object owner. This is a powerful feature that greatly facilitates the cooperative modelling of objects.

The root object, HLAobjectRoot, has just one attribute, HLAprivilegeToDeleteObject, inherited by all HLA objects. This attribute has no value (content); the only thing that matters is which federate owns it. The federate that owns this attribute "owns" the object instance, in the sense that it is the sole federate allowed to delete the object instance.
To give a simple example of what attribute ownership means, consider a missile ordnance object. Before the missile’s launch, it is attached to its launcher (e.g. under the wing or in the bomb bay of an aircraft, in a ship’s missile storage compartment, in a man-portable launch tube, etc.). HLA ownership management allows the missile’s geographical coordinates attributes to be the responsibility of (to be owned by) whichever simulation moves the launcher about. It is only once the missile is launched that ownership of these attributes needs to be transferred to the federate responsible for the missile’s flight.

HLA defines the Management Object Model (MOM), a small hierarchy of objects and interactions which are part of every FOM. These are owned by the RTI and represent the federates and their relation to the RTI; they are mostly used for network oversight, performance tuning, and troubleshooting. The RTI never accepts ownership of anything else, and never relinquishes ownership of any part of the MOM.

Ownership is crucial within an HLA federation, because the owner of an object instance attribute is the sole custodian of its value. While an attribute is unowned, its value cannot be obtained from the RTI: each federate is left with its locally cached “last known value”, if any. Ownership is also tied to discovery: if an object instance ends up an orphan (that is to say, none of its attributes are owned by any federate and it is currently not known by any federate), it cannot be discovered at all.

Ownership management is a delicate process, as the associated state diagrams show (see Figure 12 and Figure 13). A federate starts off owning all of the published attributes of an object instance it registers (a minimum of one attribute). It can divest itself of ownership in four ways: through the Unconditional Attribute Ownership Divestiture service, the Negotiated Attribute Ownership Divestiture service, the Attribute Ownership Divestiture If Wanted service, and by unpublishing the owned object’s class. Conversely, it may acquire ownership in two ways: through the Attribute Ownership Acquisition service, or the Attribute Ownership Acquisition If Available service.
The Attribute Ownership Acquisition If Available service is “non-intrusive” (like a passive subscription) and is immediately resolved by the RTI based on current ownership availability. The requesting federate receives an Attribute Ownership Acquisition Notification for the instance attributes acquired and an Attribute Ownership Unavailable for the others.

The Attribute Ownership Acquisition request, on the other hand, puts the federate in the “acquiring” state (with respect to the specific instance attribute), whence it may return through the Cancel Attribute Ownership Acquisition service. The RTI acknowledges this cancellation through the Confirm Attribute Ownership Acquisition Cancellation callback. This form of acquisition is “intrusive” (like an active subscription) because the RTI will prompt the owner (if not already divesting) through a Request Attribute Ownership Release callback. The Attribute Ownership Unavailable callback does not occur in this context, but several Attribute Ownership Acquisition Notifications may occur depending on the number of previous owners involved and their ownership release schedules. Note that the Attribute Ownership Acquisition If Available service will not return the federate from the “acquiring” state since it is a weaker request than the outstanding Attribute Ownership Acquisition.

Figure 12. Owned/Unowned transition state diagram
Part of the state diagram for object instance attributes, focusing on the owned/unowned transitions (the asterisk in Figure 11). The circled H indicates the historical state (meaning the service may be invoked from any one of the inner states and returns whence it was invoked). The details of the Not acquiring/Acquisition pending transitions (the asterisk here) are given in Figure 13.
When an Unconditional Attribute Ownership Divestiture occurs, the federate immediately ceases to own the instance attribute. It may become unowned, or the RTI may immediately grant ownership to some other federate that had previously requested Attribute Ownership Acquisition. If there are no such “acquiring” federates, other publishers are prompted by the RTI through the Request Attribute Ownership Assumption callback. This is the only circumstance under which the RTI will actively pester the federates to assume ownership.

With the Negotiated Attribute Ownership Divestiture service, the federate puts itself in the “divesting” state, whence it may return through the Cancel Negotiated Attribute Ownership Divestiture service. If other federates are already in the “acquiring” state (failing that, the RTI will prompt potential new owners as in the unconditional case), the divesting federate receives a Request Divestiture Confirmation callback. It then confirms the transfer of ownership by invoking the Confirm Divestiture service. The federate can also leave the “divesting” state by unpublishing the attribute or by divesting it in one of the other two means mentioned earlier.

**Figure 13. Not Acquiring/Acquisition Pending transition state diagram**
Part of the state diagram for object instance attribute ownership, focusing on the Not acquiring/Acquisition pending transitions. The left-hand acquisition pending state represents non-intrusive acquisition, whereas the right-hand states represent intrusive acquisition.
With the Unpublish Object Class Attributes service, the federate immediately ceases to own any attribute of every single instance of the unpublished object class; unlike Unconditional Attribute Ownership Divestiture, the RTI will not offer ownership to eligible federates other than those actively pursuing acquisition.

Ownership management is rounded out by the Query Attribute Ownership service, to which the RTI responds through an Inform Attribute Ownership callback. A simpler form of the query is the Is Attribute Owned By Federate service, which returns a Boolean immediately.

**Time Management**

These services and associated mechanisms provide a federation execution with the means to order the delivery of messages (service and callback invocations) throughout the federation execution according to a *logical time scale*. The federation execution has a single global “clock” which tracks the federation time. This time is advanced in cooperative fashion by the various federates, and serves to preserve causality. The only way the federation clock may be turned back is through the save/restore facility. The description of the services that make up this group will be kept to a minimum here, as these services were not the focus of this work and have been better described by others (see for example [30] Fujimoto and Weatherly 1996, [31] Carothers et al. 1997, [32] DISTI 1997).

Messages are delivered according to two ordering types: receive order and time-stamp order. The former ordering type is “timeless” in the sense that it ignores logical time considerations and is delivered as soon as possible in real time. Similarly, federates may be *time-regulating*, *time-constrained*, both or neither (this status can change freely during execution). Time-regulating federates are the only ones sending time-stamped messages, whereas time-constrained federates are those receiving time-stamped messages. The actual ordering type of a sent or received message depends on the status of the involved federates, the message’s preferred ordering type, and the specific variant of the service used (i.e. whether or not a time-stamp argument is provided).

Apart from time management services, relatively few services can accept a time-stamp argument: Request Federation Save, Update Attribute Values, Send Interaction and Delete Object Instance. Note in particular that ownership management is not time-stamped, which means precautions must be taken to avoid odd situations where instance attributes are, for a given logical time, owned by either no federate or several federates.

In part to allow optimistic simulations (where some of the federates anticipate events by processing them ahead of time so that the results of their computations are readily available when the events do occur), the time management services provide an *event retraction* capability: within some logical time constraints, a time-regulating federate may send messages time-stamped in the federation’s future and later Retract them. Retracted events will either be simply removed from other federates’ queues (if not delivered yet) or cause the affected federates to receive a Request Retraction callback.
A key concept of HLA time management is the federate lookahead. This is a span of logical time, extending from the federate’s current logical time value into the future, within which the federate may not send time-stamped messages. It can be conceived as the “thickness” of the federate’s “now”, or as the federate’s “reaction time.” Its purpose is to avoid a variety of time deadlock situations. It is modifiable through the Modify Lookahead service.

A federate signals its intent to move its clock ahead through the Time Advance Request [Available], Next Message Request [Available] and Flush Queue Request services. This switches the federate from the “time granted” state (the normal operating state) to the “time advancing” state. The Time Advance Grant callback completes the process by switching the federate’s state back to “time granted.”

Federates change their time status with the Enable/Disable Time Regulation/Constrained services. The Enable services are acknowledged by the callbacks Time Regulation/Constrained Enabled (the Disable services automatically succeed, so no acknowledgement callbacks are required). The preferred ordering type of attribute updates and interactions are modifiable through the Change Attribute/Interaction Order Type services. Whether receive-order messages may be received while in the “time advancing” state is controlled through the Enable/Disable Asynchronous Delivery service.

Finally, the federate can query the RTI about a variety of time management parameters through the Query Lookahead, Query Logical Time, Query GALT (greatest available logical time) and Query LITS (least incoming time stamp) services.

**Data Distribution Management**

These services provide a set of powerful filtering mechanisms allowing the exclusion of irrelevant messages at the instance attribute or interaction occurrence level. Both producers and consumers may restrict their publishing and subscribing distribution regions according to a federation-defined set of dimensions. The RTI ensures that message delivery occurs only when the two sets of regions actually overlap. An example would be interaction classes that reflect command and control radio traffic: their publication would probably be restricted to a small range (see below) over the frequency dimension (the communication’s bandwidth), while an electronic warfare listening post would restrict its subscription to the bandwidth it is currently listening to. Only if the two ranges overlap is an intercept possible.

Each of the federation-defined dimensions is an independent axis along which ranges may be defined as needed. They are defined by an included lower bound of zero and an excluded upper bound which can be any integer type. Typical examples of dimensions are latitude, longitude, altitude, and broadcast frequency. HLA automatically provides the default region, which encompasses all of the defined dimensions in their entire range; in other words, it is the universal nothing-gets-filtered-out filter. Unless data distribution management (DDM) is used to modify object and declaration management services, these are assumed to use the default region.
In the FDD, each object class attribute or interaction may be associated with any number of dimensions, depending on the federation’s needs for filtering. This set of dimensions is called the attribute’s (or interaction’s) available dimensions. It is important to realize that an attribute’s dimension, in the physical sense, has nothing to do with the DDM dimension. For example, any attribute of a vehicle class that is likely to be of interest to other federates should be associated with the longitude dimension in order to allow spatial filtering, not just the longitude attribute.

Each dimension defines a default range, which can be an actual range or an exclusion indicator. Note that a dimension’s default range does not affect the extent of the default region. Default ranges come into play when realizing regions, as we’ll see shortly.

Figure 14. Dimensions, ranges, regions, and region sets
Overlapping attribute publication (red) and subscription (green) region sets are also shown.
The Create Region service expects as argument a set of specified dimensions, and returns a region template. The federate’s choice of specified dimensions signals to the RTI its intent to specify a range (a single continuous interval) over each of those. Once the bounds have been set for each of the specified dimensions, the Commit Region Modifications service may be invoked to create a region specification. The range bounds may be modified later on, again using the Commit Region Modifications service. This allows regions to be dynamically modified as simulated time elapses.

A region template or specification may be deleted using the Delete Region service. Regions that are “in use” (that have been realized, as we’ll see below) may not.

Region ownership is not transferrable: only the region’s original creator federate may delete it, modify it, or use it to send an interaction, or to publish or subscribe to attributes. The Receive Interaction and Reflect Attribute Values callbacks may, if the receiving federate wishes it, include the regions used by the sender. This allows the receiving federate to inspect those regions, possibly in order to create duplicate regions of its own.

![Diagram showing transitions between region templates, specifications, and realizations]

*Ranges of unspecified dimensions are filled in from default ranges (unless excluded)*

**Figure 15.** Transitions between region templates, specifications and realizations
Region specifications become *region realizations* when associated with an object instance attribute, an object class attribute or an interaction class. Unspecified dimensions that are members of the available dimensions and that had default ranges are added by the RTI to the region realization. The region’s *contained dimensions* is the set of dimensions for which the region defines a range. Note that a given region specification may give rise to a number of different region realizations, depending on the attributes or interactions it is associated with. Note also that federates can designate region realizations only in one instance: during a Reflect Attribute Values or Receive Interaction callback, the RegionHandleSet received (if any) consists of copies of the region realizations used by the sender when the attribute value update or interaction was sent. They are read-only, and cease to exist once the callback concludes.

A region cannot be associated with an attribute or interaction if overspecified. That is to say, the region’s specified dimensions must be a subset of the attribute’s (or interaction’s) available dimensions.

Figure 16 illustrates the relationships between the various dimension sets described above. The federation defines the outer set, the defined dimensions. A region’s specified dimensions and an attribute/interaction’s available dimensions are independent subsets of the defined dimensions. For a region to be applicable to an attribute/interaction, however, its specified dimensions must be a subset of the available dimensions, which is what is shown here. The defined dimensions can be subdivided into those dimensions that have a default range (on the right) and those that don’t (“excluded” dimensions, on the left). When a region realization occurs, by associating an attribute/interaction with an applicable region specification, the resulting contained dimension set is the one filled in yellow.
The fact that each region defines at most one range over each dimension (thus describing a hypercube of sorts) is quite limitative, which is why publications and subscriptions are defined over sets of regions. As Figure 14 shows (in red), a fair approximation to a two-dimensional circle can be achieved by defining a suitable set of rectangular regions. Region sets can also be used to describe discontinuous (broken up) volumes.

For an interaction to be received, for an object instance to be discovered and for attribute value updates to be reflected, the publication and subscription regions must overlap. Overlap is determined by looking at the publisher’s and subscriber’s region sets. There is overlap between the sets if at least one publisher’s region overlaps one subscriber’s region. A given publisher region overlaps a subscriber region only if they have at least one contained dimension in common and if the ranges overlap for each contained dimension common to the two regions.

Figure 17 shows a crude example where a radar has defined a subscription region set that roughly represents its coverage volume using the radar cross-section and casting dimensions (not shown are the northing and altitude dimensions). An aircraft object defines its publication region based on its current position and radar cross-section. When the two region sets do not overlap, the radar federate does not discover the target object and does not have to spend any computation time rejecting the potential detection.
Figure 17. An example of overlapping publication and subscription region sets

Figure 18 shows the converse, where the radar has defined a publication volume using the radiated power and easting dimensions. The aircraft object defines its subscription volume based on its current position and radar warning device threshold. While the two region sets do not overlap, the aircraft federate does not waste time triggering its radar warning device algorithm.
The declaration management services modified by DDM are the subscription services only. The object management services modified are the object registration, interaction sending and attribute value updating services. The unmodified services behave as if they were using the default region. Whenever an attribute is without any associated regions, it reverts to the default region. Conversely, whenever at least one region is associated with an attribute, it is unassociated from the default region.

Register Object Instance With Regions combines atomically the Register Object Instance and Associate Regions For Updates services, preventing the newly declared object from having its published attributes associated with the default region, however briefly.
The region creation process involves identifying the various handles involved (federates initially know only the names of dimensions, object classes and so on, and must therefore use a variety of Support Services to accomplish this translation), creating a dimension set from the subject's available dimensions, invoking the RTI's Create Region service (a region factory) to create a region having those specified dimensions, and modifying each specified dimension's range as needed. Once this process is complete for each of the regions making up the set, the set is committed through the Commit Region Modifications service, and can then finally be used in an attribute association or subscription sending. The region set may be modified and the changes committed again as necessary; a typical usage is to have various publisher or subscriber “footprints” move over the battlefield's dimensions (detection and coverage volumes, transmission bandwidths, and so on).

Additional regions may be associated and unassociated freely with the attributes. Once a region is no longer needed (that is to say, it is no longer associated with anything), the Delete Region service may be used to get rid of it.

Ownership services are indirectly affected by DDM. Whenever instance attribute ownership is transferred between federates, all of the original owner's region associations lapse. To preserve region associations during ownership transfer, the prospective new owner must, as part of its acquisition process, associate the instance attributes beforehand. These associations have no effect until ownership is actually transferred.

The Attribute Relevance and Attribute Scope advisories are directly affected by DDM, unlike the Object Class Relevance and Interaction Relevance advisories. Whenever an instance attribute publication region set ceases or begins to overlap a subscription region set, the publisher may get the former advisory while the subscriber definitely gets the latter advisory. The publisher's Attribute Relevance advisories are less frequent because they only occur when the status changes between “no subscribers” and “some subscribers”; they are not triggered by changes in the make-up of the set of subscribing federates. The subscriber's Attribute Scope advisories are more frequent but are more specific: only those subscribers that are concerned get them.

**Support Services**

This service group includes miscellaneous services used to perform name-to-handle and handle-to-name transformations (for object classes and instances, attributes, interaction classes, parameters, dimensions, transportation and ordering types), to change advisory switches (object class relevance, attribute relevance and scope, interaction relevance), to manipulate regions, and to control callback invocations. There is no need to detail these services here except to mention that none of them give rise to callbacks: they all return a result immediately if pertinent. As a general rule, these are the only services that can be invoked from within the federate service thread.
Threads and Process Models

Neither the IEEE 1516 specification nor the HLA 1.3 specification imposes a process model. However, the standard HLA 1.3 implementation, the DMSO 1.3NG RTI, did impose indirectly a single-threaded or asynchronous process model through the `RTIambassador tick()` service, which was expected to be “invoked frequently.”

There are three possible process models: single-threaded (also called polling), asynchronous and multi-threaded. Each specifies when to devote processor time to the local RTI component (LRC). The LRC is the part of the RTI that is responsible for transmitting the federate’s service invocations to the RTI at large, for invoking the federate’s callbacks in response to RTI messages, and for maintaining the RTI’s MOM objects and interactions.

In the single-threaded or polling process model, the federate application is responsible for yielding process time to the LRC through frequent `tick()` invocations. The LRC executes only while the federate waits for the `tick()` invocation to complete. If one considers the federate’s code and the LRC’s code, execution occurs at only one spot of either one at any time (there is a single “cursor” shared between the two blocks of code).

In the asynchronous process model, there are two concurrent threads that do not intermingle, one for the federate and one for the LRC. The federate is responsible for signaling when it is ready to receive callbacks through the `tick()` invocation. Continuing the preceding allegory, there are two “cursors”, one in each block of code (the federate’s and the LRC’s) and both execute at the same time, except when the LRC’s cursor “invades” the federate’s code in the body of the latter’s callbacks.

In the multi-threaded process model, finally, the LRC runs in its own separate thread (the federate service thread) and can invoke callbacks at any time. There is no `tick()` service. Continuing the allegory, there can be multiple cursors in either block, and the LRC’s cursors can freely scan the federate’s code at any time.

In HLA 1.3, there was a specific exception, `ConcurrentAccess`, which was thrown when a `FederateAmbassador` callback tried to invoke an `RTIambassador` service. In other words, the LRC accepted `RTIambassador` invocations only when not busy invoking a federate callback. This exception disappeared from the IEEE 1516 specification. However, Pitch pRTI1516, despite being multi-threaded, still behaves in a similar fashion: to ensure consistent sequencing of callback deliveries, each federate is assigned a single federate service thread, and while this thread is busy invoking a federate callback, most `RTIambassador` invocations will cause an `RTIInternalError: Concurrent access attempted to <method name>` to be thrown. This apparently serves to ensure the federate does not undergo any state transitions during a callback; in other words, the RTI treats federate callbacks as atomic.
In any case, making a federate multi-threaded is a tricky matter, since a new federate callback invocation can occur at any point (except during a callback invocation in the Pitch pRTI1516 case). Assertions regarding the federate’s state are only valid instantaneously unless precautions are taken. These typically involve setting up various synchronization semaphores in order to protect critical sections of code. But such precautions entail their own share of problems, such as the risk of deadlock.

**Synchronization Issues**

If we assume a fully multi-threaded RTI, the only thing that we can be sure of is that the callbacks will not occur concurrently: the LRC will evoke each callback in sequence from a single Federate Service Thread. Even that may change in some future RTI implementations. The callbacks themselves are not guaranteed to be received in any particular order (because of network delays, among other things) unless time management is used and several conditions are satisfied at once. For a message to be sent and received in time-stamped order (TSO):

- It must be an attribute value update, object deletion or interaction occurrence (note in particular that ownership negotiations are not susceptible to time-stamping);

- The interaction class or object instance must have a preferred order type of TSO in the FDD (if not, this may be changed, locally to the federate, through the Change Attribute/Interaction Order Type services, on a per-instance basis in the object case);

- The sending federate must be time-regulating;

- The time-stamped form of the service must be used by the sending federate; and

- The receiving federate must be time-constrained.

Sequences of related messages can reasonably be expected to be received in order, either because the RTI implementation makes sure of this, or because the main factor responsible for scrambling message arrival times (i.e. network jitter) is likely to be relatively constant for messages fired in quick sequence from a single origin. For example, an object instance will be Discovered before the related Attribute Scope Advisory is received, and this in turn will precede any Attribute Value Update Reflection.

What is much more likely to occur are mutually unrelated messages, such as interactions, attribute value updates of different object instances, ownership transfers and so on. It is tempting to have the federate devote a thread to each of these unrelated objects, but the overhead associated with constant context switching may be too costly, depending on the application. Some form of message dispatching is clearly desirable. The simplest form has the Federate Service Thread perform crude validation on the messages as they arrive (because the Federate Ambassador exceptions can only be thrown from that vantage point) and then store them in a queue for later processing by another thread or threads.
The queue may be a simple first-in first-out (FIFO) implementation, with a single worker thread grabbing each message in turn, or it could be more sophisticated. One possibility is to sort the messages into separate queues by type (one queue for object discoveries, another for object removals, and so on), allowing various priority schemes to be readily applied. Another possibility is to sort the messages by simulation aspect: there would be one queue for each interaction class, another for each object class, another for each object instance, maybe even another for each object instance attribute. Yet another possibility would be a mixed scheme.

Another important design consideration is the tracking of simulation-related states. The federate needs to know what it has subscribed to, through which regions, and so on. This can be filled in from the various MOM HLA report interactions, or it can be tracked from the federate's end. For example, to find out whether an object instance attribute is subscribed to or not, one can either have a flag attached to the internal representation of that attribute and change that flag as subscriptions are established and withdrawn, or one can send an HLA request Subscriptions interaction, to which the RTI responds with an HLA report ObjectClass Subscription interaction for each object class one is subscribed to. Once the one with the correct HLA objectClass is received, its HLA attribute list field can be scanned to see if the attribute in question is there or not. As can be seen, considerable overhead is involved in the latter approach, not to mention message traffic. In any case, some state information is missing from the MOM; for example, there is no way to find out whether the subscribed attribute instance of the example is currently in scope or not, nor which regions the subscription is currently being filtered through.

In our Chat Clients, we ended up tracking whether object instances were owned or not, whether they were subscribed to or not (at the class level), whether they were up for divestiture or not, and whether they were in scope or not.

**Pitch pRTI1516**

According to the DMSO RTI Verification Status Board [https://www.dmsob.mil/public/transition/hla/rti/statusboard](https://www.dmsob.mil/public/transition/hla/rti/statusboard), the only verified IEEE 1516 implementation is Pitch SA’s pRTI 1516. There are only two others currently undergoing verification: MÅK Technologies Inc.’s High Performance RTI and Magnetar Games’ Chronos. Other implementations are the open-source XR T I, and CAE Canada is experimenting with a 1516 partial RTI implementation.
Magnetar Games (http://www.magnetargames.com/, formerly Ibis Research, http://www.ibis-research.com/) has been developing FederationX since 1997. Magnetar stands for Metaprogrammable AGent NETwork ARchitecture (http://www.magnetar.org/). The aim is the creation of a research-oriented open-software Collaborative Metaprogramming Framework (CMF) supporting small teams of developers and product designers who endeavour to create rich-media collaborative real-time entertainment and educational software. A core concept of FederationX is that of systems which execute domain-specific metaprograms expressed in eXtensible Markup Language (XML). The metaprogrammable features of systems are built up from FederationX engines. The networking engine is Chronos, an IEEE 1516 implementation using Microsoft’s DirectPlay communications layer. The result is a high performance and scalable service supporting large scale peer-to-peer and massive client-server applications. A similar effort is Cybemet Systems Corp.’s OpenSkies RTI, although it is based on the older HLA 1.3 specification.

The eXtensible Run-Time Infrastructure (XRTI; http://www.npsnet.org/~npsnet/xrti/) is an open-source, freely redistributable, Java-based partial implementation of IEEE 1516.1. The aim of the XRTI project is not to compete with commercially available run-time infrastructures, but rather to provide a foundation for research into improvements and extensions to the HLA. It does address a number of themes currently being actively pursued by a number of SISO HLA Product Development Groups (PDGs), such as dynamic FOM modification during federation execution and RTI interoperability.

Because of this lack of real choice, no comparisons could be made, and the decision was made to leave such efforts for a later endeavour. A few comments are nevertheless in order.

The pRTI 1516 package installs and runs without any trouble, although it must be said that no extensive testing was conducted. The Javadoc is essentially nonexistent, which is unfortunate. This is a result of the IEEE 1516.1 specification, whose Annex B (the Java API) is normative.

The supplied documentation is meagre; the User’s Guide mentions one way to obtain an RTIambassador interface for C++ (through the undocumented hla.rti1516.dlc.RTIambassadorFactory class’s createRTIambassador(String[]) method). Proximity to the mysterious hla.rti1516.dlc.RtiFactory and hla.rti1516.dlc.RtiFactoryFactory classes does make one hesitate to use it, however. Later on, the same User’s Guide mentions the Java preferred approach, the se.prti1516.RTI class’s constructor. This is also where one learns of the se.prti1516.FederateAmbassadorImpl class.

Inspection of the prti1516.jar Java archive file reveals the hla.rti1516.utility classes, which seem to be an OMT data type implementation. Alas, without documentation, one hesitates to use them.
We ended up modifying the API files to include extensive Javadoc, mainly for ease of development and reference (Annex B). The Pitch-specific API is, understandably, proprietary and undocumented. There are some classes with tantalizingly significant names, such as hla.time1516.NormalizedInterval, hla.time1516.NormalizedTime, and se.prti1516.Encoder (which seems to implement a few of the OMT datatypes: HLAinteger32BE, HLAboolean, HLAunicodeString, HLAresignAction, HLADeal, HLALogicalTime, plus the apparently misnamed HLAlogicalTimeInterval).

**Bugs**

*IllegalArgument{Exception} not thrown*

According to IEEE 1516.1-2000, “12.4.2.15 Constrained set of attribute designator and value pairs: [...] AttributeHandleValueMap [...] extends java.util.Map [...] The values are instances of byte[]. IllegalArgumentException shall be thrown for violations. [...] The implementation shall not accept null mappings.”

However, `someAttributeHandleValueMap.put(someAttributeHandle, null)` fails to throw the expected `IllegalArgumentException`.

`ParameterHandleValueMap` (IEEE 1516.1-2000 12.4.2.16 Constrained set of interaction parameter designator and value pairs) suffers from the same bug: it fails to throw `IllegalArgumentException` when putting a null value.

**Large Dimensions not Handled Correctly**

Pitch pRffi1516 (2.3rl build 101) is unable to read FDDs that define large Dimensions. Specifically, as soon as a dimension’s upper bound reaches or exceeds $2^{31} = 2,147,483,648$, the RTIexec throws an "ErrorReadingFDD: Invalid dimension upper bound" exception. Clearly this is because it internally attempts to convert the upper bound to an int instead of a long (which is what the API allows). Note that Visual OMT 1516 has no problem dealing with large Dimensions.

**Reserved Names Not Rejected**

Although IEEE 1516.1-2000 simply states “6.2 Reserve Object Instance Name: [...] The value of the supplied name argument shall not begin with “HLA””, the intent seems to be as stated in IEEE 1516.2-2000: “3.3.1 Conventions - Names: [...] e) Names beginning with the string “hla”, or any string that would match ((‘H’|’h’) (’L’|’l’)) (’a’|’A’)), are reserved and shall not be included in user-defined names. [...] e) A name consisting of the string “ha”, or any string that would match ((‘H’|’h’) (’a’|’A’)), is reserved [...] and shall not be included as a user-defined name.” 1516.2-2000 goes on to list the names for which the rules are applicable, omitting object instances presumably because of the document’s focus on the OMT.
Pitch pRTI1516 adheres strictly to the 1516.1 definition, and throws an exception only for user-defined object instance names beginning with "HLA". The fact that it allows names beginning with "hla" or "Hla", etc., is arguably not a bug per se but rather a potential problem with the interpretation of IEEE 1516.

**Orphans Not Deleted**

IEEE 1516.1-2000 states: “6.1 Object Management - Overview: [...] Orphaned object instances are unknown by all joined federates and cannot be discovered by any means. [...] Orphaned object instances shall be unreachable by joined federates and should be dealt with appropriately by the RTI.”

Unfortunately, the phrase "should be dealt with" is never defined. Arguably, deletion is the appropriate policy: the orphans are undiscoverable, and attempting to register new instances of the same names will fail whether the orphans have been deleted or not. So they might as well be deleted, one may reason. The fact that Pitch pRTI1516 (2.3r1 build 101) fails to "deal appropriately with" (i.e. delete) orphaned object instances is again not a bug per se, but rather an IEEE 1516 interpretation problem.

**Integer64Time Missing**

The 1516.1 specification states “12.4.2.23: [the] implementers of the Java API shall provide [...] an implementation of the interface LogicalTime called Integer64Time, [...] an implementation of the interface LogicalTimeFactory called Integer64TimeFactory, [...] an implementation of the interface LogicalTimeInterval called Integer64TimeInterval, and [...] an implementation of the interface LogicalTimeIntervalFactory called Integer64TimeIntervalFactory". No such implementations could be found in the Pitch packages.

Instead, we find se.prti1516.LogicalTimeDouble, se.prti1516.LogicalTimeFactoryDouble, se.prti1516.LogicalTimeIntervalDouble, and se.prti1516.LogicalTimeIntervalFactoryDouble, which offer some additional (undocumented) methods. The end functionality is essentially the same, but one nevertheless wonders what else the certification process overlooked.

**Federate Service Thread Synchronization Bug**

Assume a Java class with subclasses handling the FederateAmbassador calls. The HLAautoProvide switch is Enabled. In what follows, the code has been simplified to the utmost (try/catch blocks aren’t shown, etc.).
In the main thread, at some point we have:

```java
synchronized(_me) {
  _rtiAmbassador.assoclateRegionsForUpdates(_handle, _newDistributionRegions);
  [ some calls that prepare an AttributeHandleValueMap object (ahvm) ]
  _rtiAmbassador.updateAttributeValues(_handle, _ahvm, null);
}
```

In the `FederateAmbassador.provideAttributeValueUpdate` handler, we have:

```java
[ validation code that checks the arguments passed in ]
synchronized(_me) {
  [ some calls that prepare an AttributeHandleValueMap object (_ahvm) ]
  new Thread() { public void run() {
    _rtiAmbassador.updateAttributeValues(_handle, _ahvm, null);
  } }.start();
}
```

The key factor is that both snippets of code synchronize on the same object (\_me).

Now a race condition occurs. Sometimes, the main thread enters and leaves the synchronized block before the Federate Service thread reaches its synchronized block. All is well, execution flows smoothly.

However, at other times the main thread is still within the synchronized block when the Federate Service thread starts up (because the `associateRegionsForUpdates` call has caused the object instance represented by \_handle to come into scope for some other federate, and the RTI then fetches the attribute updates in preparation for distribution—remember that AutoProvide is turned on). The Federate Service thread will appropriately stop just before the synchronized block, because that monitor is owned by the main thread. What is buggy, however, is that the main thread’s `updateAttributeValues` call then blocks for no discernible reason. The entire execution freezes as a result. By rights, the main thread’s `updateAttributeValues` call, even though it'll put calls in the Federate Service thread’s queue for later processing, should return normally so the main thread can then exit the synchronized block.

Put succinctly, if the Federate Service thread is in any kind of synchronization-waiting state (a semaphore, say), then a federate-triggering RTI ambassador call by any thread will freeze, even if the latter RTI ambassador-caller does not own any monitors.

It’s as if the RTI ambassador call, in the local RTI component, needed to wait after the `FederateAmbassador`, running in the Federate Service thread, to acknowledge the events being added to its queue, or something like that. Without knowing how the RTI is implemented, this surmise may be completely wrong; what matters is that the bug is real and potentially disastrous in its consequences.
Interestingly, even the pRTI Explorer seems confused about the state of the federation, as it reports the frozen federate as having no discovered instances of anything (even though there are some). This may conceivably be just a side-effect of the federate’s frozen state on the Explorer’s display methods.

Visual OMT 1516

Although there were a number of commercial offerings when the time came to edit OMT files for HLA 1.3, there is a dearth on the IEEE 1516 side. Several of the companies that existed at this project’s outset have apparently gone out of business, or have failed to make the 1516 move.

Pitch offers its own tool, Visual OMT 1516, which has the excellent feature of being able not only to import 1.3 OMT files, but also to export 1516 object models to the 1.3 format. Automatic conversion is supported to a reasonable extent; since DDM is very different in 1.3 compared to 1516, human intervention is unavoidably required.

Visual OMT 1516 supported our federation object model design process without any serious trouble. Because there were really no other tools to compare it with, and because we made only light use of object model editing, no recommendations are made on this subject.

Bugs

The Visual OMT 1516 (version 1.04) interface suffers from one apparent bug, which may be in part a matter of design. When one wants to attach a note (a table of which is part of the OMT) to some item in the object model, one pops the contextual menu and navigates to the Notes sub-menu. It displays the Notes table, which can be edited in-place (a very nice feature). The Enter key shifts the focus to the next note in the list, the delete key deletes the current note, and so on.

To (un)associate one (or several) note(s) to the item, one (un)checks the tick box(es) appearing beside the chosen note(s). This can be done with the mouse or the space bar. But the changes “take” only if one exits the menu in a very specific, counter-intuitive way: by (right or left) clicking one of the application menu titles. This is very puzzling behaviour. It would have been much simpler to have the ticks “take” without the need for any further action. The Visual OMT 1516 User’s Guide is unhelpful on this subject, as it declines to detail how notes are (un)associated to object model items. It laconically states “To add a note, right-click in the table-cell or on the label and select Notes on the menu.”
3. **JACK™ Intelligent Agents**

JACK™ Intelligent Agents is an agent-oriented development environment built on the JACK Agent Language (JAL), which is an extension of the Java programming language. It is the property of Agent Oriented Software (AOS) Group, based in Australia, the United Kingdom and the United States.

In the same way that object-oriented programming introduces a number of key concepts that influence the entire logical and physical structure of the resulting software system, so too does agent-oriented programming. In agent-oriented programming, a system is modelled in terms of agents. These agents are autonomous, reasoning entities capable of making pro-active decisions while reacting to events in a real-time environment.

Agent-oriented programming ([33] Shoham 1993, [34] Wooldridge and Jennings 1995) is an advanced software modelling paradigm that arose from research in distributed artificial intelligence. It addresses the need for software systems to exhibit rational, human-like behaviour in their respective problem domains. This makes it ideally suited to the task of lightening the operator load required for a constructive simulation. Agents of varying degrees of sophistication could be responsible for simulating various expected entity behaviours, ranging from a civilian bystander’s reaction to a sudden burst of gun fire, to the cunning expected of an adversary general. A similar pursuit of verisimilitude has been observed in the software gaming industry, where the quality of the behaviour of simulated entities can make or break the commercial success of the title. Adventure role-playing titles strive to have complex non-player characters that can actually keep parts of the plot in motion independently of the player’s choice of actions, or serve as reasonably believable adventuring associates (hirelings, comrades, allies). Empire-building simulations strive to provide believable (and tuneable) adversaries that can give the player a good run for his money without resorting to obvious cheats such as drastically reduced time and money costs. The list goes on.

The term *agent* is widely used to describe a range of software components ([35] Franklin and Graesser 1996), varying in capability from procedural wizards (found in popular desktop applications), to information agents (used to automate information search and retrieval), and to intelligent agents capable of reasoning in a well-defined way. The agents used in JACK are *intelligent agents*. They model reasoning behaviour according to the theoretical Belief Desire Intention (BDI) model of artificial intelligence ([36] Bratman 1999).
Following the BDI model, JACK intelligent agents are autonomous software components that have explicit goals to achieve or events to handle (desires). To describe how they should go about achieving these goals, these agents are programmed with a set of plans. Each plan describes how to achieve a goal under varying circumstances. Set to work, the agent pursues its given goals (desires), adopting the appropriate plans (intentions) according to its current set of data (beliefs) about the state of the world. This combination of desires and beliefs initiating context-sensitive intended behaviour is part of what characterises a BDI agent.

An agent can be thought of as a person with access to a procedures manual. The procedures manual (set of plans) describes the steps that the agent should take when a certain event arises or when it wants to achieve a certain outcome. At first glance, this may seem like ordinary expert system behaviour—with all the limitations that this implies. However, the crucial difference in agent-oriented systems is that the agent exhibits goal-directed focus: it focuses on the objective and not the method chosen to achieve it. In addition, the agent is continuously aware of context, re-evaluating the validity or relative importance of various goals being pursued simultaneously in the light of new events or goals.

**Overview of JACK**

**Events**

Quite naturally, agent-oriented programming is an extension of event-driven programming. When an event occurs, it is either posted to the agent owning the entity wherein it arises, or sent to a specific different agent. The agent handling the event may react by taking action. There are a number of event types in JACK, each with different uses. These different event types help model:

External stimuli, such as messages from other agents, percepts that an agent receives from its environment, or belief state changes. These events make the agent reactive.

Internal stimuli, events that an agent sends to itself. These can represent sub-tasks that the agent uses to break down complex goals or meta-reasoning events that the agent uses to decide which plans to implement. They are integral to the ongoing execution of an agent and the reasoning that it undertakes, and make the agent proactive.

Motivations that the agent may have, such as goals that the agent is committed to achieving.
Events are the origin of all activity within an agent-oriented system. In the absence of events, an agent sits idle. Whenever an event occurs, an agent initiates a task to handle it. This task can be thought of as a thread of activity within the agent. The task causes the agent to choose between the plans it has available, executing a selected plan or plan set (depending on the event processing model chosen) until it succeeds or fails. Multiple tasks can be active at once, the agent using a specified task management policy to allocate its execution resources between them.

Belief Sets

In addition to ordinary Java data members and other data structures, the JACK Agent Language provides belief set classes that facilitate the maintenance of an agent's beliefs about the world, represented in a first order, tuple-based relational model. Each belief set relation consists of a set of fields, each of which can be any Java primitive type (boolean, byte, char, short, int, long, float and double), a String or an Object. All, none or some of the fields may be declared to be keys. Keys serve to indicate the object of a tuple; that is to say, what the belief is about. There will be at most one tuple in the belief set for any given combination of distinct keys (if none of the fields are keys, then all of the belief set’s tuples are about the same unique object).

The belief sets automatically maintain their logical consistency: when an agent adds a belief that contradicts a previously held belief, the belief set detects this and automatically removes the old belief. They are also able to post events automatically when certain changes occur. Finally, cursor statements are supported (more on these shortly).

Belief Sets descend from two root classes, supporting Open World and Closed World semantics respectively. In a Closed World belief set relation, all statements (possible tuples) are either true or false, so that the unknown state cannot occur. In an Open World belief set relation, by contrast, all possible statements (beliefs) are initially in the unknown state and the agent can therefore establish a fact either by assertion (by affirming it) or by elimination (by denying all other possibilities). Another way of looking at it is that, in a Closed World, all possible queries have a definite yes or no answer, whereas in an Open World some if not most queries will probably be unanswerable.

Closed World relations theoretically represent every possible tuple; in practice, only those tuples that the agent believes to be true are stored, and any tuple that is not stored is assumed to be false. Open World relations store all true and false tuples, and assume any tuple that is not stored to be unknown.
The JACK Agent Language makes extensive use of logical variables, which, although typed, are initially unbound (have no definite value). They can be thought of as the unknowns in the initial statement of a problem. The problem is solved once all unknowns have been bound to specific values, a process known as unification. Unification can be invoked explicitly, or it can occur indirectly. Belief sets are normally queried through cursors, which are a means of scanning through the successive possible bindings of a set of logical variables. A cursor, when first queried, provides bindings for its implicated logical variables. If these bindings are determined to be unsuitable to the agent's purposes, the cursor, when next queried, rolls the bindings back and provides a new set of bindings (assuming one exists). In this fashion, an agent can entertain successive hypotheses until a satisfactory one is achieved.

Finally, for ease of interoperability the JACK Agent Language defines views, which can be thought of as highly abstracted belief sets. Much of the methods and behaviours automatically generated by JACK for a normal belief set become optional and can be implemented with great freedom of choice by the programmer. Views allow any external entity to be encapsulated to serve as a belief set. They can be used in a variety of ways, such as querying across multiple belief sets without duplicating the beliefs, or turning the keyboard input stream into a source of events, to name but two.

**Plans**

Plans are the heart of JACK agent programming. Plans describe sequences of actions that an agent can take when events occur. They can be thought of as the pages from a procedures manual; each describes, in explicit detail, exactly what an agent should do when a given event occurs.

Each plan is declared as handling a single event. When an instance of a given event arises, the agent first constructs the set of applicable plans. This set is initially constituted of those plans which the agent uses and which declare they handle this event. The agent further discriminates between these plans by checking their relevance. The relevant() method of each plan is a static (class-level) method which can examine the attributes of the event to decide whether it is relevant or not. A final criterion is applicability. The context() method of each plan is dynamic (instance-level) and serves to take into account the current circumstances (the agent's current beliefs and any other agent-accessible data). This is where, typically, the plan's logical members' values are bound. For every possible set of bindings, a separate applicable instance of the plan is generated.

Once the agent has found all applicable instances of each relevant plan, it selects one of these to execute. The selection method used is controlled through various programming means, which may include the posting of a PlanChoice event. JACK agents are fully capable of meta-level reasoning, that is to say, plans about plans.
The body of a plan is an instance of what the JACK Agent Language calls a reasoning method. Reasoning methods extend the normal Java execution paradigm, in that each statement is treated as a logical expression that can either succeed or fail. Another way to look at it is to consider the reasoning method's statements as being connected by Java conditional-AND operators (each reasoning method statement is atomic, however). A reasoning method fails as soon as it reaches a failed statement: this fails the plan instance, and the agent then fails the event handling or tries again with a different plan instance depending on how it has been programmed.

Besides the mandatory body() reasoning method, a plan may declare other reasoning methods as members. This is a convenience that facilitates the breakdown of complex plans for maintenance, re-usability and verification purposes. In particular, there are the pass() and fail() optional reasoning methods, which are automatically invoked by JACK once the plan succeeds or fails, respectively.

Another JACK Agent Language feature that appears within plans are the reasoning method statements or @-statements, language extension tokens which serve to wait or watch for certain conditions, to reply to events sent by other agents, or to post or send events under a variety of conditional modes (analogous to the usual control flow paradigms: if, switch, for, while, repeat until).

Capabilities

Capabilities are a programming convenience similar to Java packages. They are a means of structuring reasoning elements of agents into clusters that implement selected reasoning capabilities. This simplifies agent system design, allows code re-use, and encapsulation of agent functionality.

Capabilities are built in a similar fashion to simple agents: constructing them is merely a matter of declaring the JACK Agent Language elements required. Events, belief sets, views, plans, Java code and other capabilities can all be combined to make a capability.

It should be fairly obvious that a generic, re-usable HLA “module” for JACK is best implemented as a capability.

Multi-Threading

When we described events earlier, we mentioned that each event triggers a task, which corresponds to a thread of activity within the agent. Java is inherently multi-threaded, and switching between execution threads is normally the responsibility of the programmer. The Java execution engine does not guarantee safe points at which it will switch execution threads, so it is up to the Java programmer to implement object locking monitors and any other concurrency controls which may be required.
JACK is also multi-threaded but the context switching is managed in large part by the JACK kernel. Reasoning methods and task executions are Finite State Machines, meaning each statement is executed in a series of atomic steps, between which the agent can switch to other execution threads safely. A compound logical statement, for example, is executed in the usual short-circuit way, and thus each individual Boolean evaluation is a separate step. This simplifies considerably the task of making a plan thread-safe.

Each agent uses a TaskManager object to decide how to allocate its execution time between tasks (threads). The two most basic TaskManager strategies are depth-first (the agent pursues one task until it completes or blocks, then switches) and breadth-first (the agent executes a certain number of steps in any one task before switching to another, in round-robin fashion). More elaborate TaskManagers are possible, for example, prioritizing tasks.

**JACK-HLA interface**

**HLA services, callbacks and JACK events**

It was immediately realised that the HLA callbacks, which can occur at any time and are the only form of message issuing from the RTI besides the few services that actually return an immediate response value, should map to JACK events. The services, on the other hand, can be invoked from anywhere within the JACK code simply by providing the RTI ambassador reference.

The HLA-expected behaviour of the Federate Ambassador is, in any case, minimalist: in order to allow callbacks to occur at any time and at as high a rate as necessary, the callback handler should conclude as quickly as possible. This is best achieved, as was discussed earlier, by storing the callback’s arguments and either queuing the event internally (for processing by another thread) or spawning a separate worker thread. This is precisely what is achieved by posting a JACK event, so the match is natural.

However, the federate does have the opportunity (indeed, the responsibility) of throwing exceptions from within the callback’s thread in order to advise the RTI that something is amiss. This behaviour is exceptional but may nevertheless be necessary. As a general rule, these exceptions are thrown if argument validation fails, each exception corresponding to a different validation failure mode. Because of the thread constraint, a generic HLA-JACK adaptor module must treat this validation activity as an exterior handler. We’ll see later how this was achieved.
In order to post JACK events, our code needs access to an event posting method reference (an event factory) and to an agent reference (in order to invoke the postEvent or send methods). We first thought of putting the HLA FederateAmbassador implementing class in the project’s Other Files section, but this does not work for the simple reason that an agent’s event posting methods, once pre-compiled to Java, are private to the agent. There are two ways around this obstacle. The first is to add to the agent a series of public methods that each return a different event factory. It is crude but effective, and has the main drawback that the programmer must go through the drudgery of creating each event factory access method manually. In addition, it forces the agent to declare that it posts or sends the events (even though it does not) because the handles declaration only mentions the event type, omitting the event factory instance.

The second way is to move the HLA FederateAmbassador implementing class into the agent class as an inner class. This immediately gives it access to all of the enclosing agent’s members.

Since the agent must manage several HLACHat instances in any case, putting their references in a belief set makes more sense. It becomes relatively easy to recover the HLACHat reference as part of each plan’s context() method.

**HLA1516 JACK Capability**

An RTI implementation typically supplies a “null” Federate Ambassador implementation which the user is expected to subclass, overriding only those callbacks which are of interest to him. This null implementation simply implements each of the 56 callbacks but does nothing (the method bodies are empty). This makes the user’s subclass code a lot more compact and easier to read. To follow this pattern, our HLA 1516 capability should post all possible events but not require the user to supply “null plans” for those events which are not of interest. We achieved this by having the capability handle each of the events as well as post them, but use in each case a null plan that categorically declares itself as irrelevant (the relevant() static method returns false in each case). This satisfies the JACK runtime initialization, which won’t complain that the agent using the capability has unhandled events, and neatly avoids the undesirable side effect of having significant events “ambushed” by the capability. This would occur because JACK capabilities are searched for applicable plans before any of the agent’s (or enclosing capability’s) own plans.
Although the HLA Federate Ambassador interface consists of 56 different callbacks, there are only 43 differently named ones. Four callbacks (initiateFederateSave, reflectAttributeValues, receiveInteraction, and removeObjectInstance) are overloaded to varying degrees. These sets of overloaded callbacks can be safely combined into four events by substituting null arguments as needed, since the HLA specification guarantees that no argument will ever be null, except for the userSuppliedTag byte arrays which appear in eight callbacks (announceSynchronizationPoint, reflectAttributeValues, receiveInteraction, removeObjectInstance, provideAttributeValueUpdate, requestAttributeOwnershipAssumption, attributeOwnershipAcquisitionNotification, and requestAttributeOwnershipRelease). For example, initiateFederateSave has two forms, one with a single argument (a String), the other with two (a String and a LogicalTime); the former is rolled into the latter by passing a null LogicalTime.

Several sets of events have the same payloads (field sets) and could conceivably be further merged provided some discriminant fields were added. For example, the federationRestoreBegun, federationRestored and federationSaved events have no fields beside the identifier. There are no clear benefits to be gained from such mergers, however, except possibly for those events which pair off in Boolean fashion: attributes In/OutOf Scope, turnUpdates Off/On ForObjectInstance, objectInstanceNameReservation Failed/Succeeded, requestFederationRestore Failed/Succeeded, start/stop RegistrationForObjectClass, and turnInteractions Off/On. This is really a matter of programming style more than anything else.

To allow an agent using the capability to potentially manage multiple federates, the capability’s inner class (HLA federate) has a single constructor that expects an identifier String argument. This argument is prefixed into each event posted out of the capability, thus allowing the agent to tell which of the Federate Ambassadors generated it.

Next, to allow the HLAfederate’s callbacks to throw exceptions as needed, we declared 43 interfaces (one for each differently named callback) which consist of a single method (validate) accepting the callback’s arguments and throwing its exceptions (the method is overloaded just like the callback itself). The HLAfederate stores privately a reference to each of these interfaces, null by default, and exposes get and set methods to allow the references to be manipulated. When a callback is invoked by the RII, if the corresponding validation interface reference is non-null, the arguments are passed to it before moving on to the event posting method.
Thus, the capHLA1516 capability consists of:

- An inner class (HLA federate), which includes a placeholder member for the federate’s RTI ambassador instance;
- Forty-three events (MessageEvent) encapsulating the 56 HLA 1516 Federate Ambassador callbacks, each both posted (#posts external event) and handled (#handles external event) externally; and
- Forty-three used plans (#uses plan), each of which declares itself as irrelevant (the static relevant method returns false) and has consequently no reasoning method; and
- A belief set prototype (blfHLA), an instance of which you must supply to the capability.

The blfHLA belief set is used by the capability to manage the HLA federate instances. The enclosing agent (or capability) queries the belief set in order to recover the HLA federate instance, using the identifier prefixed to the event. This is typically done by its plan’s context() method.

Figure 19 illustrates the process. The RTI’s callbacks are handled by the HLA federate instance, which tacks its identifier to the event parameters set when invoking the capability’s event factory. The agent may supply validator methods to the HLA federate instance if it wishes exceptions thrown at the RTI. The HLA federate instance has a placeholder for the corresponding RTI ambassador instance, so the agent’s plans may easily invoke RTI services.
Figure 19. How an agent uses the HLA capability
The agent is free to specify Validator interfaces or not; it is also free to ignore irrelevant events.

External dependencies were kept to a minimum. The capability imports the generic hla.rti1516 classes, and two of our own packages. The first, ca.gc.drdc_ddc.hla.rti1516.omt, implements the HLA opaqueData datatype (as specified by IEEE 1516.2 at 4.12.6) which is used to wrap the byte[] arguments occurring with certain callbacks. This was necessary to circumvent JACK's inability (in its 4.1wj version) to cope with byte[] event fields. Another possible solution would have been a java.nio.ByteBuffer implementation (the Java 2 Platform class is abstract), but since an HLA-capable JACK agent is likely to use other HLA OMT classes, it seemed more logical to supply those anyway.

The second, ca.gc.drdc_ddc.hla.rti1516.FedAmb, specifies the event validation interfaces.
In a concrete application, an `RTIambassador` implementation is required. This is part of what each RTI vendor supplies, and its constructor can take a variety of guises. Luckily, we don’t need to worry about this. The `HLAfederate` class supplies a placeholder `RTIambassador` member (`rtiAmbassador`), which is expected to be filled in by the agent using whichever means it finds convenient.

Finally, we expect the using agent to extend the `HLAfederate` class as a convenient way of keeping all federate-related fields and methods together. In order for the belief set’s `getInstance` query to return the correct `Object` instance, the only requirement on the extending class is that its constructor invoke the superclass’s constructor through the `super` keyword, something which Java enforces anyway.

**HLA13 JACK Capability**

Retrofitting the HLA1516 capability for use within an HLA 1.3 context turned out to be very easy. The HLA1516 capability was used as a template, the processing model remaining the same. The classes and interfaces were repackaged to avoid conflicts with 1516, and the methods and interfaces appropriately renamed.

**JACK Integrated Development Environment (IDE)**

The JACK Development Environment (JDE) is decent but suffers from a number of really annoying deficiencies. The built-in text editor recognises only a small set of key strokes and is seriously deficient when compared with such simple editors as the Windows built-in Notepad. Understandably, AOS does not want to spend much effort on it and prefers investing in the useful aspects of the engine. JACK does offer the capability to use an “external” text editor, which is something we never got around to trying out. It may solve the problem nicely. The JDE also seems to suffer from some form of memory leakage: after opening, editing, closing and saving various parts a fair number of times, we get a message that the JDE is running out of memory. At that point, one saves the project and restarts the JDE: a mere annoyance.

It may be that version 5, delivered as this report was nearing completion, solves some of these small complaints.

The JDE is not as robust as it should be, and even an unimaginatively devious user can create all sorts of problems for himself. A few examples:

- If you drag and drop a Named Data to an Agent’s Belief Data container twice, you get two identically-named references, which of course won’t compile properly.

- If you create several Named Data instances of various types, they all get the same default name "data", which leads to confusion and/or compilation errors later on. A simple default name convention could be "data_<BeliefSet Type>_1" (e.g. "BeliefSetDelusions" would generate "data_BeliefSetDelusions_1").
• If you remove an Event Type from the project window, any separate windows previously created for that Event remain opened instead of closing (and will still accept drag-and-drop, editing as JACK file, etc). One can even create a new Event Type using the still-open Event’s name, leading to potentially catastrophic confusion within the project’s file structure. The same can occur with Events’ Fields, Agents, Plans, Capabilities, Events’ Posting Methods, BeliefSet Queries, etc.

• Using drag-and-drop, it is quite possible to mark a Capability as having itself as a Sub-Capability. Is this reasonable or will it create an infinite loop of some sort? If the latter, care will need to be taken to make sure the Capability hierarchy remains an acyclic directed graph (thus if B is a sub-capability of A, then it should not be possible to declare A as a sub-capability of B, and so on regardless of the number of intermediate steps).

If one treats the JDE as the fragile thing it is, all is well. Productivity may not be what it should be (compared to Java Integrated Development Environments (IDEs) such as NetBeans or Eclipse), but it is tolerable.

**JACK Bugs**

Defining an array field for a MessageEvent or BDINessageEvent (but not for the other event classes) causes a compilation problem. This occurs both with primitive data types (byte, int, etc.) and with classes in general, although the error reported is different. This is apparently (according to AOS) a limitation of using the JACOB transportation of message events by default. The solution is either to force the use of Java serialization as the event transport mechanism (by adding #set transport java; to the event class definition), or to wrap the array in a utility class (HLA opaqueData in our case).

**JACK Pre-Processor Bugs**

The JACK-HLA interfacing effort revealed a number of unexpected JACK bugs, which arose from its pre-processor. When a JACK application is compiled, the pre-processor reads the plans, events, agents and other JACK-specific constructs in order to process the JACK Agent Language syntax extensions. The remaining code is expected to be standard Java but is nevertheless validated as part of the pre-processor’s parsing. This step is required because JACK must parcel out the Java statements into the sometimes rather involved Java structures that make the extensions understandable to the Java virtual machine. For example, each of the statements in a plan’s reasoning method has to be put in a separate branch of a large switch statement at the heart of a finite-state-machine loop understandable by the proprietary JACK executable classes.

It turns out that some unusual Java syntax is involved in invoking or extending inner classes, and that the JACK pre-processor did not understand those correctly. This resulted in spurious errors being reported during the pre-processing phase.

Specifically, this line would not compile:
String s = AgentClass.InnerClass.aConstantString;

Here we use a qualified class identifier to access an inner class’s static member. A simple workaround is to have the outer class pass the inner class’s member out through a static member of its own. This line wouldn’t compile either:

AgentClass.InnerClass inner = agent.new InnerClass();

Here we invoke an inner class’s constructor by qualifying it with the enclosing instance, as is proper. A simple workaround is to have the outer class pass the inner class’s constructors out through a factory method of its own.

A third instance of the problem arises when one extends an inner class from outside itself. In such a case, the Java language dictates that the extending class’s constructor use the ancestor’s enclosing instance as its first argument. By itself, that works fine, but if you need to invoke the ancestor’s constructor through the super keyword, the latter will also need to be qualified by the ancestor’s enclosing instance, like so:

```java
public class OuterClass {
    ...
    public class InnerClass {
        ...
    }
}
public class InnerClass_Descendant
    extends InnerClass {
    public InnerClass_Descendant(OuterClass containingInstance) {
        containingInstance.super();
    }
    ...
}
```

In this case the qualified super stalls JACK’s pre-processor. There is no simple workaround in this case, except to comment the offending line out, compile the project, then manually uncomment the super call from the generated agent’s .java file and recompile it. This is awkward, but it gets the job done.

Although we have not tested this, AOS assures us these bugs are all fixed in the newest release of JACK.
4. Chat Application

We chose as our test case a chat application. The main virtue of such a simple demonstration is that it pre-existed in both the HLA and JACK cases, a natural consequence in both cases of their network-awareness.

One of the very first network chat applications was ARPANet's Planet chat system, in 1973. Internet Relay Chat (IRC) itself was invented by Jarkko Oikarinen in 1988 at the University of Oulu, Finland ([37] Hardy 1996, [38] Kantor 2003). It became popular after it was made famous during the Iraqi invasion of Kuwait in 1991, where it was used by people to get information about events in Kuwait to the outside world after all other forms of communication had been cut off.

A chat application can be a simple text-only interface, command-line-driven, which allows a user to (this list is by no means exhaustive, but it is sorted in rough order of feature necessity):

- Log into and out of the network;
- Send and receive text messages;
- Join or create chat groups (chat rooms);
- Send and receive private messages (messages addressed to a single other user);
- Find out which groups/rooms exist; and
- Find out which other users are logged on.
Figure 20 illustrates the functioning of a chat application. Each client instance logs in and out of the chat network, appearing in the Default Group (a variety of names are possible: the “Reception Hall”, the “Lobby”, the “Town Square”, etc.). Groups (chat rooms, channels, etc.) are created at will, and destroyed once empty. Clients can join a group or leave it (revert to the default group). Messages are sent to all clients within a group unless private messaging is used. Groups can be organized hierarchically, sub-groups being considered “in” their enclosing group, but this is not considered here.

**JACK Chat Demonstration**

We chose as our starting point the JACK chat demonstration. We strove to change as little of the application as possible, which meant the Client agent could not be changed. Because the JACK chat demonstration uses a Client-Server paradigm, this was easily achieved.
The JACK chat demonstration consists of two executables, a Server and a Client. Each instance of the Server agent manages a separate chat network, and each Client instance must specify (in its command line) to which Server it will connect. The Server is completely automated; the only user interventions consist of starting it and shutting it down.

![Image of JACK Chat Server window]

**Figure 21. The JACK Chat Server window**

Its role is analogous to the RTI's RTI Manager application.

The Client uses a command line interface with a two-state architecture. Initially, the Client is in the "Logged-Out" state. The user has a very limited choice of commands:

- The commands `quit` or `exit` will shut down the Client agent altogether;
- The command `login <username>` will attempt to log the Client onto the chat network;
- Any other command acts as a help command and lists the possible commands.

![Image of JACK Chat Client window at start-up]

**Figure 22. The JACK Chat Client window at start-up**
If the login fails, the Client remains in the “Logged-Out” state. Otherwise, the login request was successfully processed by the Server (that is to say, the requested username was available) and the Client is now in its “Logged-In” state, and joins the “General” chat group. New commands become available:

- The **logout** command disconnects the Client from the chat network and shuts down the Client application;
- The **join** <groupname> command joins the Client to the specified chat group, creating the group if necessary;
- The **leave** <groupname> command unjoins the Client from the specified chat group;
- The **who** [<groupname>] command lists the users (including the Client) currently in the specified chat group (if unspecified, the Client’s current chat group joined is assumed);
- The **msg** <text> command sends the message (the specified text) to the users in the Client’s current chat group;
- The **msgusr** <username> <text> command sends the message (the specified text) to the specified user;
- The **group** command lists the existing chat groups; and
- The **help** command lists the available commands.

Advisories appear when another user either joins or leaves the group the Client is in.
The application, as supplied, suffers from two design flaws: 1) nothing prevents a Client from joining multiple groups; and 2) nothing prevents a Client from leaving all groups, even the default one. This was clearly unintended since each Client remembers only the last group it joined and can only send to its “current” group (the last one joined). Joining multiple groups means a client will receive messages from all of the groups it has joined. A Client that joins more than one group and then leaves the last one no longer knows which group(s) it is in. It considers itself group-less and thus unable to send any group messages. Finally, a group-less Client can only be reached by private messages.

Clearly the intent was to restrict the Clients to belonging to one and only one group at all times, and to have them revert to the default group when leaving a specified group. It is possible to properly support multiple-group affiliation (by sending messages to all groups one belongs to, and warning the user when he ends up group-less), but that adds complexity which is not useful for our purpose: an HLA-JACK interoperability demonstration.

Figure 24 shows the key features of the JACK Chat demonstrator. Each Client instance must go through the same Server, identifying itself through a login request. All Client requests go through the Server. The latter maintains three belief sets: a mapping of usernames to Client instance references, a list of extant groups, and a mapping of Client usernames to group names (which keeps track of the group each Client is “in”). Using these belief sets, the Server can easily supply lists of groups and lists of co-located Clients, as well as manage the flow of messages between Clients.
In retrospect, the arbitrary decision to maintain the Client-Server architecture used by the original JACK demonstration may not have been the best approach. By keeping the Client unchanged, we are incurring a double networking overhead. Every time HLA invokes a callback in a JACK FederateAmbassador instance, this occurs within the Server’s process space. The Server then posts an event to itself and goes through the overhead of binding to the FederateAmbassador instance from within (one of) its task processing thread(s). The plan then eventually sends a message to the appropriate Client, going through JACK’s own communication layer. The process occurs in reverse when a Client sends a command out. Note that if the Server is using a single task thread (the default setting), it will handle only one client’s callback at a time.

We could have had each JACK Chat instance be its own server. This would do away with the JACK communication layer, get rid of the FederateAmbassador instance binding overhead (or at least reduce it to a minimum), and allow true multitasking (since each client would operate in its own process space).
**HLA Chat Demonstration**

Although a number of HLA chat demonstrations exist, we chose to develop one from scratch in order to match the operational concept of the JACK chat demonstration as closely as possible.

HLA is a very “democratic” architecture: all federates are equal in importance in the eyes of the RTI. Breaking this symmetry is possible but is a function of the federation’s semantics: during federation development, a federate may be singled out to act as a “server” for certain classes of objects, for example. For our Chat demonstration, we decided to use a peer-to-peer architecture, where all federates are “clients.” This means the JACK “Servers” must become proxies for their Clients, managing multiple FederateAmbassador instances on their behalf but having no existence as far as the federation is concerned.

**Figure 25.** The Java Chat demonstration

Besides events, the client is aware of User (Participant) and Group (ChatRoom) objects.

Our pure-Java Chat Client uses a graphical interface. Initially, the Client is in the “Logged-Out” state. The user can then either:

- Shut down the Client altogether; or
- Attempt to log in after specifying a `<username>`.
Once successfully logged in, the Client joins the "General" chat group. The extant chat rooms are listed, as well as the other users that are in the same chat room. The user can now:

- Shut down the Client altogether;
- Log out from the chat network and return to the "logged out" state;
- Join a group (chat room) by selecting it from the drop-down list;
- Create a group (chat room), specifying its name (the "New" button);
- Send a message to the users in the Client’s current chat group; or
- Send a private message to one of the other users in the Client’s current group (message addressees are designated using the “Send To” drop-down list).

Advisories appear when another user either joins or leaves the group the Client is in.

As can be seen, the semantics are essentially the same. Key differences are that the Client is now always in exactly one group at a time, and that private messages are only possible within the current group. These changes were integrated into the JACK chat at the same time that it was interfaced with the HLA chat. We confined the changes to the Server part of the JACK chat demonstration, which means the Client’s interface is now less suitable. The “leave” command still requires that the group being left be specified, which is pointless now since the client can only leave the current group. Such is the price of backward compatibility.

Figure 25 adapts Figure 24 to the Java context. The server is eliminated and the RTI acts as a connector between the Client instances, mediating User and Group objects (as well as Message interactions). The semantics of the federation now reside in the expected behaviour of the federates (the Clients), and are thus much less readily apparent from this type of diagram.
Figure 27. The Java and JACK Chat Client windows after both have joined

Figure 28. The Chat federation in action
**HLA Chat Federation Object Model**

*Dimensions*

Because we need to use DDM in two different ways (to channel messages by group and by user), we shall define two dimensions:

- **ChatRoomSlots** ([0..32 768[, excluded by default)
- **UserHandleSlots** ([0..2 147 483 647[, excluded by default)

Here we ran into a bug with Pitch pRTI 1516. The 1516.2 specification states that dimensions can be of any simple or enumerated datatype, although their upper bound is always specified as an integer. Since the integer basic datatypes (1516.2 4.12.3 Table 23) are 16-, 32- and 64-bit (two’s complement signed) integers, it should be perfectly legitimate to define a dimension that uses the `HLAinteger64BE` datatype. However, pRTI 1516 RTIexec throws an `ErrorReadingFDD “Invalid dimension upper bound”` exception when `createFederationExecution` is invoked with an FDD containing dimension upper bounds reaching or exceeding $2^{31} = 2 \times 147483648$. Apparently pRTI 1516 tries to store internally the upper bound as a Java `int` (32-bit two’s complement signed integer). Note also that the specification does not mention what should happen if you define a dimension with datatype `HLAinteger64BE` and then try to define a default range that covers the entire positive span: 64-bit signed integers can range up to $2^{63}-1$, which means an upper bound of $2^{63}$, a number which cannot be represented as an `HLAinteger64BE`. This occurs only because upper bounds are defined by their excluded value (dimensions are semi-open intervals, ranging from zero, inclusive, to the upper bound, exclusive).

*Switches*

The switches we enabled are:

- **AutoProvide**
- **AttributeScopeAdvisory**
- **InteractionClassRelevanceAdvisory**

The AutoProvide switch is simply convenient; it means our subscribing federates need not send a Request Attribute Value Update after discovering an object. The RTI will solicit the owner automatically with a Provide Attribute Value Update callback.
The Java Chat federate uses the Attribute Scope Advisories to add to/remove from its drop-down list of private message targets. The JACK Chat federate does not care about these advisories, since the Server resolves the list of potential chatters only when a Client requests it (using the who command). The Java federate could have used object discovery to accomplish the same thing (an attribute scope advisory always follows a discovery notification), but this approach involves considerably more overhead. Firstly, in order to receive a discovery notification every time an object comes into scope, the federate must “forget” about the discovered object (using the LocalDelete service) every time it goes out of scope. This required action cannot be accomplished within the scope advisory callback because of the ConcurrentAccess feature (q.v.): it must be threaded off. Secondly, this also entails the additional attribute value traffic concomitant with discovery.

The Interaction Class Relevance Advisories, finally, are used in an unexpected fashion to solve the federation destruction problem, as we’ll see later on.

**Datatypes**

We defined only two additional datatypes beyond the mandatory basic datatypes:

- ChatRoomRegistryEntry is a fixed record of two fields:
  
  ```
  name is an HLAunicodeString
  slot is an HLAinteger16BE
  ```

- ChatRoomRegistryEntries is a dynamic array of ChatRoomRegistryEntry

Attributes of our custom datatypes which are sent to the RTI or received from it are passed as byte arrays, and the RTI does no validation whatsoever over them. It is entirely up to the federates to ensure that they use the same classes to encode and decode the custom datatype attributes. This can be seen as providing flexibility, but it is also fraught with potential disaster. Subtle differences in the implementations of a custom datatype between two distinct federates may lead to apparently correct behaviour until very specific circumstances arise where the byte representations diverge, unless cross-certification is exhaustive.

**Interaction Classes**

We defined a single interaction class:

- Communication has two attributes:
  
  ```
  message is an HLAunicodeString
  sender is an HLAunicodeString
  ```
This represents a chat message being sent across the network. The message is the “payload”, that is to say, the text being sent by the user. The sender serves simply to identify the sending federate by its user name; this is necessary because users expect the interface to attribute messages as they occur, whereas HLA does not explicitly provide an interaction’s provenance. In order to make DDM possible, the interaction is associated with both dimensions.

Object Classes

We defined three object classes:

- **ChatRoomRegistry** has one attribute:
  
  `list` is a `ChatRoomRegistryEntries`

- **ChatRoom** has two attributes:
  
  `slot` is an `HLAinteger16BE`

  `name` is an `HLAunicodeString`

- **Participant** has three attributes:
  
  `logged_in` is an `HLAboolean`

  `user_handle` is an `HLAinteger32BE`

  `chat_room_slot` is an `HLAinteger16BE`

The attributes of both the ChatRoom and the Participant are associated with the ChatRoomSlots dimension. The ChatRoomRegistry is treated as a “global” object and therefore does not have its attribute associated with a dimension.

General Principles of Operation

Although HLA considers ownership at the instance attribute level (that is, an object instance’s attributes could each be owned by a different federate), in our simulation we will treat ownership at the object instance level. Ownership will be transferred using the same single attribute set for each instance of any given object class: the object’s FOM-specified attributes plus the `HLAprivilegeToDeleteObject` attribute. This simplifies what is meant by “ownership” and suits our purposes nicely.

The first important design decision was to forgo time management entirely. None of the federates are time-regulating or time-constrained, and all HLA messages will be receive-ordered. This simplifies considerably some aspects of the federation, but it also means that any required synchronization will have to be accomplished by non-time-dependent means.
In order to allow the users complete freedom in specifying their user names and chat room names while avoiding collisions with the RTI-reserved names (anything beginning with “HLA”, in any mixture of case), we chose to systematically prefix object names. These prefixes are for the federation’s consumption, and are not seen by the users. Participant objects’ names are prefixed by “p”, ChatRoom names are prefixed with “e”, and chat-system reserved names are prefixed with “_”. This approach also prevents collisions between user-specified ChatRoom names and system-reserved ChatRoom names.

Role of the Participant Objects

The Participant object can be described as a “pass”, required for admittance to the chat federation. When a user logs on, he does so under a specific user name, which must be unique. The user name is borne, appropriately prefixed, by at most one Participant object instance. If the federate manages to obtain ownership of that Participant object instance (through creation or ownership transfer), the log-on is successful. Accordingly, the federate shall refuse to relinquish ownership as long as it remains logged on.

To ensure the uniqueness of user names, it is much simpler to use HLA’s Name Reservation service rather than implement some name control scheme of our own. However, HLA’s Name Reservation service is persistent, in the sense that once a name has been reserved, it cannot be used again, even if the object instance is deleted. This means our Participant objects must also be persistent, being deleted only when the federation itself is destroyed. Hence the logged_in attribute, which serves to mark those Participant instances that are “in use.”

Another feature of HLA is that objects must not become “orphans.” That is to say, they must have at least one owner (i.e. at least one attribute must be owned) at all times, otherwise they become undiscoverable (see Ownership Management in Section 2 for details). Therefore, we will need to pass ownership (custody) of “logged out” Participant objects around as federates join and resign from the federation.

The user_handle attribute will serve not so much to identify the owner of the Participant object as to allow private messages to be sent. It designates a slot in the UserHandleSlots dimension and is very simply the 32-bit integer representation of the HLA object instance handle. In that sense, it is just as unique as the object instance’s name, and is associated with that name for the duration of the federation.
Whether we map object instance handles or object instance names (Unicode strings) to a dimension (the HLA specification, at 1516.2 4.6.1, states that each federate must provide a normalization function that maps values from the federate view of a dimension to values in the RTI view of a dimension), we face precisely the same problem: they are dynamic arrays of bytes. It is tempting to use the `hashCode()` method to map these “keys” to a 32-bit integer dimension. However, the standard `java.lang.String.hashCode()` implementation is more likely to produce collisions (between Strings of four or less characters and Strings of five or more characters, for example) than the `hla.rti1516.ObjectInstanceHandle.hashCode()`. No collisions at all are expected for handles which are 32-bit or less wide, as long as the comparisons are between same-class objects. An eventual RTI using 64-bit wide handles could allow `hashCode` collisions to occur, but even then this seems extremely unlikely. For our purposes, `ObjectInstanceHandle.hashCode()` was deemed sufficient.

One notes that the Management Object Model (MOM) specifies the `HLAfederateHandle` datatype as being a 32-bit integer. However, as the specification states, “this is a pointer to an RTI-defined programming language object, not an integer 32.” The datatype indeed represents a Java reference to a `FederateHandle` interface, and is used (within the MOM) only for the Federate dimension. In other words, the instance attributes `HLAmanager.HLAfederate.HLAfederateHandle` are `HLAhandles`, and each of those contains a `FederateHandle` reference, passed to `RTIambassador.normalizeFederateHandle` as the MOM `HLAfederateHandle` datatype.

The `chat_room_slot` attribute, finally, indicates in which chat room the Participant “is.”

### Role of the ChatRoom Objects

The ChatRoom objects represent the chat groups which are dynamically formed and dissolved as the federation executes. A logged-in user’s Participant token is “in” precisely one chat room at all times. Each ChatRoom occupies a slot in the `ChatRoomSlots` dimension, and this serves to channel communications using DDM. Ownership of ChatRoom objects is not as critical as with Participant objects, and it is expected that it will pass around. The federation ensures that ownership of a ChatRoom object (other than the Waiting Room) remains with one of the Participants in it. A ChatRoom that would become “empty” is deleted (except for the Waiting Room).
There are three special chat rooms. The “Nowhere” room is never instantiated; it is a reserved chat room slot that can be published into but to which no federate ever subscribes. It serves as a publication safety net, to prevent object attributes from reverting to the default region. The “Waiting Room” is instantiated only once, using the Name Reservation service, and is as a result never deleted, even if empty. It serves as a storage area for inactive (“logged out”) Participant objects. The “General” chat room, finally, is the default room one joins when first logging in. Like normal chat rooms, it may be created and destroyed repeatedly.

As with Participant objects, we need to ensure the uniqueness of ChatRoom object names. However, since they will possibly be created and destroyed repeatedly over the duration of the federation, HLA’s Name Reservation service cannot be used. Thus a ChatRoom’s name is not its HLA object instance name, but rather an attribute. Secondly, we need a mechanism to allocate chat room slots without causing any collisions. This is the role of the ChatRoomRegistry object.

**Role of the ChatRoomRegistry Object**

This unique object stores the list of current name-slot attributions in its single attribute. By subscribing to the single ChatRoomRegistry object instance, each federate keeps its local copy of the list up to date. It is an easy matter to scan the list for any given ChatRoom name to see if it already exists, or to scan the list for the first free slot value. Since only the owner of the list attribute can modify it, there is no risk of several federates attempting to create or delete a ChatRoom object instance simultaneously.

Ownership of the ChatRoomRegistry is used as a means of synchronization between the federates—a sort of baton or token. For example, before a federate can decide whether it needs to create a ChatRoom or not, it will obtain ownership of the ChatRoomRegistry, thus ensuring that there won’t be any changes to the list concurrent with its own consultation of the list. In that sense, ownership of an HLA object is similar (at the federation level) to the procurement of a monitor lock on a Java object (at the federate thread level).

The drawback of this approach is that it creates a bottleneck (the ChatRoomRegistry being passed around), but this is an unavoidable consequence of the need for synchronization. It would have been much worse to introduce a server-like special federate, whose only purpose would be to maintain this list and respond to creation and deletion requests, because this introduces a single point of failure and forces network traffic into a hub-and-spoke pattern.

The ChatRoomRegistry’s list attribute may seem superfluous. Since all ChatRooms are known by all federates at all times (by design), each federate is fully capable of maintaining a list of name-slot pairs from those subscriptions alone. This may be true.
On the one hand, this approach would certainly lighten the traffic load somewhat, since there would not be any need to broadcast the entire ChatRoomRegistry list attribute every time it is modified. Note that this is equivalent to breaking up the ChatRoomRegistry object (with its single ChatRoomRegistryEntries list attribute) into a series of ChatRoomRegistryEntry objects (each containing a single ChatRoomRegistryEntry attribute).

On the other hand, this approach would not improve the federation's resistance to failure (due to RTI events arriving out of sequence). When you obtain an update of the ChatRoomRegistryEntries attribute, you can rely on its value to ascertain whether a certain chat room slot is present or not, whereas if you rely on the set of subscribed ChatRoomRegistryEntry objects, you can never affirm a slot's absence, as its discovery may very well be pending.

One possible alternate mechanism for achieving this kind of certainty is the federation synchronization service group. A federate would announce its intent to create a chat room at a given slot value by constructing a unique synchronization label from that slot value and attempting to register it. If the registration succeeds, the slot is deemed "reserved" and the federate can proceed with the chat room creation process. When the chat room object is eventually deleted, the deleting federate would assert that synchronization has been achieved (for the synchronization point label corresponding to the slot), and so would the other federates upon receiving the Remove Object Instance callback. In this way the synchronization label would exist as long as the chat room slot is occupied.

**Publication and Subscription**

We are relying extensively on DDM to achieve the filtering expected of our chat application. Here is how it plays out.

The Communication interaction is sent through either a chat room slot region (one's current chat room) or a user handle slot region (the targeted user). It is subscribed through those same regions and through the Waiting Room. This latter subscription serves a single purpose: tripping the interaction scope advisories in order to ascertain whether the federate is "alone" or not. Set up this way, a joined federate will receive a **turnInteractionsOn** when at least one other federate is joined, and will receive a **turnInteractionsOff** when all other federates have resigned. The federate needs to know this when it resigns, in order to decide whether to negotiate the ownership divestiture of objects in its custody, or to destroy the federation behind it. We could also have achieved this by subscribing to the federation's MOM HLAfederate objects, but this way is simpler.

The ChatRoomRegistry's attributes are published and subscribed through the default region.

The ChatRooms' attributes are also published and subscribed through the default region.
The Participants’ attributes are published through two regions. The first is the “Nowhere” chat room slot. This serves only to ensure that a Participant’s publication regions never revert to the default area. The second is the Participant’s current chat room slot. When a Participant instance moves between chat room slots, the [un]associateRegionsForUpdates service is used to remove it from the old slot and then to add it to the new slot. If the Participant were not also associated with the “Nowhere” slot, it would revert to the default region between its withdrawal and its return. Subscription is through the “Waiting Room” (at all times) and the user’s current ChatRoom slot.

Figure 29 shows how Participant objects are distributed in the ChatRoomSlots dimension. Shown in red are those Participant objects which are currently logged-in; each one is owned by the corresponding logged-in federate. Each federate may also own (have custody of) none, one or more of the logged-out Participants, shown in black. Note that logged-out Participants all lie in the Waiting Room slot, and logged-in Participants all lie perforce elsewhere. By subscribing to the chat room slot a federate is currently “in”, it knows of the other federates sharing the chat room. By subscribing to the Waiting Room slot, it is readily capable of accepting custody of those Participants should the need arise.
Figure 29. The Participant objects and the ChatRoomSlots dimension
Assertions

Each active federate:

- represents a “logged-in” user;
- is “in” one ChatRoom at all times (the General chat room or any one of the user-created chat rooms);
- knows all ChatRoom object instances (the Nowhere room is virtual);
- knows the unique ChatRoomRegistry object instance;
- owns the Participant object instance that bears its user name (and will not relinquish it while logged in);
- knows the Participant object instances that are “in” the same ChatRoom;
- knows the Participant object instances that are dormant (“logged-out”);
- listens to the Communication interaction through a DDM channel corresponding to its user handle slot;
- listens to the Communication interaction through a DDM channel corresponding to its chat room slot; and
- listens to the Communication interaction advisories through a DDM channel corresponding to the Waiting Room slot.

Each Participant object:

- represents a unique (by user name and user slot) federate;
- if “logged-out”, is “in” the Waiting Room and owned by some federate;
- if “logged-in”, is “in” a chat room other than the Waiting Room (e.g. the General chat room or any one of the user-created chat rooms);
- is published through a DDM channel corresponding to its user handle slot;
- is published through a DDM channel corresponding to its chat room slot; and
- is published through a DDM channel corresponding to the Nowhere slot.

When a Participant switches chat rooms, it is simply unassociated from its old chat room slot (so that it is now published only through the Nowhere slot) and then associated with its new chat room slot. When a Participant logs out, it simultaneously switches to the Waiting Room slot. When it logs in, it is either created in the General chat room or switched from the Waiting Room to the General chat room.
Each ChatRoom object:

- represents a unique (by name and slot) chat room;
- is created when a Participant moves “in”; and
- is deleted when the last Participant moves “out.”

Each Communication interaction occurrence:

- represents a message being sent across the chat network;
- if public, is sent through a DDM channel corresponding to the sender’s current chat room slot; and
- if private, is sent through a DDM channel corresponding to the target’s user handle slot.

**Federate Life Cycle**

Federates are created and initialised in the not-joined state, obviously. They join the federation with the sole intent of logging in. Having joined, their first step is to obtain the values of the various static FOM handles (interaction and object class handles, parameter and attribute handles, dimension handles).

Generally, a federate’s ownership, subscription and publication interests are dynamic and thus a variety of objects are created just before being consumed by an RTI ambassador service invocation. For example, the Subscribe Object Class Attributes With Regions service consumes an attribute-set region-set pair list, so the federate must generate attribute handle sets and region handle sets. Each region included in the latter forces the federate to generate a dimension handle set, to set the range bounds over each dimension and to commit the region modifications. Although straightforward, these preparations can nevertheless be lengthy.

In our case, the ownership, subscription and publication policies are static, so we generate as many of these objects as possible during initialization: attribute handle sets, dimension handle sets, as well as some of the region handle sets, regions and attribute-set region-set pair lists (i.e. those dealing with the three reserved chat room slots).

**The Login Process**

The preliminaries consist in publishing and subscribing to the federation’s four classes. First, the federate publishes the Communication interaction class and subscribes to it through the waiting room slot.
Next, the federate publishes the ChatRoomRegistry class and then attempts to reserve the ChatRoomRegistry’s name through the Name Reservation service. If the reservation succeeds, the ChatRoomRegistry did not exist and it is the responsibility of this federate to create it. If it fails, the federate simply waits for the object discovery to occur.

The federate repeats the reservation/creation/discovery process for the waiting room. These two objects are the only “fixtures” of the federation.

The federate concludes the login preliminaries by publishing the Participant class and subscribing to Participants in the waiting room.

The login proper then begins. The federate attempts to reserve the chosen username through the Name Reservation service. If the reservation succeeds, the username is new to the federation, so the federate can create the corresponding Participant object and log itself into the general chat room. If it fails, the username already has a corresponding Participant object floating about, which may be logged-in or not.

If logged-out, that Participant will be in the waiting room and will thus be discovered; once discovered, the federate can acquire it (if it does not already own it) and proceed to log it into the general chat room. The login process is then complete.

If logged-in, the Participant will be in the general chat room or some other chat room; even if known the acquisition attempt will fail. At this point the login process is a failure and the federate resigns from the federation.

Although the login process is relatively quick, it does nevertheless take a finite time. The federation runs the risk of putting itself in a pathological state if it had only one joined federate left and it logs out simultaneously with the log in of another. As stated above, the first step of the login process is the publication of the Communication interaction, which triggers an interaction scope advisory at the logging-out federate. The logging-out federate may then tender ownership of its objects (the ChatRoomRegistry, the waiting room ChatRoom and others) to the logging-in federate before the latter has had time to publish the relevant classes, discover the objects and receive their values. Conversely, the logging-out federate could destroy its owned objects before it receives the advisory, and the logging-in federate will then be unable to recreate the federation’s standard objects.

Another possible problem is a race for the same pre-existing (logged-out) Participant: if two users attempt to log in at about the same time, the slower one may see the desired object go out of scope (be acquired by the faster federate) after it has already committed itself (after checking for a number of pre-conditions) to wait for the acquisition. Conversely, duplicate ChatRooms (except for the waiting room, which is named) are possible if federates are racing to create them (e.g. both pass the non-existence check points and commit themselves to creating the chat rooms). See the MyChat.registerParticipant() code in Annex E for details.
Skilful use of a federation synchronization point could prevent these kinds of race conditions. The very first action a logging-in federate would take would be to register a synchronization point named something like “Login in progress”; other federates would achieve the point immediately, while logging-out federates would await the Federation Synchronized callback before proceeding. A similar “Logout in progress” synchronization point would also be used. These precautions were not taken with this work for expediency’s sake.

**The Logout Process**

As soon as it starts the logout process, the federate sets an internal flag that will have it decline offers of ownership; as discussed in the login process, this could be a problem if all remaining joined federates try to log out simultaneously.

First, the federate proceeds as if switching chat rooms (see below), from the current one to the “waiting room”, except that no subscription to the Communication interaction through the latter occurs. While “nowhere”, the federate’s Participant avatar has its `logged_in` attribute changed in addition to `chat_room_slot`.

The federate unsubscribes from any remaining subscriptions and then, if the interaction scope advisories indicate that there are other joined federates, it negotiates the divestiture of any owned objects to those federates (in order to prevent the potential orphaning of any instances), including, of course, its Participant avatar. Once that is done, any remaining publications can be shut down, and the federate can now resign freely from the federation. The last federate to resign from the federation deletes all objects from the federation and then destroys the federation.

**The Chat Room Switching Process**

When a user switches chat rooms, the first sub-task is to find out if the new chat room already exists or not. If not, it needs to be created, a fairly simple matter of creating the ChatRoom object and registering it with the federation.

The federate now withdraws its Participant avatar from the current chat room slot, which leaves it published solely through the nowhere room. This means it goes out of scope for all other joined federates. Concurrently, the federate’s subscription to the Communication interaction through the old chat room slot is shut down.

If the chat room left behind becomes empty, it must be deleted. This sub-task entails obtaining ownership of the ChatRoomRegistry in order to update its list of extant chat rooms.

To decide if the chat room is now empty, we count the number of Participant objects (out of the known set) that are in it (according to their `chat_room_slot` attributes). Once that count is secured, the federate can unsubscribe the Participant class from the current chat room (this causes any co-Participants to go out of scope, so attribute updates are no longer received for those).
A race condition exists at this point: it is possible another federate may join the chat room after it has been determined to be empty but before the actual deletion occurs. The solution would be to modify the FOM to add a `count` attribute to the ChatRoom object, which would reflect its occupancy on a continuous basis. A little extra overhead would be incurred as ownership of that attribute gets passed around (whenever a federate joins or leaves the chat room).

The potential deletion process would then be as follows. First, check the chat room’s `count`; if greater than 1, no deletion is required and we are done. Otherwise, we must acquire the ChatRoomRegistry (since we intend to strike the chat room from its `list`). Once that is done, check the `count` again, just in case it has changed in the meantime. If deletion is still indicated, obtain ownership of the ChatRoom and check the `count` one last time. If still holding at 1, we can delete it (there is no point in updating the `count` to zero and deleting the object immediately afterwards). Other federates wanting to join the chat room will try to acquire it, so if we receive an ownership release request we will simply ignore it and forge ahead with deletion. This will cause the evocation of RemoveObjectInstance at the other federates, which will tell them “back to square one, the requested ChatRoom doesn’t exist any more.”

Once the old chat room has been deleted or simply left behind, the federate’s Participant avatar has its `chat_room_slot` set locally to the new chat room, and then it is re-associated (published) through the new chat room slot, the Communication interaction is subscribed through the new chat room slot, and the Participant class is subscribed through the new chat room slot (this reveals the co-Participants in the new chat room, if any). The AutoProvide switch means an attribute value update request occurs immediately after the Participant is discovered by or goes back into scope for other joined federates (i.e. those whose Participants already are in the new chat room).

**Ownership Transfer Processes**

As the federation evolves, it is expected that the ownership of most objects will be transferred repeatedly between the joined federates. The ChatRoomRegistry is passed around whenever a chat room needs to be added or deleted. ChatRoom objects are transferred so that they are always owned by one of the federates “in” that room. Logged-in Participants are each owned by the federate they represent, while logged-out Participants are divested by resigning federates (there are no particular criteria for ownership besides simply being joined).
The process of ownership is straightforward. Each federate receiving a request for ownership assumption checks if it is in the “normal” state (i.e. not in the process of logging out) and that the class is currently subscribed to (because a federate’s local copy of an unsubscribed object is probably outdated as far as the object’s attribute values go). Once these formalities are out of the way, the key step is establishing the to-be-acquired object’s publication regions before requesting tentative ownership (using the Attribute Ownership Acquisition If Available service). This ensures continuity of an object’s publication regions. Note that this is applicable only to Participant objects, since all other objects are published through the default region.

When prompted to acquire a ChatRoom object, it is the federate’s responsibility to check if it is fully eligible (i.e. is the chat room being offered either the waiting room or the client’s current chat room?), as the RTI offers ownership to all joined federates that know of the object and that publish its class—a broader criterion.
5. Conclusions

The IEEE 1516 series of standards offers great promise, and is a great improvement over the preceding 1.3 standard. Designing a flexible middleware standard that could accommodate vastly differing simulation contexts without presuming on the underlying semantics was a daunting task, and the IEEE must be congratulated on succeeding at it.

The use of state diagrams is crucial in understanding how each federate and the RTI are expected to work together. They help tremendously in coming to grips with the subtle implications of seemingly innocuous concepts such as “ownership transfer.”

Few blind spots remain with IEEE 1516, and these should be easily remedied by the active IEEE/SISO involvement in evolving the standard. The most obvious one is the need for RTI-supplied standard encoding and decoding facilities for value transport. It is surprising that this need was not recognised (or explained away) in the published standard.

The IRC example application turned out to have rich implications with respect to DDM, potential race conditions, and simultaneous actions (imagine all logged-in users logging out “at once”); being time-less exacerbated some of these. A new appreciation of the federation synchronization services emerges from this.

JACK’s built-in thread management turned out to avoid a slew of potential problems. The Java interface proved very easily adaptable, once the appropriate design decisions were identified.
This page intentionally left blank.
Annex A – Notes on the IEEE 1516-2000 Series of Standards

This annex comments on the IEEE 1516-2000 Series of Standards in the same manner as the U.S. Department of Defense (DoD) Interpretations of the IEEE 1516-2000 series of standards: Release 2 [39].

IEEE 1516-2000 Series

Multiple Federation Executions

The subject of a single federate application running multiple federates, possibly spread over several distinct federate executions, is touched upon only in an indirect way in the 1516 specification. Recall that the RTI can handle several federation executions running at once, and that each federation execution may have any number of joined federates. Each of these is an emanation of a federation application but, although each joined federate is logically distinct, nothing prevents several of them from being emanations of a single federation application instance. This is somewhat analogous to a single instance of a word processing application handling multiple documents at once, each document containing embedded objects that call upon a single third party service (e.g. multimedia clips).

The key reference is the brief mention, in 1516.1-2000, 1.4.3 General nomenclature and conventions, that “For all joined federate-initiated services in this specification, except Create/Destroy/Join Federation Execution, an implied supplied argument is a joined federate’s connection to a federation execution. For all RTI-initiated services, an implied supplied argument is also a joined federate’s connection to a federation execution.”

In practice, the behaviour implemented by the existing RTIs is the following. The RTIambassador interface obtained from the RTI represents implicitly the federate’s connection to a federation execution, in the sense that once the federate has joined a federation execution, that particular RTIambassador is committed and will throw a FederateAlreadyExecutionMember exception if a new Join is attempted, regardless of the specified federation execution, federate type or FederateAmbassador instance.
Nothing prevents the federate from obtaining a distinct \texttt{RTIambassador} interface and using that to Join the same (or another) federation execution as a logically different federate. Although the specification does permit the same \texttt{FederateAmbassador} instance to be used for both Joins, it is extremely unlikely that this will be done intentionally, as the \texttt{FederateAmbassador} instance will be hard pressed to know which federation execution is invoking its callbacks, or, in the case where the \texttt{FederateAmbassador} instance is used by two federates of the same federation execution, which of the two federates the RTI is sending messages to (and therefore which \texttt{RTIambassador} instance the federate application should use in replying).

Unless the \texttt{FederateAmbassador} is carefully designed with this mode of operation in mind (setting up thread service variables to distinguish between the federate service thread contexts seems required as a start), utter chaos and eventual failure of the federate application is certain to ensue.

\section*{Simple Inheritance}

A feature common in object-oriented languages is the ability to \textit{hide}, \textit{override} or \textit{overload} an inherited member of a class. In the HLA “objects are defined entirely by their identifying characteristics (attributes) [...]. These are, in OO parlance, data members of the class” (1516 1. Overview). Java calls data members \texttt{fields} and, consequently, the only pertinent mechanism is \texttt{hiding}, wherein one redeclares an inherited field, possibly giving it a different data type. This is not legal in HLA.

Although one can reasonably derive this result from the definitions for the “available/inherited attributes/parameters” (3.1.5; 3.1.7; 3.1.39; 3.1.40) as well as the additional statements 1516.1 1.4.3 (“The handle for a class attribute that is inherited shall be the same as the handle that is assigned to the class attribute in the object class in which it is declared”) and 1516.2 4.2.1 (“Subclasses shall always inherit the attributes of their superclasses, and they may possess additional attributes to provide the desired specialization”), the clincher is 1516.2 4.4.2: “The names assigned to attributes of any particular object class shall not duplicate (overload) the names of attributes of this class or any higher level superclass”).

It would nevertheless have been useful to mention this more explicitly and earlier. To give but one example of the confusion possible over this point, the Pitch Visual OMT 1516 application, used to create and edit Federation Object Model (FOM) Document Data (FDD) files, allows one to hide inherited fields and nevertheless declares the file as valid when using its Check Consistency tool. It is only when one \texttt{tries to createFederationExecution} using the resulting FDD that one gets an \texttt{ErrorReadingFDD} exception.
Therefore, change the third paragraph of 1516 (Framework and Rules) 1.3
Relationship of HLA and object-oriented concepts to:

As discussed above, HLA object classes are described by the attributes that are
defined for them. These are, in OO parlance, data members of the class. These
attributes, which are abstract properties of HLA object classes, are referred to as
class attributes. Another important difference between HLA and OOAD concepts is
that an inherited class attribute cannot be hidden; that is to say, a class attribute
defined at a given class is inherited by its subclasses and may not be redeclared.
HLA object instances are spawned via an HLA service using an HLA object class as
a template. Each attribute contained by an HLA object instance is called an
instance attribute.

**Declaring Dimensions**

Clause 4.6 of 1516.2 (Object Model Template Specification) describes how one sets
up dimensions within the OMT. Interestingly, although dimension upper bounds are
described in terms of “non-negative integers”, the XML specification expects a
string and the range of legal strings is not defined. As a result, the Pitch pRTI 1516
implementation is unable to read correctly an HLAinteger32BE or
HLAinteger64BE dimension because it attempts to translate the string into a Java
int, whose maximum value is $2^{31} - 1$ (~2 147 483 647). Since the upper bound is
excluded, a full 32-bit dimension would have a range of [0, 2,147,483,648] and pRTI
1516 throws an ErrorReadingFDD exception with the message “Invalid
dimension upper bound.” Even if pRTI1516 correctly handled the upper bounds
as Java longs, which is what the API specifies for its GetDimensionUpperBound
service, there would be a problem with the largest possible upper bound, which
would be $2^{63}$ (whereas the Java long type’s upper bound is $2^{63} - 1$).

Note also that a floating-point dimension is legal, although the upper bound of the
RTI view of such a dimension is still specified as a non-negative integer.
Considerable quantization will occur if linearly normalizing such floating-point
values over spans approaching the full domain. It should be noted, however, that
when the bit-widths are the same, the lack of precision inherent in the floating-point
representation itself (at large magnitudes) is much worse than that imposed by the
quantization.
Encoding/Decoding Values

When sending an interaction or an attribute instance update, the values are expected to be encoded into opaque byte arrays; likewise, a received interaction or attribute instance reflection supplies values as opaque byte arrays. If the values are of certain types, the RTI supplies the encode and decode methods necessary for the translation. Unfortunately, the standard only specifies eleven datatypes: AttributeHandle, DimensionHandle, FederateHandle, InteractionClassHandle, LogicalTime, LogicalTimeInterval, ObjectClassHandle, ObjectInstanceHandle, OrderType, ParameterHandle and TransportationType. Clearly, any federates that partake of a given FOM must also share the relevant encode/decode methods. Even then, it is not clear which encode/decode methods should be used when subscribing to MOM objects and interactions.

Clause 4.12.3 of 1516.2 (Object Model Template Specification) describes, in table 23, the basic data representation formats (HLAinteger16BE, HLAinteger32BE, HLAinteger64BE, HLAfloat32BE, HLAfloat64BE, HLAoctetPairBE, HLAinteger16LE, HLAinteger32LE, HLAinteger64LE, HLAfloat32LE, HLAfloat64LE, HLAoctetPairLE, HLAoctet). The following clauses describe the remaining datatypes: simple (HLAASCIIchar, HLAunicodeChar, HLAbyte), enumerated (HLABoolean), fixed array (none mandated), variable array (HLAASCIIstring, HLAunicodeString, HLAopaqueData), fixed record (none mandated) and variant record (none mandated). Note that a user-defined datatype could be an arbitrarily complex construct (e.g. a variable array of variant records). The byte array representations are specified in detail later on, at clause 4.12.9. However, no encode/decode methods are supplied, opening the door to inefficient or erroneous implementations.

Note also that 1516.1 11.6 (MOM OMT Tables) defines a number of additional datatypes that would need to be supported if one were to make full use of the MOM:

- **Simple:** HLAcount, HLAfederateHandle, HLAmsec, and HLAseconds

- **Enumerated:** HLAfederateState, HLAorderType, HLAownership, HLAresignAction, HLAserviceGroupName, HLAswitch, HLAasyncPointStatus, and HLAtimeState

- **Variable Array:** HLAargumentList, HLAhandle, HLAhandleList, HLAinteractionCounts, HLAinteractionSubList, HLAlogicalTime, HLAobjectClassBasedCounts, HLAasyncPointFederateList, HLAasyncPointList, HLAtimeInterval, and HLAtransportationName

- **Fixed Record:** HLAinteractionCount, HLAinteractionSubscription, HLAobjectClassBasedCount, and HLAasyncPointFederate
The RTI Ambassador ought to supply factories for each and every one of the OMT-described datatypes, so that a common set of encode/decode methods may be used throughout the federation. This could be done through a single RTI ambassador support service along the lines of `GetDatatypeInstance(String DatatypeName)`, where `DataTypeInterface` would include methods such as `getIntValue()`, `setIntValue(int value)`, `encode(byte[] [, offset])`, `decode(byte[] [, offset])`, `getName()`, `getDatatypeType() (simple, enumerated, HLAfixedArray, HLAvariableArray, HLAfixedRecord, HLAvariantRecord)` and so on. Each federate could then obtain an encoding/decoding interface from the RTI for any arbitrary datatype declared in the FDD, and be assured that any values transmitted as part of an interaction or attribute value update will be correctly interpreted by the other federates. Until such an extension to the IEEE 1516 specification appears, one is forced to implement separately the encoding scheme as laid out in 1516.2 4.12.9 (Predefined encodings for constructed datatypes).

We believe one of the SISO HLA Product Development Groups (PDGs) is working on precisely this problem (SISO has recently become the custodian of HLA at IEEE’s behest).

**Names**

The only part of the standard that defines what is an allowable name is 1516.2 (Object Model Template Specification), at clause 3.3.1. Even then, the clause omits to state whether the definition applies to object instance names, presumably because these are the only designators not appearing in the FOM Document Data (FDD). The `IllegalName` exception thrown by the `reserveObjectInstanceName` service is laconically described as occurring when “the name argument begins with "HLA".” This is obviously incomplete. Assuming that the 1516.2 rules apply, add to clause 3.1 (Definitions) the following (renumbering subsequent definitions accordingly):

3.1.52 name: A human-readable hopefully significant character string used to designate an object or interaction class, an attribute, a parameter, a datatype (including enumerators and enumerated values), a record field, a dimension, a transportation or ordering type, a synchronization point or federation save label, a federation execution, a federate type, or an object instance. There are two forms of names: simple names and qualified names. A qualified name consists of a name, a "" (period [Unicode $002E$]), and a simple name. They are used to designate object and interaction classes by tracing their ancestry from their respective root classes. Simple names are constructed from a combination of letters (Latin letters “A” through “Z” [Unicode $0041$ through $005A$] and “a” through “z” [Unicode $0061$ through $007A$]), digits (“0” through “9” [Unicode $0030$ through $0039$]), hyphens [Unicode $002D$], and underscores [Unicode $005F$]. A simple name cannot begin with a digit or hyphen. Case is significant. The name 'na' and all names beginning with "HLA" (including all case variations in both instances) are reserved.
Examples:

```
HLAfederate  is a simple name (from the MOM)
HLAobjectRoot.HLAmanager.HLAfederate is a qualified name
hLa-Immunological-Complex is an illegal name ("hla" prefix reserved)
h-L-a-Immunological-Complex is a legal name
nA  is an illegal name ("na" reserved)
nA_nanoampere is a legal name
```

Note that the definition proposed above is more restrictive than clause 1516.2.3.3.1 as it allows only the ASCII subset. If one wishes to allow the full XML 1.1 [40] gamut, it should be explicitly listed:

A simple name consists of a NameStartChar followed by zero or more NameChar.

The set of NameStartChars consists of the following Unicode characters:
- the colon (""), $003A;
- the upper case Latin letters 'A' through 'Z' inclusive, $0041-$005A;
- the underscore '_', $005F;
- the lower case Latin letters 'a' through 'z' inclusive, $0061-$007A;
- the accented Latin letters up to and including the spacing modifier letters (except the multiplication and division signs), $00C0-$00D6, $00D8-$00F6, and $00F8-$02FF;
- the Greek letters (except the Greek question mark), $0370-$0370, $037F-$1FFF;
- a wide selection of other characters $200C-$2000, $2070-$21SF, $2COO-$2FEF, $3001-$07FF, $F900-$FDCF, $FDFO-$FFFD, and $10000-$EFFFF.

The set of NameChars consists of the set of NameStartChars plus the following supplementary Unicode characters:
- the hyphen '-', $002D;
- the digits '0' through '9' inclusive, $0030-$0039;
- the mid-dot '.', $00B7;
- the diacritics $0300-$036F; and
- the general punctuation signs ordinarily '_' and character tie '~', $203F-$2040.

While on the subject of names and labels, it seems odd that various services which accept String input (such as createFederationExecution, requestFederationSave, etc.) do not throw IllegalName-like exceptions when the supplied String violates the name construction rules. The only services (in addition to reserveObjectInstanceName) that currently do throw such exceptions are 10.18 getTransportationType and 10.20 getOrderType.

To give an example of the pathologies that may arise as a consequence of this omission, consider that, as the standard is currently written, an RTI must accept any form of save or synchronization label, including ones with embedded control or line break characters. If an RTI simply filters these out, then the situation occurs where two federation-defined labels considered different by the federation (because control characters differ or are placed differently within the label) are treated as congruent by the RTI.
Therefore, add the following exceptions (it may be useful to define all of these new exceptions as subclasses of `IllegalName`):

```java
// 4.2
[...] createFederationExecution [...] 
   throws IllegalFederationExecutionName [...] 

// 4.3
[...] destroyFederationExecution [...] 
   throws IllegalFederationExecutionName [...] 

// 4.4
[...] joinFederationExecution [...] 
   throws IllegalFederateType, IllegalFederationExecutionName [...] 

// 4.6
[...] registerFederationSynchronizationPoint [...] 
   throws IllegalLabel [...] 

// 4.9
[...] synchronizationPointAchieved [...] 
   throws IllegalLabel [...] 

// 4.11
[...] requestFederationSave [...] 
   throws IllegalLabel [...] 

// 4.18
[...] requestFederationRestore [...] 
   throws IllegalLabel [...] 

// 6.2
[...] reserveObjectInstanceName [...] 
   throws IllegalObjectInstanceName [...] 

// 6.4
[...] registerObjectInstance [...] 
   throws IllegalObjectInstanceName [...] 

// 9.5
[...] registerObjectInstanceWithRegions [...] 
   throws IllegalObjectInstanceName [...] 

// 10.2
[...] getObjectClassHandle [...] 
   throws IllegalObjectClassName [...] 

// 10.4
[...] getAttributeHandle [...] 
   throws IllegalAttributeName [...] 
```

1.4.3 General nomenclature and conventions

Replace:

For all joined federate-initiated services in this specification, except 4.2, Create Federation Execution, 4.3, Destroy Federation Execution, and 4.4, Join Federation Execution, an implied supplied argument that is a joined federate's connection to a federation execution.

With:

For all joined federate-initiated services in this specification, except 4.2, Create Federation Execution, 4.3, Destroy Federation Execution, and 4.4, Join Federation Execution, an implied supplied argument is a joined federate's connection to a federation execution.

4.4 Join Federation Execution

The consequences of the “implied supplied arguments” mentioned in 1.4.3 are worked out incorrectly in the Join Federation Execution service description as currently written. To the first paragraph, add:

[...] Until the Resign Federation Execution service is invoked by the federate, the RTI ambassador interface instance used to invoke this service is considered committed to the specified federation execution. Although the 4.1 Create Federation Execution and 4.2 Destroy Federation Execution services may still be freely invoked, any invocation of this service while committed will fail, regardless of the explicitly supplied arguments.
Change 4.4.3 b) to read:

b) The federate is not joined to any federation execution.

Change 4.4.5 a) to read:

a) The federate is already joined to a federation execution.

5.2/5.3 [Un]Publish Object Class Attributes

The Annex B RTIambassador.java file introduces a potentially dangerous source of confusion when it labels (on five occasions) an AttributeHandleSet argument “attributeList”. The Java class java.util.List is quite different from the java.util.Set class. Therefore, replace:

```java
// 5.2
public void publishObjectClassAttributes (ObjectClassHandle theClass,
                                          AttributeHandleSet attributeList)
{...
// 5.3
public void unpublishObjectClassAttributes (ObjectClassHandle theClass,
                                          AttributeHandleSet attributeList)
{...
// 5.6
public void subscribeObjectClassAttributes (ObjectClassHandle theClass,
                                          AttributeHandleSet attributeList)
{...
public void subscribeObjectClassAttributesPassively (ObjectClassHandle theClass,
                                          AttributeHandleSet attributeList)
// 5.7
{...
public void unsubscribeObjectClassAttributes (ObjectClassHandle theClass,
                                          AttributeHandleSet attributeList)
{...
```
With:

```java
// 5.2
public void publishObjectClassAttributes (
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes)
[...]
// 5.3
public void unpublishObjectClassAttributes (
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes)
[...]
// 5.6
public void subscribeObjectClassAttributes (
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes)
[...]
public void subscribeObjectClassAttributesPassively (
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes)
// 5.7
[...]
public void unsubscribeObjectClassAttributes (
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes)
[...]
```
5.6/5.7 [Un]Subscribe Object Class Attributes

See 5.2/5.3 [Un]Publish Object Class Attributes, above.

5.8/5.9 [Un]Subscribe Interaction Class

The Annex B RTIambassador.java file introduces a minor source of confusion when it labels (on eight occasions) an InteractionClassHandle argument “theClass”, a name used everywhere else to designate an ObjectClassHandle. Other services use “theInteraction” for InteractionClassHandle arguments. Therefore, replace:

```java
// 5.8
public void subscribeInteractionClass (InteractionClassHandle theClass) [...] public void subscribeInteractionClassPassively (InteractionClassHandle theClass) [...] // 5.9
public void unsubscribeInteractionClass (InteractionClassHandle theClass) [...] // 6.14
public void changeInteractionTransportationType (InteractionClassHandle theClass,
                                                TransportationType theType) [...] // 8.24
public void changeInteractionOrderType (InteractionClassHandle theClass,
                                        OrderType theType) [...] // 9.10
public void subscribeInteractionClassWithRegions (InteractionClassHandle theClass,
                                                  RegionHandleSet regions) [...] public void subscribeInteractionClassPassivelyWithRegions (InteractionClassHandle theClass,
                                                                                     RegionHandleSet regions) [...] // 9.11
public void unsubscribeInteractionClassWithRegions (InteractionClassHandle theClass,
                                                   RegionHandleSet regions) [...]
```
With:

```java
// 5.8
public void subscribeInteractionClass (InteractionClassHandle theInteraction) [...] public void subscribeInteractionClassPassively (InteractionClassHandle theInteraction) [...] // 5.9
public void unsubscribeInteractionClass (InteractionClassHandle theInteraction) [...] // 6.14
public void changeInteractionTransportationType (InteractionClassHandle theInteraction, TransportationType theType) [...] // 8.24
public void changeInteractionOrderType (InteractionClassHandle theInteraction, OrderType theType) [...] // 9.10
public void subscribeInteractionClassWithRegions (InteractionClassHandle theInteraction, RegionHandleSet regions) [...] public void subscribeInteractionClassPassivelyWithRegions (InteractionClassHandle theInteraction, RegionHandleSet regions) [...] // 9.11
public void unsubscribeInteractionClassWithRegions (InteractionClassHandle theInteraction, RegionHandleSet regions) [...]```

### 6.1 Overview

To clarify the important distinction between known and owned object instances, insert this paragraph after the one on orphan object instances (p. 71):

However, an object instance can become wholly unowned without becoming an orphan. It is important to note that object instances that are known remain known even if they become undiscoverable. As long as a joined federate that knows of the object instance does not invoke Local Delete Object Instance, it continues to know of the object instance and ownership acquisition of some or all of the latter's instance attributes remains possible.
6.3 Object Instance Name Reserved

A clarification is needed as to whether deleting an object instance frees up its object instance name for re-use. The text of clause 6.3 implies that it does not: "[...] no joined federate joined to the current federation execution shall subsequently receive a successful reservation for that name". That is certainly the way the Pitch pRTI1516 interprets it. Under certain circumstances, this behaviour is undesirable. For instance, if a reserved name is used to represent a “username token”, then that username cannot be re-used (after a “log-off”, say) for the remainder of the federation execution.

Also, replace the first sentence:

Notifies the joined federate whether the name provided in a previous invocation of Register Object Instance Name service has been reserved, which shall mean that the name is federation execution-wide unique.

With:

Notifies the joined federate whether the name provided in a previous invocation of Reserve Object Instance Name service has been reserved, which shall mean that the name is federation execution-wide unique.

6.7 Reflect Attribute Values

In all forms, it seems odd that the federate is not allowed to throw the InvalidOrderType or InvalidTransportationType exceptions. When the service conveys a MessageRetractionHandle, the InvalidMessageRetractionHandle exception should be expected. According to the specification, the two forms that convey a LogicalTime and are allowed to throw the InvalidLogicalTime exception should do so only if the received ordering is TIMESTAMP.

These problems are common to the C++ and Java annexes.
6. 10/6.12 [Local] Delete Object Instance

The Annex B RTIambassador.java file introduces a minor source of confusion
when it labels (on three occasions) an ObjectInstanceHandle argument
"objectHandle". The other services all use "theObject". Therefore, replace:

```java
6.10
public void deleteObjectInstance (ObjectInstanceHandle objectHandle,
byte[] userSuppliedTag)
[...]
public MessageRetractionReturn deleteObjectInstance (ObjectInstanceHandle objectHandle,
byte[] userSuppliedTag,
LogicalTime theTime)
[...]
6.12
public void localDeleteObjectInstance (ObjectInstanceHandle objectHandle)
[...]
```

With:

```java
6.10
public void deleteObjectInstance (ObjectInstanceHandle theObject,
byte[] userSuppliedTag)
[...]
public MessageRetractionReturn deleteObjectInstance (ObjectInstanceHandle theObject,
byte[] userSuppliedTag,
LogicalTime theTime)
[...]
6.12
public void localDeleteObjectInstance (ObjectInstanceHandle theObject)
[...]
```
6.14/8.24 Change Interaction Transportation/Order Type

See 5.8/5.9 [Un]Subscribe Interaction Class, above.

7.3 Negotiated Attribute Ownership Divestiture

The description is incomplete. As the DoD Interpretations clearly indicate, the Unconditional Attribute Ownership Divestiture service is a fifth, legal way to “divest ownership by other means.” The second paragraph is thus replaced by¹:

A request to divest ownership shall remain pending until the request is completed (via the Request Divestiture Confirmation † and Confirm Divestiture services), or the requesting joined federate successfully cancels the request (via the Cancel Negotiated Attribute Ownership Divestiture service), or the joined federate divests itself of ownership by other means (i.e. the Attribute Ownership Divestiture If Wanted, Unconditional Attribute Ownership Divestiture or Unpublish Object Class Attributes services). A second negotiated divestiture for an instance attribute already in the process of a negotiated divestiture shall not be legal.

9.5 Register Object Instance With Regions

The Annex B RTI ambassador.java file introduces a minor source of confusion when it labels a String argument “theObject”. The Register Object Instance service uses, more appropriately, “theObjectName”. Therefore, replace:

```java
// 9.5
[...]
public ObjectInstanceHandle registerObjectInstanceWithRegions (ObjectClassHandle theClass,
        AttributeSetRegionSetPairList attributesAndRegions,
        String theObject)
[...]
```

With:

```java
// 9.5
[...]
public ObjectInstanceHandle registerObjectInstanceWithRegions (ObjectClassHandle theClass,
        AttributeSetRegionSetPairList attributesAndRegions,
        String theObjectName)
[...]
```

¹ In the IEEE 1516 documents, the printer’s dagger (†) is used to denote the callbacks; that is to say, the federate-implemented, RTI-initiated services.

See 5.8/5.9 [Un]Subscribe Interaction Class, above.

Logical Time, Time Stamps, and Lookahead

(pp. 259+) The definitions given for the Java interfaces LogicalTime, LogicalTimeInterval, LogicalTimeFactory and LogicalTimeIntervalFactory suffer from a number of inconsistencies when one compares the 12.4.2.23 clause with the (normative) Java API supplied by Annex B. There are further inconsistencies when one compares the Java clause and annex with the corresponding C++ and Ada 95 clauses and annexes. Specifically:

**Interpretation 1**

According to 12.4.2.23, "Methods [...] are provided [...] to set an instance to the initial and final [Logical Time] values. A method to set one instance to the same value as another is provided as well". The C++ and Ada 95 clauses (12.5.2.23 and 12.3.2.24, respectively) describe the same methods and their Annexes (C - LogicalTime.h and A, respectively) declare them (e.g. for C++: setInitial, setFinal and setTo). By contrast, the Java API is missing three methods from its LogicalTime interface:

```java
/**
 * Sets the value to <code>initialTime</code>.
 */
public void setInitial();

/**
 * Sets the value to <code>finalTime</code>.
 */
public void setFinal();

/**
 * Sets the value to equal the <code>other</code>'s.
 * @param other A <code>LogicalTime</code> whose value is
 * used to set <code>this</code>
 * @throws InvalidLogicalTime
 */
public void setTo(LogicalTime other) throws InvalidLogicalTime;
```
**Interpretation 2**

The Ada 95 clause (12.3.2.24) is the only one that reads "one instance of a Logical_Time may be subtracted from another, but not added. The result is of type Logical_Time_Interval". On the other hand, all three annexes define the method (e.g. "subtract" for C++ and "distance" in Java). Thus the C++ clause (12.5.2.23) should have this sentence added to its third paragraph:

Finally, a method is provided to obtain the RTI::LogicalTimeInterval separating two instances of RTI::LogicalTime.

And the Java clause (12.4.2.23) should have this sentence added to its third paragraph:

Finally, a method is provided to obtain the LogicalTimeInterval separating two instances of LogicalTime.

**Interpretation 3**

The LogicalTimeFactory is described as having "[..] a method to create a LogicalTime instance whose value is the user-defined initial value". The notion of "user" is misleading; replace the sentence (in the Java and C++ clauses) with:

The LogicalTimeFactory interface has a method to create a LogicalTime instance whose value is the federate-developer-defined initial value.

**Interpretation 4**

The aforementioned LogicalTimeFactory method is described in the C++ (12.5.2.23) and Java (12.4.2.23) clauses, while the Ada 95 clause (12.3.2.24) is mute on the subject, the concept of Factory class being alien to that language. Annex C - LogicalTimeFactory.h declares makeInitial as expected. Java’s Annex B, however, declares two Factory methods, makeInitial and makeFinal. Either the extraneous Java method should be deleted from the interface, or the specification should be enhanced to include the makeFinal method for C++.
**Interpretation 5**

The LogicalTime manipulation methods bear radically different names in each API:

<table>
<thead>
<tr>
<th>Ada</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>add</td>
<td>increaseBy</td>
</tr>
<tr>
<td>-</td>
<td>subtract</td>
<td>decreaseBy</td>
</tr>
<tr>
<td>-</td>
<td>distance</td>
<td>subtract</td>
</tr>
</tbody>
</table>

One must question the wisdom of such inconsistencies, since most other method names are immediately recognizable between APIs (e.g. Set_To_Epsilon vs. setEpsilon vs. setEpsilon).

**Interpretation 6**

The description of LogicalTimeInterval should stress that it is used to represent strictly non-negative values; that is to say, it represents the magnitude of a logical time interval. This is crucial for the proper implementation of such methods as C++’s LogicalTime.increaseBy, LogicalTime.decreaseBy, LogicalTime.subtract and LogicalTimeInterval.subtract. The LogicalTimeInterval constructor supplied in sub-clause B.3 should be modified consequently:

```java
ExampleLogicalTimeInterval(long value)
{
    _value = java.lang.Math.abs(value);
}
```
**Interpretation 7**

The behaviour of `LogicalTimeInterval.subtract` is undefined when the subtrahend is greater than the current object. We can either throw an `IllegalTimeArithmetic` exception or the `InvalidLogicalTimeInterval` exception which the C++ API throws (and which is re-introduced by Interpretation 11, below):

```java
/**
 * Returns a new <code>LogicalTimeInterval</code> whose value is <code>(this - subtrahend)</code>.  
 * @param subtrahend The <code>LogicalTimeInterval</code> to subtract from <code>this</code> 
 * @throws InvalidLogicalTimeInterval if the <code>subtrahend</code> is larger than the value 
 */
public LogicalTimeInterval subtract(LogicalTimeInterval subtrahend) throws InvalidLogicalTimeInterval
{
    if (_value < ((LogicalTimeInterval)subtrahend)._value)
        throw new InvalidLogicalTimeInterval();
    else
        return new LogicalTimeInterval{
            _value = (_value - ((LogicalTimeInterval)subtrahend)._value);
        }
}
```

**Interpretation 8**

In both the Java and Ada clauses, `LogicalTimeInterval` has "methods to set a `LogicalTimeInterval` instance to zero or epsilon and to test an instance to determine if it is zero or epsilon". Erroneously, the C++ clause omits the epsilon-related methods, while the annex omits just the `setEpsilon` method:

```cpp
virtual
void
setEpsilon() = 0;
```
Interpretation 9

Again on the LogicalTimeInterval "methods to set a LogicalTimeInterval instance to zero or epsilon", the Java annex omits these methods from the interface:

```java
/**
 * Sets the value to <code>zero</code>.
 */
public void setZero();

/**
 * Sets the value to <code>epsilon</code>.
 */
public void setEpsilon();
```

Interpretation 10

In all three clauses, the description of the LogicalTimeInterval methods establishes the analogy with the LogicalTime methods but does not explicitly mention a "method to set a LogicalTimeInterval instance to the same value as another". This should be corrected.

Interpretation 11

The aforementioned LogicalTimeInterval methods "to set a LogicalTimeInterval instance to the same value as another" appear in the Ada and C++ annexes but not in the Java annex:

```java
/**
 * Sets the value to equal the <code>other</code>’s.
 * @param other A <code>LogicalTimeInterval</code> whose value
 * is used to set <code>this</code>
 * @throws InvalidLogicalTimeInterval
 */
public void setTo(LogicalTimeInterval other)
    throws InvalidLogicalTimeInterval;
```

Note that this means InvalidLogicalTimeInterval must be added to the existing set of RTI exceptions (this exception is already thrown by the C++ API).
Interpretation 12

The Java LogicalTimeIntervalFactory is simply described as "completely analogous to its LogicalTime counterpart". However, the C++ description mentions "one more method [... which] returns a constant reference to epsilon". Annex C - LogicalTimeIntervalFactory.h contradicts this by declaring epsilon as returning an RTI::LogicalTimeInterval instance. Clearly the analogy with RTI::LogicalTimeFactory would be much more consistent. Thus RTI::LogicalTimeIntervalFactory should have two methods, makeZero and makeEpsilon. Thus (C++):

```cpp
virtual RTI::auto_ptr< RTI::LogicalTimeInterval >
makeEpsilon() = 0;
```

Interpretation 13

In 12.4.2.23 Logical time, time stamps, and lookahead, the last bullet of the list of logical time interfaces to be implemented by RTI providers incorrectly begins with: "An implementation of LogicalTimeIntervalFactory called Integer64TimeIntervalFactory". Change the sentence to: "An implementation of LogicalTimeIntervalFactory called Integer64TimeIntervalFactory".

Annex B: Java application programmer's interface

One of the strengths of the Java language is the Javadoc self-documentation convention. Besides being directly consultable, the Javadoc is typically exploited directly by most Integrated Development Environments (IDEs), and developers have to come to rely on it extensively as a productivity aid. It is therefore regrettable that the Annex B API omits most of these, particularly since the API is said to be normative. Adding the requisite comment lines is a time-consuming task, but it does have the virtue of leaving the compiled byte-code unchanged.

An API with the Javadoc comments integrated into it is supplied in one of the other annexes to this report.
Annex B: Java application programmer’s interface -
InvalidFederateHandle

This exception was introduced with the `normalizeFederateHandle` Java service. One notes, however, that it could also have been added to the list of exceptions thrown by the `registerFederationSynchronizationPoint` service when a `FederateHandleSet` is supplied. Currently, it seems that when an invalid handle is part of the supplied `FederateHandleSet` the RTI invokes `synchronizationPointRegistrationFailed` with the `SYNCHRONIZATION_SET_MEMBER_NOT_JOINED` reason (cf: DoD Interpretations, Release 2, Appendix A, Service 4.6: Register Federation Synchronization Point (Clarification 1)). The rationale behind this decision is probably the impossibility of distinguishing between an “invalid federate handle” and the “valid” federate handle of a now-unjoined federate.

Annex B: Java application programmer’s interface -
MessageRetractionReturn

The Java `MessageRetractionReturn` class is similar to the `TimeQueryReturn` class, except that it omits the `toString`, `equals` and `hashCode` methods. There is no good reason to do this. Therefore:

**Interpretation 1**

Add, in the `MessageRetractionReturn.java` file, the lines:

```java
public String toString() {
    return retractionHandleIsValid + " " + handle;
}
```
/**
   * Returns true iff <code>this</code> and <code>other</code> represent
   * the same message retraction return.
   * @param other The <code>Object</code> to compare with
   * @return true iff supplied <code>other</code> is of type
   * <code>MessageRetractionReturn</code> and has same value
   */
public boolean equals(Object other) {
    if (other instanceof MessageRetractionReturn)
    {
        MessageRetractionReturn mrrOther =
        (MessageRetractionReturn)other;
        if ((retractionHandleIsValid == false) &&
            (mrrOther.retractionHandleIsValid == false))
            return true
        else if ((retractionHandleIsValid == true) &&
            (mrrOther.retractionHandleIsValid == true))
            return handle.equals(mrrOther.handle)
        else
            return false;
    } else
        return false;
}

/**
   * Returns a hash code for <code>this</code>; two
   * <code>MessageRetractionReturn</code> s for which <code>equals()</code>
   * is <code>true</code> should yield the same hash code.
   * @return An <code>int</code> hash code
   */
public int hashCode() {
    return (retractionHandleValid ? handle.hashCode() : 7);
}

Annex B: Java application programmer's interface -
Service Groups

The Java implementation of the <code>ServiceGroup</code> class uses internally the values 4
through 10 to represent the service groups. These values are accessible through the
class's <code>toString()</code> method which will return, for example, "ServiceGroup(4)"
for the <code>FEDERATION_MANAGEMENT</code> service group. On the other hand, the
Management Object Model (MOM) clearly states (Table 11) that it uses the values 0
through 6 for the same purpose. The potential for confusion is there. The fix is
simple:

<code>Interpretation 1</code>

Replace, in the <code>ServiceGroup.java</code> file, the line:

private static final int _lowestValue = 4; //fedn mgt is chapter 4
With:

```java
private static final int _lowestValue = 0;
```

**D.8.1 [Time Management] Overview**

*Interpretation 1*

Replace the last paragraph:

Additionally, each of the language APIs also provide default concrete implementations of these two ADTs based on the float 64-datatype.

With:

Additionally, each of the language API Examples subclauses also provides a default concrete implementation of these two ADTs based on the integer-64 data type.

**IEEE 1516.2-2000: Object Model Template (OMT) Specification**

*4.4.2 [Attribute] Table Format*

Replace paragraph 14 (page 30):

The eighth column (Available dimensions) shall record the association of the class attribute with a set of dimensions if a federate or federation is using DDM services for this attribute. The column shall contain a comma-separated list of names of rows in the dimension table described in 4.6. [..]

With (emphasis added for readability):

The eighth column (Available dimensions) shall record the association of the class attribute with a set of dimensions if a federate or federation is using DDM services for this attribute. The column shall contain a space-separated list of names of rows in the dimension table described in 4.6. [..]

Table 8 (p. 31) is also incorrect in showing a comma in the Food.Drink.Soda Available dimensions column. The same error also appears on p. 32, 4.5.2 [Parameter] Table Format, paragraph 5:

The fourth column (Available dimensions) shall record the association of an interaction class with a set of dimensions if the federate or federation is using DDM services for this interaction. The column shall contain a comma-separated list of names of rows from the dimension table described in 4.6. [..]
Which becomes (emphasis added for readability):

The fourth column (Available dimensions) shall record the association of an interaction class with a set of dimensions if the federate or federation is using DDM services for this interaction. The column shall contain a space-separated list of names of rows from the dimension table described in 4.6. [..]

Annexes D and E indeed use space-separated dimension lists (see pp. 82 [Food.Drink.Soda object class] and 111 [HLAReportServiceInvocation interaction class]). Table 7 of IEEE 1516.1-2000, 11.6 MOM OMT Tables, page 224 is correct.

4.6 Dimension Table

In 1516.1 9.1.4, it was stated that “The normalization of federation data to [0, a particular dimension’s upper bound) for use with DDM services shall be left to the federation.” This is, besides Table 8 in 11.6 MOM OMT Tables, the only mention of normalization in the Federate Interface Specification. Buried here in the fourth paragraph is the sentence “The federate shall provide a normalization function for each dimension, that maps values from the federate view of a dimension to values in the RTI view of a dimension.” This requirement is undecidable, as there is no member of the FederateAmbassador interface, nor any part of the hla.rti1516 package, that mentions these normalization functions. A federate has no means whatsoever of providing the RTI (or any other federate) with the “required” normalization functions.

The only guidance supplied for their implementation is 1516.2’s Annex B, which lays out common normalization functions (so the algorithms are known), and the RTIambassador’s Normalize Federate Handle and Normalize Service Group services (so the general declaration form is known).

Is a mechanism for sharing normalization functions between federates desirable? On the one hand, having each federate’s view of a dimension be a private implementation matter is a good thing, allowing each federate to choose the internal representation that best suits its purpose and capabilities. On the other hand, if the federation semantics are such that federate views are to be shared (they could be transmitted as object attribute values or as interaction parameters, for example), then a common approach to normalization is highly desirable. Providing a normalization function sharing mechanism allows both cases to be handled, whereas the absence of such a mechanism (which is the current situation) essentially forces the federation semantics into the mould of the first case.
Normalization sharing is not an obvious extension of the RTIambassador and FederateAmbassador interfaces. Should the value(s) to be normalized be passed like interaction parameters or object attribute values? Will the HLA rules really require that a federate ask itself to normalize a value through the RTI (when it happens to be the “normalizer”)? How will federates indicate their normalizing responsibility? How will the RTI allow multiple federates to normalize for a given dimension handle (in order to allow differing views for different federate types)? How can a single method-callback pair encapsulate the various normalization schemes possible?

The RTI-provided normalization services (normalizeFederateHandle and normalizeServiceGroup) are problematic examples: they introduce two new RTIexception descendents, specific to those two services alone: InvalidFederateHandle and InvalidServiceGroup. This is not an example that federates can follow, if only because there is no way to introduce a new exception class to an existing interface.

One also notes that the federate view of a dimension is not necessarily integer like the RTI view. The linear normalization, for example, could operate from a floating-point federate view. This means the normalization API must be able to accept integer as well as floating-point domain specifications. It is less obvious whether the logarithmic and hyperbolic tangent normalization functions should exist in an integer form besides the floating-point one. In all cases, it is reasonable to assume that the value to normalize is of the same type as the domain’s upper and lower bounds.

Looking more closely at the RTIambassador’s 10.32 Set Range Bounds service, it is clear that the normalization functions are to be used in converting the federate’s view of the range bounds into values usable by the RangeBounds constructor (RTI view values). However, what is the federate to make of the values returned by the 10.31 Get Range Bounds service? They need to be converted back into the federate’s view of the dimension. Hence, unnormalization functions are also required.

Normalization and unnormalization algorithms must be defined in tandem such that, as a general rule, normalizing and then unnormalizing a given value leaves it unchanged (as long as the dimension and domain remain the same, of course). When the RTI view has grosser quantization than the federate view (e.g. the federate domain is 64-bit integer but the RTI dimension is merely 32-bit integer), or when the federate view is floating-point, some degeneracy (binning) of federate view values is expected, but no drift should occur—that is to say, the federate value should not change on a second normalization-unnormalization round-trip.
Enumerated datatypes raise a different kind of issue. The API-defined enumerated types (OrderType, ResignAction, RestoreFailureReason, RestoreStatus, SaveFailureReason, SaveStatus, ServiceGroups, SynchronizationPointFailureReason, and TransportationType) as well as the OMT-defined enumerated datatypes (HLAfederateState, HLAtimeState, HLAownership, HLAresignAction, HLAorderType, HLAswitch, HLAsyncPointStatus, HLAserviceGroupName, plus any user-defined types) are implemented as classes with prototype static field instances. In the absence of a structured hierarchy of datatype interfaces, a generic linear enumerated normalization method, using a class reference argument, must use Java's introspection methods to extract the prototype instances, and must also rely on the hash method for the RTI view values.

**B.1 Linear normalization function**

Since the RTI view is integer, either the result of the equation as written needs to be rounded or truncated, or the division operator must be replaced by an integer division and the parentheses re-arranged. The floor() operation used here is such that it returns the largest integer which is smaller than or equal to its argument. Note that the expression is the same whether the domain itself is integer or floating-point.

Function \( \text{floor}(\{\text{domain} - \text{domainLower}\} / \{\text{domainUpper} - \text{domainLower}\}) \times (\text{DUB} - 1) \)

Or:

Function \( \{\text{domain} - \text{domainLower}\} \times (\text{DUB} - 1) \) \(\text{div} \) \( \{\text{domainUpper} - \text{domainLower}\} \)

Where DUB is, as 1516.2-2000, Annex B, states, “the dimension upper bound specified for the dimension in the third column of Table 11.”

**B.1.1 Linear unnormalization function**

To properly invert the normalization function when the domain is integer, we define the ceiling() operation such that it returns the smallest integer which is larger than or equal to its argument. This expression and the corresponding normalization function work correctly when the domain range \( \{\text{domainUpper} - \text{domainLower}\} \) is larger or smaller than the dimension range \( (\text{DUB} - 1) \), and also whether the domain extends over the negative integers or not, in whole or in part. When the domain is floating-point, the ceiling() function is omitted; and the number of significant digits preserved then depends on the quantization imposed by the RTI view (for example, a domain extending from 0.0 through 1.0 would have a maximum precision of about \( 1/2^{63} \sim 10^{19} \)).

Form \( \text{unlinear}(\text{normalized}, \text{domainLower}, \text{domainUpper}) \)

Parameters: normalized: the federation’s view of the domain value
B.2 Linear enumerated normalization function

Same problem as with B.1:

Function \( \text{domainLower} + \text{ceiling}([\text{normalized} \times (\text{domainUpper} - \text{domainLower})] / (\text{DUB} - 1)) \)

Or:

Function \( \text{floor}([\text{positionInMappedSet}(\text{domain}) / (\text{mappedSetLength} - 1)] \times (\text{DUB} - 1)) \)

B.2.1 Linear enumerated unnormalization function

Here the elementFromMappedSet(\(n\)) function returns the zero-based \(n\)th element of the mapped set:

Form unlinear (normalized, mappedSet)

Parameters normalized: the federation’s view of the domain value

other parameters as per B.2

Function elementFromMappedSet(ceiling(n\(\times\)(mappedSetLength - 1) / (DUB - 1)))

B.3.1 Enumerated set unnormalization function

Whereas the normalization function’s RTI set is best described by a Java HashMap, (which uses mappedSet for its keys), unnormalization won’t be as efficient since the implementation will have to iterate over the values in order to retrieve any one key.

Function unRTIset(normalized)

where

unRTIset(normalized) is a function that maps the RTI’s view to mappedSet values

B.4 Logarithmic normalization function

Same problem as with B.1:

Function \( \text{floor}([\log(\text{domain}/\text{domainLower}) / \log(\text{domainUpper}/\text{domainLower}) \times (\text{DUB} - 1)] \)
B.4.1 Logarithmic unnormalization function

Function \( \text{ceiling}(\text{domainLower} \times \exp(\log(\text{domainUpper}/\text{domainLower}) \times \text{normalized} / (\text{DUB} - 1))) \)

B.5 Hyperbolic tangent normalization function

Same problem as with B.1:

Function \( \text{floor}(\lfloor \tanh((\text{domain} - \text{domainCenter})/\text{domainSize}) + 1/2 \rfloor) \times (\text{DUB} - 1) \)

B.5.1 Hyperbolic tangent unnormalization function

Function \( \text{ceiling}(\text{domainCentre} + \text{domainSize} \times \arctanh((2 \times \text{normalized})/(\text{DUB} - 1) - 1)) \)

C.2 HLA OMT DIF DTD declaration

Interpretation 1

Replace (fourth paragraph, first bullet):

In the tabular representation of object models, object class and interaction class inheritance are represented by columns in which the parent of a class was represented to its left and children to its right. In the XML depiction of object models, inheritance is represented by membership, in which where children of a parent class are depicted as class elements contained within the parent class element.

With:

In the tabular representation of object models, object class and interaction class inheritance are represented by columns in which the parent of a class was represented to its left and children to its right. In the XML depiction of object models, inheritance is represented by membership, in which children of a parent class are depicted as class elements contained within the parent class element.
This page intentionally left blank.
// File: AttributeHandle.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Type-safe handle for an attribute. Generally these are created
 * by the RTI and passed to the federate.
 * <p>
 * <code>AttributeHandle</code>s are obtained from the
 * (10.4) {@link RTlambassador#getAttributeHandle getAttributeHandle}
 * method and are used by a variety of other methods.
 * <p>
 * They can also be obtained by using the
 * <code>AttributeHandleFactory</code>'s {@link
 * AttributeHandleFactory#decode decode} method on a <code>byte[]</code>
 * received as part of an attribute update or interaction.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface AttributeHandle
    extends java.io.Serializable
{
    /**
     * Returns a <code>String</code> representation of the
     * <code>AttributeHandle</code>.
     * @return A {@link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String toString();

    /**
     * Returns true iff <code>this</code> and
     * <code>otherAttributeHandle</code> represent the same attribute handle.
     * @param otherAttributeHandle the <code>Object</code> to compare
     * with
     * @return <code>true</code> iff supplied
     * <code>otherAttributeHandle</code> is of type
     * <code>AttributeHandle</code> and has same value
     */
    public boolean equals(Object otherAttributeHandle);
/**
 * Returns a hash code for <code>this</code>; two
 * <code>AttributeHandle</code>s for which <code>equals()</code> is
 * <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 */
public int
hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]{}</code>
 * representation of the <code>AttributeHandle</code>.
 * @return The length (in bytes) of the <code>byte[]{}</code>
 * representation of <code>this</code>
 */
public int
encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]{}</code> at the
 * specified <code>offset</code>.
 * @param buffer the <code>byte[]{}</code> into which to encode
 * <code>this</code>
 * @param offset the offset into the <code>byte[]{}</code> at which
 * to encode <code>this</code>
 */
public void
encode(byte[] buffer,
int offset);
}
//end AttributeHandle
// File: AttributeHandleFactory.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {@link RTIambassador#getAttributeHandleFactory getAttributeHandleFactory}
 * method.
 * Its one method creates a new {@link AttributeHandle} from a supplied <code>byte[]</code> representation,
 * itself received as an attribute or parameter value.
 * @author IEEE
 * @version 1516.1.5 (DoD V2)
 */
public interface AttributeHandleFactory
    extends java.io.Serializable
{
    /**
     * Creates a new {@link AttributeHandle} from the supplied <code>byte[]</code> representation.
     * @param buffer A <code>byte[]</code> containing a representation of an <code>AttributeHandle</code>
     * @param offset Offset into the <code>buffer</code> at which the representation of the <code>AttributeHandle</code> begins
     * @return An <code>AttributeHandle</code> constructed from the buffer’s contents
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory’s target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
    public AttributeHandle decode(byte[] buffer,
                                  int offset)
        throws CouldNotDecode,
                FederateNotExecutionMember;
}
//end AttributeHandleFactory
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * A Set of [link AttributeHandle]s.
 * All [link java.util.Set] operations are required, none are optional.
 * <code>add()</code> and <code>remove()</code> should throw [link IllegalArgumentException] if
 * the argument is not an <code>AttributeHandle</code>.
 * <code>addAll()</code>, <code>removeAll()</code> and <code>retainAll()</code> should throw [link IllegalArgumentException] if
 * the argument is not an <code>AttributeHandleSet</code>.
 *<p>
 * <code>AttributeHandleSet</code>s are used by numerous [link RTIambassador] methods and
 * are provided by numerous [link FederateAmbassador] callbacks.
 * Empty ones can be created by the
 * [link AttributeHandleSetFactory]s
 * [link AttributeHandleSetFactory#create create] method.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public interface AttributeHandleSet
    extends java.util.Set,
            java.lang.Cloneable,
            java.io.Serializable
{
}

//end AttributeHandleSet
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {@link RTIambassador#getAttributeHandleSetFactory
 * getAttributeHandleSetFactory} method. 
 * Its one method creates a new {@link AttributeHandleSet},
 * to be loaded up and then passed to various {@link RTIambassador}
 * methods.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface AttributeHandleSetFactory
    extends java.io.Serializable
{

    /**
     * Creates a new {@link AttributeHandleSet}, initially empty.
     * @return An empty <code>AttributeHandleSet</code>
     */
     public AttributeHandleSet
        create();
}

//end AttributeHandleSetFactory
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * A Map linking AttributeHandle keys with the corresponding byte[] attribute values.
 * All java.util.Map operations are required, none are optional. Null mappings are not allowed.
 * put(), putAll() and remove() should throw IllegalArgumentException if
 * the key is not an AttributeHandle or the value a byte[].
 * Its main purpose is for attribute update transmission using the
 * (6.6) RTIambassador#updateAttributeValues updateAttributeValues (both forms)
 * method. The updates are received through the (6.7) FederateAmbassador#reflectAttributeValues reflectAttributeValues (all forms)
 * callback.
 * Empty AttributeHandleValueMap s can also be created by
 * AttributeHandleValueMapFactory#create method.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface AttributeHandleValueMap
    extends java.util.Map,
            java.lang.Cloneable,
            java.io.Serializable
{

//end AttributeHandleValueMap
package hla.rti1516; // the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {link RTIambassador.getAttributeHandleValueMapFactory}
 * method.
 * Its one method creates a new {link AttributeHandleValueMap},
 * to be loaded up and then passed to the (6.6) {link RTIambassador.updateAttributeValues}
 * (both forms)
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface AttributeHandleValueMapFactory
    extends java.io.Serializable
{
    /**
     * Creates a new {link AttributeHandleValueMap} with the specified initial capacity.
     * @param capacity The initial size of the AttributeHandleValueMap, in number of keys
     * @return An AttributeHandleValueMap of the specified initial capacity (in keys)
     */
    public AttributeHandleValueMap create(int capacity);

    // end AttributeHandleValueMapFactory
// File: AttributeRegionAssociation.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Record stored in {@link AttributeSetRegionSetPairList}.  
 * Each record of that list associates a set of attributes with a set 
 * of regions.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 adds this class, 
 * @author IEEE
 * @version 1516.1.5 (DoD V2)
 */
public final class AttributeRegionAssociation
    implements java.io.Serializable
{
    /**
     * The associated attribute set.
     */
    public AttributeHandleSet ahset;

    /**
     * The associated region set.
     */
    public RegionHandleSet rhset;

    /**
     * Constructs an AttributeRegionAssociation from the component 
     * attribute and region sets.
     * @param ahs The {@link AttributeHandleSet} to associate with the 
     * region set
     * @param rhs The {@link RegionHandleSet} to associate with the 
     * attribute set
     */
    public AttributeRegionAssociation(AttributeHandleSet ahs,
                                       RegionHandleSet rhs)
    {
        ahset = ahs;
        rhset = rhs;
    }
}
//end AttributeRegionAssociation
package hla.rti1516;

/**
 * This packages the attributes supplied to the RTI for various DDM services with
 * the regions to be used with the attributes.
 * It is a {@link java.util.List} of {@link AttributeRegionAssociation}s.
 * Each attribute appearing in an
 * <code>AttributeRegionAssociation.AttributeHandleSet</code> is
 * unassociated with each
 * region appearing in the corresponding
 * <code>AttributeRegionAssociation.RegionHandleSet</code>.
 * All List operations are required, none are optional.
 * <code>add()</code>, <code>addAll()</code> and <code>set()</code> should throw
 * {@link IllegalArgumentException}
 * if the argument is not an <code>AttributeRegionAssociation</code>.
 * Used by the (9.5) {@link RTIambassador#registerObjectInstanceWithRegions
 registerObjectInstanceWithRegions} (both forms),
 * (9.6/9.7) {@link RTIambassador#unassociateRegionsForUpdates} [un]
 * associateRegionsForUpdates,
 * (9.8/9.9) {@link RTIambassador#unsubscribeObjectClassAttributesWithRegions
 unsubscribeObjectClassAttributesWithRegions} [un] or
 * (9.13) {@link RTIambassador#requestAttributeValueUpdateWithRegions
 requestAttributeValueUpdateWithRegions}
 * methods.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see AttributeSetRegionSetPairListFactory
 */
public interface AttributeSetRegionSetPairList
    extends java.util.List,
            java.lang.Cloneable,
            java.io.Serializable
{ }

//end AttributeSetRegionSetPairList
package hla.rti1516;  // the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {@link RTIambassador#getAttributeSetRegionSetPairListFactory
getAttributeSetRegionSetPairListFactory} method.
 * Its one method creates a new {@link AttributeSetRegionSetPairList},
 * to be loaded up and then passed to the [9.5] {@link RTIambassador#registerObjectInstanceWithRegions
registerObjectInstanceWithRegions} (both forms),
 * [9.6/9.7] {@link RTIambassador#unassociateRegionsForUpdates [un]}
 * {@link RTIambassador#associateRegionsForUpdates associateRegionsForUpdates},
 * [9.8/9.9] {@link RTIambassador#unsubscribeObjectClassAttributesWithRegions [un]}
 * {@link RTIambassador#subscribeObjectClassAttributesWithRegions subscribeObjectClassAttributesWithRegions}
 * [Passively]} or
 * [9.13] {@link RTIambassador#requestAttributeValueUpdateWithRegions
requestAttributeValueUpdateWithRegions}
 * methods.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface AttributeSetRegionSetPairListFactory
        extends java.io.Serializable

        /**
         * Creates a new {@link AttributeSetRegionSetPairList} with the
         * specified initial capacity.
         * @param capacity The initial size of the
         * <code>AttributeSetRegionSetPairList</code>, in number of
         * <code>AttributeSetRegionSetPair</code>s
         * @return An <code>AttributeSetRegionSetPairList</code> of the
         * specified initial capacity (in
         * <code>AttributeSetRegionSetPair</code>s)
         */
        public AttributeSetRegionSetPairList
                create(int capacity);
    } // end AttributeSetRegionSetPairListFactory
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Type-safe handle for a dimension. Generally these are created by
 * the RTI and passed to the federate.
 * <p>
 * <code>DimensionHandle</code>s can be obtained from the
 * (10.12) {@link RTIambassador#getDimensionHandle getDimensionHandle}
 * and (as parts of a [link DimensionHandleSet]) the
 * (10.15) [link RTIambassador#getAllAvailableDimensions getAllAvailableDimensions}
 * getAvailableDimensionsForClassAttribute
 * getAvailableDimensionsForInteractionClass
 * and (10.30) [link RTIambassador#getAllAvailableDimensions getAllAvailableDimensions}
 * getDimensionHandleSet] methods.
 * <p>
 * They can also be obtained by using the
 * <code>DimensionHandleFactory</code>'s [link DimensionHandleFactory#decode decode] method on a <code>byte[]</code>
 * received as part of an attribute update or interaction.
 * <p>
 * [code]DimensionHandle[/code]s are used to build a [link DimensionHandleSet] which can be passed to
 * the (9.2) [link RTIambassador#createAttribute createAttribute] method.
 * They are also used by the
 * (10.13) [link RTIambassador#setDimensionName setDimensionName],
 * (10.14) [link RTIambassador#setDimensionUpperBound setDimensionUpperBound]
 * getDimensionUpperBound],
 * (10.31/10.32) [link RTIambassador#setRegion setRegion]
 * [link RTIambassador#setRangeBounds setRangeBounds] methods.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface DimensionHandle
    extends java.io.Serializable
{
    /**
     * Returns a [code]String[/code] representation of the
     * [code]DimensionHandle[/code].
     * @return A [link java.lang.String] reflecting [code]this[/code]
     * value
     */
    public String toString();
/**  
 * Returns true iff <code>this</code> and
 * <code>otherDimensionHandle</code> represent the same dimension handle.
 * @param otherDimensionHandle the <code>Object</code> to compare
 * with
 * @return <code>true</code> iff supplied
 * <code>otherDimensionHandle</code> is of type
 * <code>DimensionHandle</code> and has same value
 */
public boolean
equals(Object otherDimensionHandle);

/**
 * Returns a hash code for <code>this</code>, two
 * <code>DimensionHandle</code>s for which <code>equals()</code> is
 * <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int
hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]</code>
 * representation of the <code>DimensionHandle</code>.
 * @return The length (in bytes) of the <code>byte[]</code>
 * representation of <code>this</code>
 */
public int
encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the
 * specified <code>offset</code>.
 * @param buffer the <code>byte[]</code> into which to encode
 * <code>this</code>
 * @param offset The offset into the <code>byte[]</code> at which
 * to encode <code>this</code>
 */
public void
encode(byte[] buffer, int offset);
}
// File: DimensionHandleFactory.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {link RTIambassador#getDimensionHandleFactory getDimensionHandleFactory}
 * method.
 * Its one method creates a new {link DimensionHandle} from a supplied <code>byte[]</code> representation,
 * itself received as an attribute or parameter value.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface DimensionHandleFactory
        extends java.io.Serializable
{
    /**
     * Creates a new {link DimensionHandle} from the supplied <code>byte[]</code> representation.
     * @param buffer A <code>byte[]</code> containing a representation of a <code>DimensionHandle</code>
     * @param offset Offset into the <code>buffer</code> at which the representation of the <code>DimensionHandle</code> begins
     * @return A <code>DimensionHandle</code> constructed from the buffer’s contents
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory’s target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
    public DimensionHandle decode(byte[] buffer, int offset)
            throws CouldNotDecode, FederateNotExecutionMember;
}

//end DimensionHandleFactory
// File: DimensionHandleSet.java
package hla.rti1516;  //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
   * A {@link java.util.Set} of {@link DimensionHandle}s.
   * All Set operations are required, none are optional.
   * <code>add()</code> and <code>remove()</code> should throw {@link
   * IllegalArgumentException} if the argument is not a
   * <code>DimensionHandle</code>.
   * <code>addAll()</code>, <code>removeAll()</code> and
   * <code>retainAll()</code> should throw
   * <code>IllegalStateException</code> if
   * the argument is not a <code>DimensionHandleSet</code>.
   *
   * Used with the (9.2) {@link RTIambassador#createRegion createRegion}
   * method.
   * Returned by the
   * (10.15) {@link RTIambassador#getAvailableDimensionsForClassAttribute
   * getAvailableDimensionsForClassAttribute},
   * (10.17) {@link RTIambassador#getAvailableDimensionsForInteractionClass
   * getAvailableDimensionsForInteractionClass} and
   * (10.30) {@link RTIambassador#getDimensionHandleSet
   * getDimensionHandleSet}
   * methods.
   * @author IEEE
   * @version 1516.1.5 (DoD v2)
   * @see DimensionHandleSetFactory
   */
public interface DimensionHandleSet
    extends java.lang.Cloneable,
            java.io.Serializable,
            java.util.Set
{
}
//end DimensionHandleSet
// File: DimensionHandleSetFactory.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {@link RTIambassador#getDimensionHandleSetFactory
 getDimensionHandleSetFactory} method.
 * Its one method creates a new {@link DimensionHandleSet},
 * to be loaded up and then passed to the (9.2) {@link
 RTIambassador#createRegion createRegion} method.
 * @author IEEE
 * @version 1516.1.5 (DoD V2)
 */
public interface DimensionHandleSetFactory
    extends java.io.Serializable
{
    /**
     * Creates a new {DimensionHandleSet}, initially empty.
     * @return An empty <code>DimensionHandleSet</code>
     */
    public DimensionHandleSet create();

} //end DimensionHandleSetFactory
// File: FederateAmbassador.java
package hla.rti1516;  //the package name was changed by DoD

// Interpretations of IEEE 1516-2000v2

/**
 * Federate must implement this interface.
 * 
 * As of DoD Interpretations of IEEE 1516-2000v2, none of the
 * <code>FederateAmbassador</code> callbacks will ever occur
 * with a <code>null</code> argument, with one exception: user-
 * supplied <code>tag</code> arguments may be <code>null</code>.
 * 
 * @author IEEE
 * @version 1516.1.5 (DoD V2)
 */

public interface FederateAmbassador
{

    // 4.7

    /**
     * Notifies the federate that it has successfully registered a
     * federation synchronization point.
     * 
     * @param synchronizationPointLabel a <code>String</code> giving the
     * synchronization point's identifier
     * @throws FederateInternalError should be thrown if something goes
     * wrong
     * 
     * @see RTIambassador#registerFederationSynchronizationPoint
     * registerFederationSynchronizationPoint
     * @see RTIambassador#synchronizationPointAchieved
     * synchronizationPointAchieved
     * @see #synchronizationPointRegistrationFailed
     * synchronizationPointRegistrationFailed
     * @see #announceSynchronizationPoint announceSynchronizationPoint
     * @see #federationSynchronized federationSynchronized
     */
    public void synchronizationPointRegistrationSucceeded(String synchronizationPointLabel)
        throws FederateInternalError;
}
/**
 * Notifies the federate that it has failed to register a federation synchronization point.
 * @param synchronizationPointLabel a {@link java.lang.String} giving the synchronization point's identifier
 * @param reason a {@link SynchronizationPointFailureReason} specifying what went wrong
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#registerFederationSynchronizationPoint
 * @see RTIambassador#synchronizationPointAchieved
 * @see #synchronizationPointRegistrationSucceeded
 * @see #announceSynchronizationPoint
 * @see #federationSynchronized
 */
public void synchronizationPointRegistrationFailed(String synchronizationPointLabel, SynchronizationPointFailureReason reason)
    throws FederateInternalError;
/**
 * Notifies the federate that a synchronization point exists.
 * Achievement of the point is signalled to the RTI through the
 * {@link RTIambassador#synchronizationPointAchieved}
 * synchronizationPointAchieved} method.
 * 
 * Federates are by default part of the synchronization set, unless
 * a {@link FederateHandleSet} has been specifically
 * supplied to the @link
 * RTIambassador#registerFederationSynchronizationPoint(String,byte[],FederateHandleSet)} method.
 * A federate that resigns is simply removed from the
 * synchronization set.
 * The synchronization point exists until it has been achieved by
 * all concerned federates, which could be
 * until the federation execution concludes.
 * @param synchronizationPointLabel a {@link java.lang.String}
 * giving the synchronization point's identifier
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador#registerFederationSynchronizationPoint
 * registerFederationSynchronizationPoint
 * @see RTIambassador#synchronizationPointAchieved
 * synchronizationPointAchieved
 * @see #synchronizationPointRegistrationSucceeded
 * synchronizationPointRegistrationSucceeded
 * @see #synchronizationPointRegistrationFailed
 * synchronizationPointRegistrationFailed
 * @see #federationSynchronized federationSynchronized
 */

class
 announceSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag)
 throws FederateInternalError;

/**
 * Informs the joined federate that all members of the
 * synchronization set of the specified synchronization point
 * have invoked the {@link RTIambassador#synchronizationPointAchieved}
 * method for that point.
 * The synchronization point ceases to exist once this callback has
 * been invoked on all concerned federates.
 * @param synchronizationPointLabel a @link java.lang.String
 * giving the synchronization point's identifier
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador#registerFederationSynchronizationPoint
 * @see RTIambassador#synchronizationPointAchieved
 * @see #synchronizationPointRegistrationSucceeded
 * @see #synchronizationPointRegistrationFailed
 * @see #announceSynchronizationPoint
 */

public void federationSynchronized(String synchronizationPointLabel)
  throws FederateInternalError;
/**
 * Instructs the joined federate that it is now in the Instructed To Save state
 * and should therefore save state as soon as possible.
 * The joined federate should use the supplied <code>label</code>,
 * the name of the federation execution (see the
 * <code>federationExecutionName</code> of the {@link RTIambassador#joinFederationExecution
 * joinFederationExecution} invocation),
 * its joined federate designator (returned by the aforementioned invocation)
 * and
 * its federate type (the aforementioned invocation's
 * <code>federateType</code> argument)
 * to distinguish the saved state information.
 * A federate that is not time constrained should expect this callback at any point.
 * A time constrained federate can receive this callback only whilst in the Time Advancing state.
 * Once in the Instructed To Save state, the federate is severely limited in which RTIambassador
 * methods it can invoke. This lasts through the Saving and Waiting For Federation To Save states,
 * concluding with the {@link #federationSaved federationSaved} or
 * {@link #federationNotSaved federationNotSaved} callbacks.
 * @param label a {@link java.lang.String} holding the saved state's identifier
 * @throws UnableToPerformSave should be thrown if the save operation seems doomed
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void initiateFederateSave(  
    String label  
) throws UnableToPerformSave,  
    FederateInternalError;
/**
 * Instructs the joined federate that it is now in the Instructed
 * To Save state (as of the specified <code>time</code>) and should therefore
 * save state as soon as possible.
 * <p>
 * For details, see the {@link #initiateFederateSave(String)}
callback.
 * @param label a {@link java.lang.String} holding the saved
 * state's identifier
 * @param time a {@link LogicalTime} specifying when the save was
 * scheduled
 * @throws InvalidLogicalTime should be thrown if the specified
 * <code>LogicalTime</code> is invalid
 * @throws UnableToPerformSave should be thrown if the save
 * operation seems doomed
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus
 * queryFederationSaveStatus
 * @see #federationSaved federationSaved
 * @see #federationNotSaved federationNotSaved
 * @see #federationSaveStatusResponse federationSaveStatusResponse
 */
public void
initiateFederateSave(
String label,
LogicalTime time)
throws InvalidLogicalTime,
UnableToPerformSave,
FederateInternalError;
/**
 * Informs the joined federate that the federation save process is complete and successfully.
 * <p>
 * All joined federates at which the [@link #initiateFederateSave initiateFederateSave} callback
 * was invoked have in turn invoked the [@link RTIambassador#federateSaveComplete federateSaveComplete} method.
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #initiateFederateSave initiateFederateSave
 * @see #federationNotSaved federationNotSaved
 * @see #federationSaveStatusResponse federationSaveStatusResponse
 */

public void federationSaved()
throws FederateInternalError;

/**
 * Informs the joined federate that the federation save process has completed in failure.
 * The possible save failure reasons are:
 * <ul>
 * <li>RTI_UNABLE_TO_SAVE: The RTI was unable to save
 * <li>FEDERATE_REPORTED_FAILURE: One or more joined federates have invoked the RTIambassador#federateSaveNotComplete method
 * <li>FEDERATE_RESIGNED: One or more joined federates have resigned from the federation execution
 * <li>RTI_DETECTED_FAILURE: The RTI has detected failure at one or more of the joined federates
 * <li>SAVE_TIME_CANNOT_BE_HONORED: The time stamp specified by the RTIambassador#requestFederationSave(String,LogicalTime) request cannot be honoured, due to possible race conditions in the distributed calculation of GALT [Greatest Available Logical Time]
 * </ul>
 * @param reason a SaveFailureReason specifying why the save failed
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationSave
 * @see RTIambassador#federateSaveBegun
 * @see RTIambassador#federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus
 * @see #initiateFederateSave
 * @see #federationSaved
 * @see #federationSaveStatusResponse
 */
public void federationNotSaved(
    SaveFailureReason reason)
    throws FederateInternalError;

/**
 * Supplies the federate with the previously requested save status indicators.
 * @param response a `FederateHandleSaveStatusPair[]` specifying the `SaveStatus` of each federate
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #initiateFederateSave initiateFederateSave
 * @see #federationSaved federationSaved
 * @see #federationNotSaved federationNotSaved
 */
 public void federationSaveStatusResponse(
   FederateHandleSaveStatusPair[] response)
 throws FederateInternalError;
/**
 * Indicates that the federate's previous `{@link RTIambassador#requestFederationRestore requestFederationRestore}` has been granted.
 * <p>
 * This means the RTI has located the RTI-specific saved state information matching the previously supplied <code>label</code>, [{@link RTIambassador#joinFederationExecution federationExecutionName} and
 * [{@link RTIambassador#createFederationExecution fdd}], and that the census of currently joined federates matches in number and [{@link RTIambassador#joinFederationExecution federateType}] that of the RTI's saved state.<p>
 * @param label a [{@link java.lang.String}] holding the saved state's identifier
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore requestFederationRestore
 * @see RTIambassador#federateRestoreComplete federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus
 * @see #requestFederationRestoreFailed requestFederationRestoreFailed
 * @see federationRestoreBegun federationRestoreBegun
 * @see #initiateFederateRestore initiateFederateRestore
 * @see #federationRestored federationRestored
 * @see #federationNotRestored federationNotRestored
 * @see federationRestoreStatusResponse
 */

public void requestFederationRestoreSucceeded(String label)
  throws FederateInternalError;

/**
 * Indicates that the federate's previous [RTIambassador#requestFederationRestore requestFederationRestore] has been denied.
 * This means the RTI failed to locate its specific saved state information or that the census of currently joined federates does not match in number and [RTIambassador#joinFederationExecution federateType] that of the retrieved RTI saved state.
 * Failures by individual federates to complete the restoration process lead to the [federationNotRestored federationNotRestored] callback instead.
 * @param label a [java.lang.String] holding the saved state's identifier
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore requestFederationRestore
 * @see RTIambassador#federateRestoreComplete federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus
 * @see federationRestoreSucceeded
 * @see federationRestoreBegin federationRestoreBegin
 * @see initiateFederateRestore initiateFederateRestore
 * @see federationRestored federationRestored
 * @see federationNotRestored federationNotRestored
 * @see federationRestoreStatusResponse federationRestoreStatusResponse
 */
public void requestFederationRestoreFailed(String label) throws FederateInternalError;
Instructs the joined federate that it is now in the Prepared To Restore state and should prepare to proceed with saved state restoration. The necessary information is later provided by the \{@link #initiateFederateRestore initiateFederateRestore\} callback. Once in the Prepared To Restore state, the federate is severely limited in which RTIambassador methods it can invoke. This lasts through the Restoring and Waiting For Federation To Restore states, concluding with the \{@link #federationRestored federationRestored\} or \{@link #federationNotRestored federationNotRestored\} callbacks. @throws FederateInternalError should be thrown if something goes wrong. @see RTIambassador#requestFederationRestore requestFederationRestore @see RTIambassador#federateRestoreComplete federateRestoreComplete @see RTIambassador#federateRestoreNotComplete federateRestoreNotComplete @see RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus @see #requestFederationRestoreSucceeded requestFederationRestoreSucceeded @see #requestFederationRestoreFailed requestFederationRestoreFailed @see #initiateFederateRestore initiateFederateRestore @see #federationRestored federationRestored @see #federationNotRestored federationNotRestored @see #federationRestoreStatusResponse federationRestoreStatusResponse

public void federationRestoreBegun()
throws FederateInternalError;
Instructs the joined federate to return to a previously saved state.

- The joined federate should use the supplied `<code>label</code>`, the name of the federation execution (see the `<code>federationExecutionName</code>` of the `RTIambassador#joinFederationExecution` invocation),
- the supplied federate designator `<code>federateHandle</code>` and its federate type (the aforementioned invocation's `<code>federateType</code>` argument)
- to retrieve the saved state information.

Note that the supplied `<code>federateHandle</code>` may differ from the federate's current `<code>FederateHandle</code>`;
- it will assume this new designator if and once it receives the `<code>federationRestored</code>` callback.

- @param label a `<link java.lang.String>` holding the saved state's identifier
- @param federateHandle the `<link FederateHandle>` that the federate will assume if and once it receives the `<link federationRestored>` callback

- @throws SpecifiedSaveLabelDoesNotExist should be thrown if the label isn't recognized
- @throws CouldNotInitiateRestore should be thrown if the federate is unwilling or unable to initiate the restore operation
- @throws FederateInternalError should be thrown if something else goes wrong

- @see `RTIambassador#requestFederationRestore`
- @see `RTIambassador#federateRestoreComplete`
- @see `RTIambassador#federateRestoreNotComplete`
- @see `RTIambassador#queryFederationRestoreStatus`
- @see `#requestFederationRestoreSucceeded`
- @see `#requestFederationRestoreFailed`
- @see `#federationRestoreBegun`
- @see `#federationRestored`
- @see `#federationNotRestored`
- @see `#federationRestoreStatusResponse`
/**
 * Informs the joined federate that the federation restore process is complete and successful.
 * This means that all joined federates which received the [RTIambassador#federateRestoreBegun]
 * callback have invoked the [RTIambassador#federateRestoreComplete] method.
 * The federate's [FederateHandle] is now the one that was supplied by the [RTIambassador#initiateFederateRestore]
 * callback.
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore
 * @see RTIambassador#federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus
 * @see #requestFederationRestoreSucceeded
 * @see #requestFederationRestoreFailed
 * @see RTIambassador#federationRestoreBegun
 * @see RTIambassador#initiateFederateRestore
 * @see #federationNotRestored
 * @see #federationRestoreStatusResponse
 */
public void federationRestored()
throws FederateInternalError;
/**
* Informs the joined federate that the federation restore process has completed in failure.
* <p>
* The possible save failure reasons are:
* <ul>
* <li> RTI_UNABLE_TO_RESTORE: The RTI was unable to restore
* <li> FEDERATE_REPORTED_FAILURE: One or more federates have invoked the [link RTIambassador#federateRestoreNotComplete federateRestoreNotComplete] method
* <li> FEDERATE_RESIGNED: One or more joined federates have resigned from the federation execution
* <li> RTI_DETECTED_FAILURE: The RTI has detected failure at one or more of the joined federates
* </ul>
* @param reason a [link RestoreFailureReason] specifying the reason for the failure
* @throws FederateInternalError should be thrown if something goes wrong
* @see RTIambassador#requestFederationRestore requestFederationRestore
* @see RTIambassador#federateRestoreComplete federateRestoreComplete
* @see RTIambassador#federateRestoreNotComplete federateRestoreNotComplete
* @see RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus
* @see #requestFederationRestoreSucceeded requestFederationRestoreSucceeded
* @see #requestFederationRestoreFailed requestFederationRestoreFailed
* @see #federationRestoreBegun federationRestoreBegun
* @see #initiateFederateRestore initiateFederateRestore
* @see #federationRestored federationRestored
* @see #federationRestoreStatusResponse federationRestoreStatusResponse
*/
public void federationNotRestored(RestoreFailureReason reason)
  throws FederateInternalError;
/**
 * Supplies the federate with the previously requested restore status indicators.
 * The \{@link FederateHandle\}s used in the \{@link
 * FederateHandleRestoreStatusPair\}[] are the pre-restore ones
 * until all federates have invoked the \{@link
 * RTIambassador\#federateRestoreComplete federateRestoreComplete\}
 * service. Once all federates have been issued the \{@link
 * federationRestored\} callbacks, the post-restore
 * \{@link FederateHandle\}s are used (and each federate's
 * status will be \{@link RestoreStatus NO_RESTORE_IN_PROGRESS\}).
 * If the \{@link RTIambassador\#queryFederationRestoreStatus
 * queryFederationRestoreStatus\} service is
 * invoked between those two times, the \{@link RestoreStatus\} and
 * \{@link FederateHandle\}s are unpredictable.
 * A federate should therefore avoid invoking the
 * \{@link queryFederationRestoreStatus\} service between the moment
 * it invokes the \{@link federateRestoreComplete\} service and
 * the moment it receives the \{@link federationRestored\} or
 * \{@link federationNotRestored\} callbacks.
 * @param response a \{@link FederateHandleRestoreStatusPair\}[]
 * specifying the \{@link RestoreStatus\} of each federate
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador\#requestFederationRestore
 * requestFederationRestore
 * @see RTIambassador\#federateRestoreComplete
 * federateRestoreComplete
 * @see RTIambassador\#federateRestoreNotComplete
 * federateRestoreNotComplete
 * @see RTIambassador\#queryFederationRestoreStatus
 * queryFederationRestoreStatus
 * @see #requestFederationRestoreSucceeded
 * requestFederationRestoreSucceeded
 * @see #requestFederationRestoreFailed
 * requestFederationRestoreFailed
 * @see #federationRestoreBegun federationRestoreBegun
 * @see #initiateFederateRestore initiateFederateRestore
 * @see #federationRestored federationRestored
 * @see #federationNotRestored federationNotRestored
 */
public void federationRestoreStatusResponse(
    FederateHandleRestoreStatusPair[] response)
throws FederateInternalError;
* Notifies the federate that registration of new object instances of the specified object class is advised because at least one of the federate-published class attributes is actively subscribed to by at least one other federate at that object class. This occurs only if the federate's Object Class Relevance Advisory Switch is turned on.

* @param theClass the ObjectClassHandle of the subject object class

* @throws ObjectClassNotPublished should be thrown if the federate denies publishing theClass

* @throws FederateInternalError should be thrown if something else goes wrong

* @see #stopRegistrationForObjectClass

* @see RTIambassador#enableObjectClassRelevanceAdvisorySwitch

* @see RTIambassador#disableObjectClassRelevanceAdvisorySwitch

*/

class void startRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished,
    FederateInternalError;
// 5.11
/**
 * Notifies the federate that registration of new object instances
 * of the specified object class
 * is not advised because there are no active subscribers to any of
 * the federate-published class attributes
 * at that object class.
 * This occurs only if the federate's Object Class Relevance
 * Advisory Switch is turned on.
 * @param theClass the [ObjectClassHandle] of the subject object class
 * @throws ObjectClassNotPublished should be thrown if the federate
denies publishing <code>theClass</code>
 * @throws FederateInternalError should be thrown if something else
goes wrong
 * @see startRegistrationForObjectClass
 * @see RTIambassador#enableObjectClassRelevanceAdvisorySwitch
 * @see RTIambassador#disableObjectClassRelevanceAdvisorySwitch
 */
public void stopRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished,
    FederateInternalError;

// 5.12
/**
 * Notifies the federate that the specified class of interactions
 * is relevant because
 * there is at least one active subscription by another federate.
 * This occurs only if the federate's Interaction Relevance
 * Advisory Switch is turned on.
 * @param theHandle the [InteractionClassHandle] of the subject interaction class
 * @throws InteractionClassNotPublished should be thrown if the federate
denies publishing <code>theHandle</code>
 * @throws FederateInternalError should be thrown if something else
goes wrong
 * @see turnInteractionsOff
 * @see RTIambassador#enableInteractionRelevanceAdvisorySwitch
 * @see RTIambassador#disableInteractionRelevanceAdvisorySwitch
 */
public void turnInteractionsOn(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError;
/**
 * Notifies the federate that the specified class of interactions is not relevant because there are no active subscriptions by other federates. This occurs only if the federate’s Interaction Relevance Advisory Switch is turned on.
 * @param theHandle the InteractionClassHandle of the subject interaction class
 * @throws InteractionClassNotPublished should be thrown if the federate denies publishing <code>theHandle</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void turnInteractionsOff(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
        FederateInternalError;

// 6.3
/**
 * Notifies the federate that the <code>objectName</code> provided in a previous invocation of the <code>reserveObjectInstanceName</code> method has been reserved.
 * @param objectName a String holding the requested object name
 * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void objectInstanceNameReservationSucceeded(
    String objectName)
throws UnknownName,
        FederateInternalError;
/**
 * Notifies the federate that the <code>objectName</code> provided in a previous invocation of the
 * {@link RTIambassador#reserveObjectInstanceName} method could not be reserved.
 * @param objectName a {@link java.lang.String} holding the requested object name
 * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#reserveObjectInstanceName
 * @see #objectInstanceNameReservationSucceeded
 */
public void objectInstanceNameReservationFailed(String objectName)
throws UnknownName, FederateInternalError;

// 6.5
/**
 * Notifies the federate that it has discovered an object instance.
 * @param theObject the {@link ObjectInstanceHandle} of the newly discovered object instance
 * @param theObjectClass the {@link ObjectClassHandle} of the class the instance was discovered as
 * @param objectName a {@link java.lang.String} holding the newly discovered object instance's name
 * @throws CouldNotDiscover should be thrown if the object instance could not be discovered for some reason other than an unrecognized object class
 * @throws ObjectClassNotRecognized should be thrown if the federate does not recognize <code>theObjectClass</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#registerObjectInstance registerObjectInstance
 */
public void discoverObjectInstance(
    ObjectInstanceHandle theObject,
    ObjectClassHandle theObjectClass,
    String objectName)
throws CouldNotDiscover,
    ObjectClassNotRecognized,
    FederateInternalError;
** Provides the federate with new values for the specified instance attributes.
  * This callback, coupled with the [@link RTIambassador#updateAttributeValues updateAttributeValues] method,
  * forms the primary data exchange mechanism supported by the RTI.
  * This form is invoked by the RTI only if the sent order type was RECEIVE,
  * no time-stamp was provided
  * and an update region set was not used by the sender (or is not pertinent or is
  * being filtered out by the receiver).
  * <p>
  * Note that the
  * absence of a [@link MessageRetractionHandle] implies the
  <code>sentOrdering</code> is RECEIVE
  * (and that therefore the received ordering is also RECEIVE).
  * <p>
  * Note that the federate is not expected to throw the [@link InvalidOrderType] or
  * [@link InvalidTransportationType] exceptions; in other words the
  RTI
  * guarantees the validity of the supplied
  <code>sentOrdering</code> and <code>theTransport</code>.<p>
  * <p>
  * The time stamp and receive message order type arguments are supplied together or not at all,
  * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(ObjectInstanceHandle theObject, AttributeHandleValueMap theAttributes, byte[] userSuppliedTag, OrderType sentOrdering, TransportationType theTransport) 
throws ObjectInstanceNotKnown, AttributeNotRecognized, AttributeNotSubscribed, FederateInternalError;
/**
 * Provides the federate with new values for the specified instance attributes
 * and specifies the update regions used.
 * This callback, coupled with the [@link RTIambassador#updateAttributeValues updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was RECEIVE,
 * no time-stamp was provided,
 * the instance attributes have available dimensions, the federate's Convey Region
 * Designator Sets Switch is enabled and an update region set was used by the sender.
 * <p>
 * Note that the absence of a [@link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [@link InvalidOrderType],
 * [@link InvalidTransportationType], [@link InvalidRegion] or [@link InvalidRegionContext]
 * exceptions (nor a notional new <code>InvalidRegionSet</code> exception); in other words the RTI
 * guarantees the validity and pertinence of the supplied
 * <code>sentOrdering</code>,
 * <code>theTransport</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            FederatelnternalError;
/**
 * Provides the federate with new values for the specified instance attributes
 * and specifies the time-stamp at which this comes into effect.
 * This callback, coupled with the [link RTIambassador#updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was RECEIVE,
 * a time-stamp was provided
 * and an update region set was not used by the sender (or is not pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the absence of a [link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the <code>receivedOrdering</code> is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [link InvalidOrderType],
 * [link InvalidTransportationType] or [link InvalidLogicalTime]
 * exceptions; in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>, <code>theTransport</code>,
 * <code>theTime</code>
 * and <code>receivedOrdering</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
    throws ObjectInstanceNotKnown,
              AttributeNotRecognized,
              AttributeNotSubscribed,
              FederateInternalError;

/**
 * Provides the federate with new values for the specified instance attributes,
 * specifies the update regions used
 * and specifies the time-stamp at which this comes into effect.
 * This callback, coupled with the [@link RTIambassador#updateAttributeValues updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * a time-stamp was provided,
 * the instance attributes have available dimensions, the
 * federate's
 * Convey Region Designator Sets Switch is enabled and an update
 * region set
 * was used by the sender.
 * <p>
 * Note that the
 * absence of a [@link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the <code>receivedOrdering</code> is also
 * RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [@link InvalidOrderType],
 * [@link InvalidTransportationType], [@link InvalidLogicalTime],
 * [@link InvalidRegion] or
 * [@link InvalidRegionContext] exceptions (nor a notional new
 * <code>InvalidRegionSet</code> exception); in other words the RTI guarantees the validity of
 * the supplied
 * <code>sentOrdering</code>, <code>theTransport</code>,
 * <code>theTime</code>,
 * <code>receivedOrdering</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
@param theObject the \{@link ObjectInstanceHandle\} of the concerned object instance
@param theAttributes an \{@link AttributeHandleValueMap\} specifying the new attribute values
@param userSuppliedTag a \code{byte[]} tag (this parameter may be \code{null})
@param sentOrdering the \{@link OrderType\} the passel was sent as
@param theTransport the \{@link TransportationType\} used to send the passel
@param theTime the \{@link LogicalTime\} at which the update comes into effect
@param receivedOrdering the \{@link OrderType\} the passel was received as
@param sentRegions the \{@link RegionHandleSet\} used to send the update
@throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
@throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
@throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
@throws FederateInternalError should be thrown if something else goes wrong
@see RTIambassador\#updateAttributeValues (ObjectInstanceHandle, AttributeHandleValueMap, byte[], LogicalTime)
@see RTIambassador\#associateRegionsForUpdates associateRegionsForUpdates

/**
 * public void
 * reflectAttributeValues(
 * ObjectInstanceHandle theObject,
 * AttributeHandleValueMap theAttributes,
 * byte[] userSuppliedTag,
 * OrderType sentOrdering,
 * TransportationType theTransport,
 * LogicalTime theTime,
 * OrderType receivedOrdering,
 * RegionHandleSet sentRegions)
 * throws ObjectInstanceNotKnown,
 * AttributeNotRecognized,
 * AttributeNotSubscribed,
 * FederateInternalError;
/**
 * Provides the federate with new values for the specified instance attributes
 * and specifies the time-stamp at which this comes into effect as well as a retraction handle.
 * This callback, coupled with the [@link RTIambassador#updateAttributeValues updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was TIMESTAMP
 * (thus a time-stamp was provided)
 * and an update region set was not used by the sender (or is not pertinent or is not being filtered out by the receiver).
 * <p>
 * Note that the presence of a [@link MessageRetractionHandle] implies the
 * @code sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the [@link
 * InvalidOrderType],
 * [@link InvalidTransportationType] or [@link
 * InvalidMessageRetractionHandle] exceptions;
 * in other words the RTI guarantees the validity of the supplied
 * @code sentOrdering</code>, @code theTransport</code>,
 * @code receivedOrdering</code> and
 * @code retractionHandle</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
 */
* @param theObject the `{@link ObjectInstanceHandle}` of the concerned object instance
* @param theAttributes an `{@link AttributeHandleValueMap}` specifying the new attribute values
* @param userSuppliedTag a `<code>byte[]</code>` tag (this parameter may be `null`)
* @param sentOrdering the `{@link OrderType}` the passel was sent as
* @param theTransport the `{@link TransportationType}` used to send the passel
* @param theTime the `{@link LogicalTime}` at which the update comes into effect
* @param receivedOrdering the `{@link OrderType}` the passel was received as
* @param retractionHandle the `{@link MessageRetractionHandle}` of the message
* @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
* @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
* @throws InvalidLogicalTime should be thrown if the specified `<code>LogicalTime</code>` is invalid (only if the `<code>receivedOrdering</code>` is `TIMESTAMP`)
* @throws FederateInternalError should be thrown if something else goes wrong
* @see `RTIambassador.updateAttributeValues(ObjectInstanceHandle, AttributeHandleValueMap, byte[], LogicalTime)`

```java
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;
```
/**
 * Provides the federate with new values for the specified instance attributes,
 * specifies the update regions used
 * and specifies the time-stamp at which this comes into effect as well as a retraction handle.
 * This callback, coupled with the [RTIambassador#updateAttributeValues updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was TIMESTAMP
 * (thus a time-stamp was provided),
 * the instance attributes have available dimensions, the federate's Convey Region Designator Sets Switch
 * is enabled and an update region set was used by the sender.
 * <p>
 * Note that the presence of a [MessageRetractionHandle] implies the <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the [InvalidOrderType],
 * [InvalidTransportationType], [InvalidMessageRetractionHandle],
 * [InvalidRegion] or [InvalidRegionContext] exceptions (nor a notional new <code>InvalidRegionSet</code> exception); in other words the RTI guarantees the validity
 * of the supplied <code>sentOrdering</code>, <code>transport</code>, <code>receivedOrdering</code>,
 * <code>retractionHandle</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;
// 6.9
/**
 * Provides the federate with a sent interaction.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * no time-stamp was provided
 * and an update region set was not used by the sender (or is not
 * pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the
 * absence of a [@link MessageRetractionHandle] implies the
 <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [@link
 InvalidOrderType] or
 * [@link InvalidTransportationType] exceptions; in other words the
 RTI
 * guarantees the validity of the supplied
 <code>sentOrdering</code> and <code>theTransport</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 supplied together or not at all,
 * which explains the absence of some other possible forms of this
 callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError;
/**
 * Provides the federate with a sent interaction and
 * specifies the broadcasting regions used.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * no time-stamp was provided,
 * the parameters have available dimensions, the federate's Convey
 * Region
 * Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the
 * absence of a [@link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [@link
 * InvalidOrderType],
 * [@link InvalidTransportationType], [@link InvalidRegion] or
 * [@link InvalidRegionContext] exceptions
 * (nor a notional new <code>InvalidRegionSet</code> exception); in
 * other words the RTI guarantees
 * the validity of the supplied <code>sentOrdering</code>,
 * <code>theTransport</code>
 * and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
@param interactionClass the {@link InteractionClassHandle} of the received interaction
@param theParameters a {@link ParameterHandleValueMap} specifying the interaction parameter values
@param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
@param sentOrdering the {@link OrderType} the interaction was sent as
@param theTransport the {@link TransportationType} used to send the interaction
@param sentRegions the {@link RegionHandleSet} used to send the interaction
@throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
@throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
@throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
@throws FederateInternalError should be thrown if something else goes wrong
@see RTIambassador#sendInteractionWithRegions(InteractionClassHandle,ParameterHandleValueMap,RegionHandleSet,byte[])

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError;
/**
 * Provides the federate with a sent interaction
 * and specifies the time-stamp at which this occurs.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * a time-stamp was provided
 * and an update region set was not used by the sender (or is not
 * pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the
 * absence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the {@link
 * InvalidOrderType} or
 * {@link InvalidTransportationType} exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * <code>sentOrdering</code> and <code>theTransport</code>.
 * Note that the federate is not expected to throw the {<br />
 * InvalidOrderType},
 * {@link InvalidTransportationType} or {@link InvalidLogicalTime}<br />
 * exceptions; in other words the<br />
 * RTI guarantees the validity of the supplied<br />
 * <code>sentOrdering</code>, <code>theTransport</code>,
 * <code>receivedOrdering</code> and <code>theTime</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering
) throws InteractionClassNotRecognized,
        InteractionParameterNotRecognized,
        InteractionClassNotSubscribed,
        FederateInternalError;
/**
 * Provides the federate with a sent interaction,
 * specifies the broadcasting regions used
 * and specifies the time-stamp at which this occurs.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * a time-stamp was provided,
 * the parameters have available dimensions, the federate's Convey
 * Region
 * Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the
 * absence of a @{link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the @{link
 * InvalidOrderType} or
 * @{link InvalidTransportationType} exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * <code>sentOrdering</code> and <code>theTransport</code>.
 * Note that the federate is not expected to throw the @{link
 * InvalidOrderType},
 * @{link InvalidTransportationType}, @{link InvalidLogicalTime},
 * @{link InvalidRegion} or
 * @{link InvalidRegionContext} exceptions (nor a notional new
 * <code>InvalidRegionSet</code> exception);
 * in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>,
 * <code>theTransport</code>, <code>receivedOrdering</code>,
 * <code>theTime</code>,
 * and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
@param interactionClass the {@link InteractionClassHandle} of the received interaction
@param theParameters a {@link ParameterHandleValueMap} specifying the interaction parameter values
@param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
@param sentOrdering the {@link OrderType} the interaction was sent as
@param theTransport the {@link TransportationType} used to send the interaction
@param theTime the {@link LogicalTime} at which the interaction occurs
@param receivedOrdering the {@link OrderType} the passel was received as
@param sentRegions the {@link RegionHandleSet} used to send the interaction

@throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
@throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
@throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
@throws FederateInternalError should be thrown if something else goes wrong

@see RTIambassador#sendInteractionWithRegions(InteractionClassHandle,ParameterHandleValueMap,RegionHandleSet,byte[],LogicalTime)

public void receiveInteraction(InteractionClassHandle interactionClass, ParameterHandleValueMap theParameters, byte[] userSuppliedTag, OrderType sentOrdering, TransportationType theTransport, LogicalTime theTime, OrderType receivedOrdering, RegionHandleSet sentRegions) throws InteractionClassNotRecognized, InteractionParameterNotRecognized, InteractionClassNotSubscribed, FederateInternalError;
/**
 * Provides the federate with a sent interaction
 * and specifies the time-stamp at which this occurs as well as a
 * retraction handle.
 * This form is invoked by the RTI only if the sent order type was
 * TIMESTAMP
 * (thus a time-stamp was provided)
 * and an update region set was not used by the sender (or is not
 * pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the
 * presence of a [link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the [link
 * InvalidOrderType] or
 * [link InvalidTransportationType] exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * <code>sentOrdering</code> and <code>theTransport</code>.
 * <p>
 * Note that the federate is not expected to throw the [link
 * InvalidOrderType],
 * [link InvalidTransportationType] or [link
 * InvalidMessageRetractionHandle] exceptions;
 * in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>,
 * <code>theTransport</code>, <code>receivedOrdering</code> and
 * <code>messageRetractionHandle</code>.
 * <p>
 * The time-stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
    throws InteractionClassNotRecognized,
               InteractionParameterNotRecognized,
               InteractionClassNotSubscribed,
               InvalidLogicalTime,
               FederateInternalError;
/**
 * Provides the federate with a sent interaction,
 * specifies the broadcasting regions used
 * and specifies the time-stamp at which this occurs as well as a
 * retraction handle.
 * This form is invoked by the RTI only if the sent order type was
 * TIMESTAMP
 * (thus a time-stamp was provided),
 * the parameters have available dimensions, the federate's Convey
 * Region
 * Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the
 * presence of a [MessageRetractionHandle] implies the
 * sentOrdering is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the [InvalidOrderType]
 * or
 * [InvalidTransportationType] exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * sentOrderings and theTransport.
 * <p>
 * Note that the federate is not expected to throw the [InvalidOrderType],
 * [InvalidTransportationType], [InvalidMessageRetractionHandle],
 * [InvalidRegion] or [InvalidRegionContext] exceptions
 * (nor a notional
 * new [InvalidRegionSet] exception); in other words the
 * RTI guarantees the validity
 * of the supplied sentOrdering, theTransport, receivedOrdering,
 * messageRetractionHandle and
 * sentRegions.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
@param interactionClass the [@link InteractionClassHandle] of the received interaction
@param theParameters a [@link ParameterHandleValueMap] specifying the interaction parameter values
@param userSuppliedTag a `<code>byte[]</code>` tag (this parameter may be `<code>null</code>`)
@param sentOrdering the [@link OrderType] the interaction was sent as
@param theTransport the [@link TransportationType] used to send the interaction
@param theTime the [@link LogicalTime] at which the interaction occurs
@param receivedOrdering the [@link OrderType] the passel was received as
@param messageRetractionHandle the [@link MessageRetractionHandle] of the message
@param sentRegions the [@link RegionHandleSet] used to send the interaction
@throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
@throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn’t recognized in the supplied context
@throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
@throws InvalidLogicalTime should be thrown if the specified `<code>LogicalTime</code>` is invalid (only if the `<code>receivedOrdering</code>` is `TIMESTAMP`)
@throws FederateInternalError should be thrown if something else goes wrong
@see RTIambassador#sendInteractionWithRegions sendInteractionWithRegions

```java
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
        throws InteractionClassNotRecognized,
               InteractionParameterNotRecognized,
               InteractionClassNotSubscribed,
               InvalidLogicalTime,
               FederateInternalError;
```
// 6.11
/**
 * Notifies the federate that an object instance has been deleted from the federation execution.
 * This form is invoked by the RTI only if the sent order type was RECEIVE
 * and no time-stamp was provided.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param userSuppliedTag a byte[].tag (this parameter may be null)
 * @param sentOrdering the OrderType the message was sent as
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#deleteObjectInstance deleteObjectInstance
 */
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering)
throws ObjectInstanceNotKnown,
    FederateInternalError;
/**
 * Notifies the federate that an object instance has been deleted from the federation execution at the specified time stamp.
 * This form is invoked by the RTI only if the sent order type was RECEIVE and a time-stamp was provided.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all, which explains the absence of some other possible forms of this callback.
 * @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the {@link OrderType} the message was sent as
 * @param theTime the {@link LogicalTime} at which the deletion occurs
 * @param receivedOrdering the {@link OrderType} the message was received as
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#deleteObjectInstance deleteObjectInstance
 */
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
    FederateInternalError;
/**
   * Notifies the federate that an object instance has been deleted
   * from the federation execution
   * at the specified time stamp and specifies a retraction handle.
   * This form is invoked by the RTI only if the sent order type was
   * TIMESTAMP
   * (and thus a time-stamp was provided).
   * <p>
   * The time stamp and receive message order type arguments are
   * supplied together or not at all,
   * which explains the absence of some other possible forms of this
   * callback.
   * @param theObject the {@link ObjectInstanceHandle} of the
   * concerned object instance
   * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
   * may be <code>null</code>)
   * @param sentOrdering the {@link OrderType} the message was sent
   * as
   * @param theTime the {@link LogicalTime} at which the deletion
   * occurs
   * @param receivedOrdering the {@link OrderType} the message was
   * received as
   * @param retractionHandle the {@link MessageRetractionHandle} of
   * the message
   * @throws ObjectInstanceNotKnown should be thrown if the federate
   * denies having previously discovered the object instance
   * @throws InvalidLogicalTime should be thrown if the specified
   * <code>LogicalTime</code> is invalid (only if the
   * <code>receivedOrdering</code> is TIMESTAMP)
   * @throws FederateInternalError should be thrown if something else
   * goes wrong
   * @see RTIambassador#deleteObjectInstance deleteObjectInstance
   */
   public void
   removeObjectInstance(
         ObjectInstanceHandle theObject,
         byte[] userSuppliedTag,
         OrderType sentOrdering,
         LogicalTime theTime,
         OrderType receivedOrdering,
         MessageRetractionHandle retractionHandle)
         throws ObjectInstanceNotKnown,
                   InvalidLogicalTime,
                   FederateInternalError;
/**
 * Notifies the federate that the specified attributes for the object instance are in its scope.
 * This occurs only if the Attribute Scope Advisory Switch is on for the federate.
 * @param theObject the {ObjectInstanceHandle} of the concerned object instance
 * @param theAttributes an {AttributeHandleSet} specifying the pertinent attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see attributesOutOfScope
 * @see RTIambassador#enableAttributeScopeAdvisorySwitch
 * @see RTIambassador#disableAttributeScopeAdvisorySwitch
 */

public void attributesInScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
              AttributeNotRecognized,
              AttributeNotSubscribed,
              FederateInternalError;

// 6.15
/**
 * Notifies the federate that the specified attributes for the object instance are in its scope.
 * This occurs only if the Attribute Scope Advisory Switch is on for the federate.
 * @param theObject the {ObjectInstanceHandle} of the concerned object instance
 * @param theAttributes an {AttributeHandleSet} specifying the pertinent attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see attributesOutOfScope
 * @see RTIambassador#enableAttributeScopeAdvisorySwitch
 * @see RTIambassador#disableAttributeScopeAdvisorySwitch
 */

public void attributesInScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
              AttributeNotRecognized,
              AttributeNotSubscribed,
              FederateInternalError;
/**
 * Notifies the federate that the specified attributes for the object instance are out of its scope.
 * This occurs only if the Attribute Scope Advisory Switch is on for the federate.
 * @param theObject the @link ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an @link AttributeHandleSet specifying the pertinent attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see attributesInScope
 * @see RTIambassador#enableAttributeScopeAdvisorySwitch
 * @see RTIambassador#disableAttributeScopeAdvisorySwitch
 */

public void attributesOutOfScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
Requests of the federate the current values of the specified instance attributes, which it owns. The federate should respond through the \{link RTIambassador\#updateAttributeValues updateAttributeValues\} method. @param theObject the \{link ObjectInstanceHandle\} of the subject object instance @param theAttributes an \{link AttributeHandleSet\} specifying the requested attributes @param userSuppliedTag a \<code>byte[\]\</code> tag (this parameter may be \<code>null</code>) @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context @throws AttributeNotOwned should be thrown if the federate denies owning an attribute @throws FederateInternalError should be thrown if something else goes wrong @see RTIambassador\#requestAttributeValueUpdate requestAttributeValueUpdate /*
public void
provideAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag
) throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        FederateInternalError;
/**
 * Indicates to the federate that the values of the specified
 * instance attributes are required somewhere
 * in the federation execution. The federate should therefore
 * update them as needed.
 * This occurs only if the Attribute Relevance Advisory Switch is
 * on for the federate.
 * @param theObject the ObjectInstanceHandle of the subject
 * object instance
 * @param theAttributes an AttributeHandleSet specifying
 * the subject attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate
 * denies owning an attribute
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see #turnUpdatesOffForObjectInstance
 * @see #enableAttributeRelevanceAdvisorySwitch
 * @see #disableAttributeRelevanceAdvisorySwitch
 * @see #updateAttributeValues
 */

public void turnUpdatesOnForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
             AttributeNotRecognized,
             AttributeNotOwned,
             FederateInternalError;

// 6.19
/** Indicates to the federate that the values of the specified instance attributes are no longer required anywhere in the federation execution. This occurs only if the Attribute Relevance Advisory Switch is on for the federate. */

@throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance

@throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context

@throws AttributeNotOwned should be thrown if the federate denies owning an attribute

@throws FederateInternalError should be thrown if something else goes wrong

@see #turnUpdatesOnForObjectInstance

@throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance

@throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context

@throws AttributeNotOwned should be thrown if the federate denies owning an attribute

@throws FederateInternalError should be thrown if something else goes wrong

@see RTIambassador#enableAttributeRelevanceAdvisorySwitch

@see RTIambassador#disableAttributeRelevanceAdvisorySwitch

@see RTIambassador#updateAttributeValues

/**
   public void
   turnUpdatesOffForObjectInstance(
       ObjectInstanceHandle theObject,
       AttributeHandleSet   theAttributes)
   throws ObjectInstanceNotKnown,
           AttributeNotRecognized,
           AttributeNotOwned,
           FederateInternalError;
   */
Requests that the federate acquire ownership of the specified instance attributes.

This can occur because the original owner invoked `unconditionalAttributeOwnershipDivestiture` or `resignFederationExecution` (with the `UNCONDITIONALLY_DIVEST_ATTRIBUTES`, `DELETE_OBJECTS_THEN_DIVEST` or `CANCEL_THEN_DELETE_THEN_DIVEST` policy).

It will not occur if the original owner invoked the `unpublishObjectClassAttributes` service or if the `registerObjectInstance` service is used (and the newly-registered object instance has some unowned attributes).

The federate may return a subset of the `offeredAttributes` for which it is willing to assume ownership through the `attributeOwnershipAcquisition` or `attributeOwnershipAcquisitionIfAvailable` methods.

**param**

- `theObject` the `ObjectInstanceHandle` of the concerned object instance
- `offeredAttributes` an `AttributeHandleSet` specifying the offered attributes
- `userSuppliedTag` a `byte[]` tag (this parameter may be `null`)

@throws `ObjectInstanceNotKnown` should be thrown if the federate denies having previously discovered the object instance
@throws `AttributeNotRecognized` should be thrown if one of the supplied attributes isn't recognized in the supplied context
@throws `AttributeAlreadyOwned` should be thrown if the federate thinks it already owns an attribute
@throws `AttributeNotPublished` should be thrown if the federate denies publishing an attribute
@throws `FederateInternalError` should be thrown if something else goes wrong

```java
public void requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag) throws ObjectInstanceNotKnown,
                        AttributeNotRecognized,
                        AttributeAlreadyOwned,
                        AttributeNotPublished,
                        FederateInternalError;
```
// 7.5
/**
 * Notifies the federate that potential new owners have been found for the specified instance attributes and that
 * the negotiated divestiture of these can now be completed. The federate can either complete the negotiated
 * divestiture using {@link RTIambassador#confirmDivestiture confirmDivestiture}, divest ownership of the instance attributes by
 * some other means (e.g., using {@link RTIambassador#unconditionalAttributeOwnershipDivestiture unconditionalAttributeOwnershipDivestiture}) or it can
 * @link RTIambassador#cancelNegotiatedAttributeOwnershipDivestiture cancelNegotiatedAttributeOwnershipDivestiture
 * @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
 * @param offeredAttributes an {@link AttributeHandleSet} specifying the offered attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
 * @throws AttributeDivestitureWasNotRequested should be thrown if the federate repudiates the divestiture
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    FederateInternalError;
/**
 * Notifies the federate that it now owns the specified set of instance attributes.
 * The federate may receive multiple notifications for a single invocation of
 * {@link RTIambassador#attributeOwnershipAcquisition} or
 * {@link RTIambassador#attributeOwnershipAcquisitionIfAvailable} if the requested
 * instance attributes are owned by different federates.
 * @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
 * @param securedAttributes an {@link AttributeHandleSet} specifying the secured attributes
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeAcquisitionWasNotRequested should be thrown if the federate repudiates its attribute ownership acquisition request
 * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
 * @throws AttributeNotPublished should be thrown if the federate denies publishing an attribute
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAcquisitionWasNotRequested,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError;
// 7.10
/**
 * Notifies the federate that the specified instance attributes were not available for ownership acquisition.
 * @param theObject the {ObjectInstanceHandle} of the concerned object instance
 * @param theAttributes an {AttributeHandleSet} specifying the declined attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn’t recognized in the supplied context
 * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
 * @throws AttributeAcquisitionWasNotRequested should be thrown if the federate repudiates its attribute ownership acquisition request
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#attributeOwnershipAcquisitionIfAvailable
 */
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeAcquisitionWasNotRequested,
        FederateInternalError;
public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        FederateInternalError;
/**
 * Notifies the federate that the pending attribute ownership
 * acquisition requests for the
 * specified instance attributes have been canceled as requested.
 * @param theObject the {@link ObjectInstanceHandle} of the
 * concerned object instance
 * @param theAttributes an {@link AttributeHandleSet} specifying
 * the subject attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate
denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeAlreadyOwned should be thrown if the federate
* thinks it already owns an attribute
 * @throws AttributeAcquisitionWasNotCanceled should be thrown if
* the federate repudiates the attribute ownership acquisition
* cancellation
 * @throws FederateInternalError should be thrown if something else
* goes wrong
 * @see RTIambassador#cancelAttributeOwnershipAcquisition
* cancelAttributeOwnershipAcquisition
 */
public void
confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError;
In response to an attribute ownership query by the federate, specifies that the instance attribute is unowned.

```java
public void attributeIsNotOwned(
  ObjectInstanceHandle theObject,
  AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
  AttributeNotRecognized,
  FederateInternalError;
```

/*

**
* In response to an attribute ownership query by the federate, specifies that the instance attribute is unowned.
* @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
* @param theAttribute an {@link AttributeHandle} specifying the attribute
* @param theOwner the {@link FederateHandle} of the federate owning the attribute
* @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
* @throws FederateInternalError should be thrown if something else goes wrong
* @see RTIambassador#queryAttributeOwnership
queryAttributeOwnership
*/
/* In response to an attribute ownership query by the federate, specifies the instance attribute's owner: the RTI. * @param theObject the {@link ObjectInstanceHandle} of the concerned object instance * @param theAttribute an {@link AttributeHandle} specifying the attribute * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context * @throws FederateInternalError should be thrown if something else goes wrong * @see RTIambassador#queryAttributeOwnership */
public void attributeIsOwnedByRTI(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
    throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError;

////////////////////////////////////////////////////////////////////////////////
// Time Management Services //
////////////////////////////////////////////////////////////////////////////////

// 8.3
/**
 * Notifies the federate that its request to enable time-regulation has been honored.
 * @param time the {@link LogicalTime} to which the federate's clock has been set
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
 * @throws NoRequestToEnableTimeRegulationWasPending should be thrown if the federate repudiates the time regulation request
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#enableTimeRegulation */
public void timeRegulationEnabled(
    LogicalTime time)
    throws InvalidLogicalTime,
    NoRequestToEnableTimeRegulationWasPending,
    FederateInternalError;
public void timeConstrainedEnabled(LogicalTime time)
throws InvalidLogicalTime, NoRequestToEnableTimeConstrainedWasPending, FederateInternalError;

public void timeAdvanceGrant(LogicalTime theTime)
throws InvalidLogicalTime, JoinedFederateIsNotInTimeAdvancingState, FederateInternalError;
// B.22
/**
 * Notifies the federate that the previously delivered message
 * specified by the supplied
 * {@link MessageRetractionHandle} has been retracted.
 * <p>
 * Time-constrained federates that do not use the {@link
 * RTIambassador#flushQueueRequest flushQueueRequest} method
 * are not subject to invocation of this service because they will
 * never receive a
 * {@link OrderType TIMESTAMP} message eligible for retraction.
 * Non-constrained federates, however, must be prepared to deal
 * with invocations of this
 * service because any received message that was sent {@link
 * OrderType TIMESTAMP} may be eligible for retraction.
 * <p>
 * @param theHandle the {@link MessageRetractionHandle} specifying
 * the retracted message
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador#retract retract
 */
public void requestRetraction(
    MessageRetractionHandle theHandle)
    throws FederateInternalError{
    }
//end FederateAmbassador
// File: FederateHandle.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
   * Type-safe handle for a federate handle. Generally these are created
   * by the RTI and passed to the federate.
   * <p>
   * The federate obtains its own <code>FederateHandle</code> from the
   * (4.4) {@link RTIAmbassador#joinFederationExecution
   * joinFederationExecution}
   * and (4.21) {@link FederateAmbassador#initiateFederateRestore
   * initiateFederateRestore} methods; other federates'
   * <code>FederateHandle</code>s are received through
   * the (4.17) {@link FederateAmbassador#federationSaveStatusResponse
   * federationSaveStatusResponse} (as part of a {
   * @link
   * FederateHandleSaveStatusPair}[]),
   * (4.25) {@link FederateAmbassador#federationRestoreStatusResponse
   * federationRestoreStatusResponse} (as part of a {
   * @link
   * FederateHandleRestoreStatusPair}[]) and
   * (7.17) {@link FederateAmbassador#informAttributeOwnership
   * informAttributeOwnership} callbacks.
   * <p>
   * They can also be obtained by using the
   * <code>FederateHandleFactory</code>'s {
   * @link
   * FederateHandleFactory#decode
   * decode} method on a <code>byte[]</code> received as part of an attribute update or interaction.
   * <p>
   * The Management Object Model (MOM) publishes the
   * <code>HLAmanager.HLAfederate</code> object class, which has a
   * <code>HLAfederateHandle</code> attribute.
   * <p>
   * The MOM also publishes the <code>HLAmanager.HLAfederate</code>
   * interaction class (and its numerous sub-classes), which has a
   * <code>HLAfederate</code> parameter.
   * <p>
   * The aforementioned attribute and parameter both have values of type
   * <code>FederateHandle</code> (known to the MOM as
   * <code>HLAHandle</code>).<p>
   * <p>
   * <code>FederateHandle</code>s are used to build a {
   * @link
   * FederateHandleSet} which can be passed to
   * the (4.6) {
   * @link
   * RTIAmbassador#registerFederationSynchronizationPoint
   * registerFederationSynchronizationPoint} method.
   * <p>
   * Lastly, the (10.33) {
   * @link
   * RTIAmbassador#normalizeFederateHandle
   * normalizeFederateHandle}
   * method can be used to project a <code>FederateHandle</code> onto
   * the <code>Federates</code>
   * <p>
   * dimension for <code>Region</code> specification purposes.
   * @author IEEE
   * @version 1516.1.5 (DoD v2)
   */
public interface FederateHandle
    extends java.io.Serializable
{
    /**
     * Returns a <code>String</code> representation of the
     * <code>FederateHandle</code>.
     * @return A {@link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String toString();

    /**
     * Returns true iff <code>this</code> and
     * <code>otherFederateHandle</code> represent the same federate handle.
     * @param otherFederateHandle the <code>Object</code> to compare
     * with
     * @return <code>true</code> iff supplied
     * <code>otherFederateHandle</code> is of type
     * <code>FederateHandle</code> and has same value
     */
    public boolean equals(Object otherFederateHandle);

    /**
     * Returns a hash code for <code>this</code>. Two
     * <code>FederateHandle</code>s for which <code>equals()</code> is
     * <code>true</code> should yield the same hash code.
     * @return An <code>int</code> hash code
     */
    public int hashCode();

    /**
     * Returns the length (in bytes) of the <code>byte[]</code>
     * representation of the <code>FederateHandle</code>. 
     * @return The length (in bytes) of the <code>byte[]</code>
     * representation of <code>this</code>
     */
    public int encodedLength();

    /**
     * Encodes <code>this</code> into the <code>byte[]</code> at the
     * specified <code>offset</code>. 
     * @param buffer the <code>byte[]</code> into which to encode
     * <code>this</code>
     * @param offset the offset into the <code>byte[]</code> at which
to encode <code>this</code>
     */
    public void encode(byte[] buffer,
            int offset);
}
//end FederateHandle
// File: FederateHandleFactory.java
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
   * Interface corresponding to the {@link RTIambassador#getFederateHandleFactory getFederateHandleFactory} method.
   * Its one method creates a new {@link FederateHandle} from a supplied <code>byte[]</code> representation,
   * itself received as an attribute or parameter value.
   * @author IEEE
   * @version 1516.1.5 (DoD V2)
   */
public interface FederateHandleFactory

   extends java.io.Serializable

   /**
     * Creates a new {@link FederateHandle} from the supplied <code>byte[]</code> representation.
     * @param buffer A <code>byte[]</code> containing a representation of a <code>FederateHandle</code>
     * @param offset Offset into the <code>buffer</code> at which the representation of the <code>FederateHandle</code> begins
     * @return A <code>FederateHandle</code> constructed from the buffer's contents
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory's target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
   public FederateHandle decode (byte[] buffer,
                              int offset)
                      throws CouldNotDecode,
                              FederateNotExecutionMember;

//end FederateHandleFactory
import java.io.Serializable;

public final class FederateHandleRestoreStatusPair
    implements java.io.Serializable {

    /**
     * The federate handle.
     */
    public FederateHandle handle;

    /**
     * The federate’s restoration status.
     */
    public RestoreStatus status;

    /**
     * Public constructor. Not expected to be used by the federate but rather by the RTI.
     * @param fh The <code>FederateHandle</code> of the federate whose corresponding <code>RestoreStatus</code> is specified by this record
     * @param rs The <code>RestoreStatus</code> of the federate specified by the <code>FederateHandle</code> field of this record
     */
    public FederateHandleRestoreStatusPair(FederateHandle fh, RestoreStatus rs) {
        handle = fh;
        status = rs;
    }
}
package hla.rti1516;  // the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * The (4.17) {@link FederateAmbassador#federationSaveStatusResponse
 federationSaveStatusResponse}
 * callback returns an array of these records. For each joined
 * federate, there will
 * be a <code>FederateHandleSaveStatusPair</code> record returned,
 * specifying the
 * {@link SaveStatus} of each {@link FederateHandle}.
 * */

public final class FederateHandleSaveStatusPair
    implements java.io.Serializable

{/**
 * The federate handle.
 */
    public FederateHandle handle;

/**
 * The federate's save status.
 */
    public SaveStatus status;

/** *
 * Public constructor. Not expected to be used by the federate but
 * rather by the RTI.
 * @param fh The <code>FederateHandle</code> of the federate whose
 * corresponding <code>SaveStatus</code> is specified by this record
 * @param ss The <code>SaveStatus</code> of the federate specified
 * by the <code>FederateHandle</code> field of this record
 */
    public FederateHandleSaveStatusPair(FederateHandle fh,
                                          SaveStatus ss) {
        handle = fh;
        status = ss;
    }
}
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * A Set of @link FederateHandle}s.
 * All @link java.util.Set} operations are required, none are optional.
 * @code add()\endcode and @code remove()\endcode should throw @link IllegalArgumentException} if the argument is not a @code FederateHandle\endcode.
 * @code addAll()\endcode, @code removeAll()\endcode and @code retainAll()\endcode should throw @code IllegalArgument EXCEPTION\endcode if the argument is not a @code FederateHandleSet\endcode.
 * @p Used with the (4.6) @link RTIambassador#registerFederationSynchronizationPoint{String,byte[],FederateHandleSet} method.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see FederateHandleSetFactory
 */

public interface FederateHandleSet
    extends java.io.Serializable,
            java.lang.Cloneable,
            java.util.Set
{
}

//end FederateHandleSet
/**
 * Interface corresponding to the {@link 
 * RTIambassador\#getFederateHandleSetFactory getFederateHandleSetFactory} method.
 * Its one method creates a new {@link FederateHandleSet},
 * to be loaded up and then passed to the \[4.6\] {@link 
 * RTIambassador\#registerFederationSynchronizationPoint registerFederationSynchronizationPoint} method.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface FederateHandleSetFactory
extends java.io.Serializable
{

    /**
     * Creates a new {@link FederateHandleSet}, initially empty.
     * @return An empty <code>FederateHandleSet</code>
     */
    public FederateHandleSet create();
}

//end FederateHandleSetFactory
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * Type-safe handle for an interaction class. Generally these are
 * created by the RTI and passed to the federate.
 */
public interface InteractionClassHandle
   extends java.io.Serializable
{
   /**
    * Returns a <code>String</code> representation of the
    * <code>InteractionClassHandle</code>.
    * @return A {@link java.lang.String} reflecting <code>this</code>
    */
   public String toString();

   /**
    * Returns true iff <code>this</code> and
    * <code>otherInteractionClassHandle</code> represent the same
    * interaction class handle.
    * @param otherInteractionClassHandle the <code>Object</code> to
    * compare with
    * @return <code>true</code> iff supplied
    * <code>otherInteractionClassHandle</code> is of type
    * <code>InteractionClassHandle</code> and has same value
    */
   public boolean equals(Object otherInteractionClassHandle);

   /**
    * Returns a hash code for <code>this</code>; two
    * <code>InteractionClassHandle</code>s for which <code>equals()</code>
    * is <code>true</code> should yield the same hash code.
    * @return An <code>int</code> hash code
    */
   public int hashCode();
/**
 * Returns the length (in bytes) of the byte[] representation of the InteractionClassHandle.
 * @return The length (in bytes) of the byte[] representation of this
 */
public int encodedLength();

/**
 * Encodes this into the byte[] at the specified offset.
 * @param buffer the byte[] into which to encode this
 * @param offset the offset into the byte[] at which to encode this
 */
public void encode(byte[] buffer,
        int offset);
} //end InteractionClassHandle
// File: InteractionClassHandleFactory.java
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
   * Interface corresponding to the {link RTIambassador#getInteractionClassHandleFactory
   * getInteractionClassHandleFactory} method.
   * Its one method creates a new {link InteractionClassHandle} from a supplied <code>byte[]</code> representation,
   * itself received as an attribute or parameter value.
   * @author IEEE
   * @version 1516.1.5 (DoD V2)
   */
public interface InteractionClassHandleFactory
    extends java.io.Serializable

    /**
     * Creates a new {link InteractionClassHandle} from the supplied <code>byte[]</code> representation.
     * @param buffer A <code>byte[]</code> containing a representation of an <code>InteractionClassHandle</code>
     * @param offset Offset into the <code>buffer</code> at which the representation of the <code>InteractionClassHandle</code> begins
     * @return An <code>InteractionClassHandle</code> constructed from the buffer's contents
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory's target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
    public InteractionClassHandle decode(byte[] buffer, int offset)
        throws CouldNotDecode, FederateNotExecutionMember;
}

//end InteractionClassHandleFactory
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * An interface to an immutable time value,
 * one of the four interfaces which the federate implements if it
 * wishes to translate
 * the federation’s logical times and logical time intervals into
 * something meaningful for itself.
 * It is best if all federates in a given federation use the same
 * implementations, but it is not strictly necessary.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see LogicalTimeFactory
 */
public interface LogicalTime
        extends Comparable,
                java.io.Serializable
{

    /**
     * Whether <code>this</code> is the "initial" logical time value or
     * not (the smallest logical time possible).
     * @return <code>true</code> iff <code>this</code> is the initial
     * time
     */
    public boolean isInitial();

    /**
     * Whether <code>this</code> is the "final" logical time value or
     * not (the largest logical time possible).
     * @return <code>true</code> iff <code>this</code> is the final
     * time
     */
    public boolean isFinal();

    /**
     * Returns a new <code>LogicalTime</code> whose value is
     * <code>this</code> moved into the future by the <code>val</code>
     * <code>LogicalTimeInterval</code>.
     * @param val a <code>LogicalTimeInterval</code> whose value is
     * "added" to <code>this</code> value
     * @return The <code>LogicalTime</code> resulting from the addition
     * @throws IllegalTimeArithmetic if the result would be beyond the
     * end time
     */
    public LogicalTime add(LogicalTimeInterval val)
            throws IllegalTimeArithmetic;
/**
 * Returns a new <code>LogicalTime</code> whose value is
 * <code>this</code> moved into the past by the <code>val</code>
 * <code>LogicalTimeInterval</code>.  
 * @param val a <code>LogicalTimeInterval</code> whose value is
 * "subtracted" from <code>this</code> value
 * @return The <code>LogicalTime</code> resulting from the
 * subtraction
 * @throws IllegalTimeArithmetic if the result would be before the
 * start time
 */
public LogicalTime
subtract(LogicalTimeInterval val)
throws IllegalTimeArithmetic;

/**
 * Returns a new <code>LogicalTimeInterval</code> whose value is
 * the logical time interval separating <code>this</code> from
 * <code>val</code>.  
 * Although this is not clearly stated, the resulting logical time
 * interval is a magnitude (that is, it cannot be negative).  
 * Therefore, lti1.distance(lti2) == lti2.distance(lti1)
 * @param val a <code>LogicalTime</code> whose value is compared
 * with <code>this</code> value
 * @return The <code>LogicalTimeInterval</code> separating the two
 * <code>LogicalTime</code>s
 */
public LogicalTimeInterval
distance(LogicalTime val);

// Comparable implementation.
/**
 * Compares <code>this</code> object with the specified
 * <code>object</code> for order.
 * Later logical time is greater, earlier logical time is smaller.
 * @param other the <code>object</code> to compare
 * @return A negative, zero or positive integer if
 * <code>this</code> object is less than, equal to or greater than the
 * <code>other</code>
 */
public int
compareTo(Object other);

/**
 * Returns a <code>String</code> representation of the
 * <code>LogicalTime</code>.
 * @return A [@link java.lang.String] representation of
 * <code>this</code>
 */
public String
toString();
/**
 * Returns true iff <code>this</code> and <code>other</code> represent the same logical time.
 * @param other the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>other</code> is of the same type and has the same value
 */
public boolean equals(Object other);

/**
 * Returns a hash code for <code>this</code>; two <code>LogicalTime</code>s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of the <code>LogicalTime</code>.
 * @return The length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the specified <code>offset</code>.  
 * @param buffer the <code>byte[]</code> into which to encode <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
 */
public void encode(byte[] buffer,
                   int offset);

//end LogicalTime
/**
 * One of the four interfaces which the federate implements if it wishes to translate
 * the federation's logical times and logical time intervals into something meaningful for itself.
 * It is best if all federates in a given federation use the same implementations, but it is not strictly necessary.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface LogicalTimeFactory extends java.io.Serializable
{
    /**
     * Creates a new {@link LogicalTime} from the supplied <code>byte[]</code> representation.
     * @param buffer A <code>byte[]</code> containing a representation of a <code>LogicalTime</code>
     * @param offset Offset into the <code>buffer</code> at which the representation of the <code>LogicalTime</code> begins
     * @return A <code>LogicalTime</code> constructed from the buffer's contents
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory's target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
    public LogicalTime decode(byte[] buffer, int offset) throws CouldNotDecode;

    /**
     * Creates a new {@link LogicalTime} of "initial" value (the smallest <code>LogicalTime</code> possible).
     * @return A <code>LogicalTime</code> of initial value
     */
    public LogicalTime makeInitial();

    /**
     * Creates a new {@link LogicalTime} of "final" value (the largest <code>LogicalTime</code> possible).
     * @return A <code>LogicalTime</code> of final value
     */
    public LogicalTime makeFinal();
}
//end LogicalTimeFactory
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
   * An interface to an immutable time interval value,
   * one of the four interfaces which the federate implements if it
   * wishes to translate
   * the federation's logical times and logical time intervals into
   * something meaningful for itself.
   * It is best if all federates in a given federation use the same
   * implementations, but it is not strictly necessary.
   * @author IEEE
   * @version 1516.1.5 (DoD v2)
   * @see LogicalTimeIntervalFactory
   */
public interface
LogicalTimeInterval
   extends Comparable,
        java.io.Serializable
{
   /**
      * Whether <code>this</code> is the zero length logical time
      * interval or not.
      * @return <code>true</code> iff <code>this</code> is of length
      * zero
      */
   public boolean
   isZero();

   /**
      * Whether <code>this</code> is the epsilon length logical time
      * interval or not (epsilon is the smallest non-zero logical time
      * interval possible).
      * @return <code>true</code> iff <code>this</code> is of length
      * epsilon
      */
   public boolean
   isEpsilon();

   /**
      * Returns a new <code>LogicalTimeInterval</code> whose value is
      * <code>(this - subtrahend) </code>.  
      * One instance of an <code>LogicalTimeInterval</code> may be
      * subtracted from another, but not added.
      * @param subtrahend a <code>LogicalTimeInterval</code> whose value
      * is subtracted from <code>this</code> value
      * @return The <code>LogicalTimeInterval</code> resulting from the
      * subtraction
      */
   public LogicalTimeInterval
   subtract(LogicalTimeInterval subtrahend);
// Comparable implementation

/**
 * Compares <code>this</code> object with the specified <code>object</code> for order.
 * @param other the <code>object</code> to compare <code>this</code> with
 * @return A negative, zero or positive integer if 
 * <code>this</code> object is less than, equal to or greater than the 
 * <code>other</code>
 * @throws java.lang.ClassCastException if the <code>other</code> is incommensurable
 */
public int compareTo(Object other);

/**
 * Returns a <code>String</code> representation of the 
 * <code>LogicalTimeInterval</code>. 
 * @return A {@link java.lang.String} representation of 
 * <code>this</code>
 */
public String toString();

/**
 * Returns true iff <code>this</code> and <code>other</code> 
 * represent the same logical time interval. 
 * @param other the <code>Object</code> to compare with 
 * @return <code>true</code> iff supplied <code>other</code> is of 
 * the same type and has the same value 
 */
public boolean equals(Object other);

/**
 * Returns a hash code for <code>this</code>; two 
 * <code>LogicalTimeInterval</code>s for which <code>equals()</code> is 
 * <code>true</code> should yield the same hash code. 
 * @return An <code>int</code> hash code 
 */
public int hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]</code> 
 * representation of the <code>LogicalTimeInterval</code>. 
 * @return The length (in bytes) of the <code>byte[]</code> 
 * representation of <code>this</code> 
 */
public int encodedLength();
/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the
 * specified <code>offset</code>.
 * @param buffer the <code>byte[]</code> into which to encode
 * <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which
 * to encode <code>this</code>
 */
 public void
 encode(byte[] buffer,
 int offset);

 //end LogicalTimeInterval
public interface LogicalTimeIntervalFactory
   extends java.io.Serializable
{
   /**
    * Creates a new <code>LogicalTimeInterval</code> from the supplied
    * <code>byte[]</code> representation.
    * @param buffer A <code>byte[]</code> containing a representation
    * of a <code>LogicalTimeInterval</code>
    * @param offset Offset into the <code>buffer</code> at which the
    * representation of the <code>LogicalTimeInterval</code> begins
    * @return A <code>LogicalTimeInterval</code> constructed from the
    * buffer's contents
    * @throws CouldNotDecode if the <code>buffer</code> could not be
decoded into an instance of the factory’s target class
    * @throws FederateNotExecutionMember if the federate is not
    * currently joined to a federation execution
    */
   public LogicalTimeInterval
      decode(byte[] buffer,
               int offset)
      throws CouldNotDecode;

   /**
    * Creates a new <code>LogicalTimeInterval</code> of value zero.
    * @return A <code>LogicalTimeInterval</code> of value zero
    */
   public LogicalTimeInterval
      makeZero();

   /**
    * Creates a new <code>LogicalTimeInterval</code> of value epsilon, the
    * smallest non-zero <code>LogicalTimeInterval</code> possible for this
    * implementation.
    * @return A <code>LogicalTimeInterval</code> of value epsilon
    */
   public LogicalTimeInterval
      makeEpsilon();
}

//end LogicalTimeIntervalFactory
package hla.rti1516;

// the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * The user can do nothing with these but employ them as keys.
 * Implementers should provide <code>equals</code>,
 * <code>hashCode</code> and <code>toString</code>
 * rather than settling for the defaults.
 * <p>
 * This object is used by the (8.21) {@link RTIambassador#retract}
 * retract} method.
 * <p>
 * It is also received by the (6.7) {@link 
 * FederateAmbassador#reflectAttributeValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle}]
 * is also received by the (6.7) {@link 
 * FederateAmbassador#reflectAttributeValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet}]
 * (6.9) {@link 
 * FederateAmbassador#receiveInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet)]
 * (6.9) {@link 
 * FederateAmbassador#receiveInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet)}
 * (6.11) {@link 
 * FederateAmbassador#removeObjectInstance(ObjectInstanceHandle,byte[],OrderType,LogicalTime,OrderType,MessageRetractionHandle)} and
 * (8.22) {@link FederateAmbassador#requestRetraction requestRetraction}
 * callbacks.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public interface MessageRetractionHandle
    extends java.io.Serializable
{
    /**
     * Returns a <code>String</code> representation of the
     * <code>MessageRetractionHandle</code>.
     * @return A <code>String</code> reflecting <code>this</code>
     * value
     */
    public String toString();
/**
 * Returns true iff <code>this</code> and <code>otherMRHandle</code> represent the same message retraction handle.
 * @param otherMRHandle The <code>Object</code> to compare with
 * @return true iff supplied <code>otherMRHandle</code> is of type <code>MessageRetractionHandle</code> and has same value
 */
public boolean equals(Object otherMRHandle);

/**
 * Returns a hash code for <code>this</code>; two <code>MessageRetractionHandle</code>s for which <code>equals()</code>
 * is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode();

} //end MessageRetractionHandle
The `MessageRetractionReturn` class, unlike the similar `TimeQueryReturn` class and the other IEEE 1516 object classes, does not override the `Object-inherited` `toString`, `equals` and `hashCode` methods. On the assumption that this is an unfortunate oversight, commented-out implementations are provided here.

```
// File: MessageRetractionReturn.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Record returned by the (6.6) `{@link
RTIambassador#updateAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],LogicalTime)}`,
 * (6.8/9.12) `{@link
RTIambassador#sendInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],LogicalTime)}` `{@link
RTIambassador#sendInteractionWithRegions(InteractionClassHandle,ParameterHandleValueMap,RegionHandleSet,byte[],LogicalTime) [WithRegions]}` and
 * (6.10) `{@link
RTIambassador#deleteObjectInstance(ObjectInstanceHandle,byte[],LogicalTime)}` methods.
 * It consists of a guard boolean `{code>retractionHandleIsValid</code> and a payload `{@link
MessageRetractionHandle} `{code>handle</code>.
 * The latter, if valid, may be used with the (8.21) `{@link
RTIambassador#retract} method.
 *
 * DoD Interpretations of IEEE 1516-2000v2 adds the java.io.Serializable implementation and a constructor.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public final class
MessageRetractionReturn
    implements java.io.Serializable
{
    /**
     * Whether the other field is valid or not.
     */
    public boolean
retractionHandleIsValid;

    /**
     * Payload Message Retraction Handle field.
     */
    public MessageRetractionHandle
    handle;
```
/**
 * Public constructor.
 * @param rhiv Whether the{@link MessageRetractionHandle} field is 
 * valid or not
 * @param mrh The <code>MessageRetractionHandle</code> field
 */
public
MessageRetractionReturn(boolean rhiv,
MessageRetractionHandle mrh)
{
  retractionHandleIsValid = rhiv;
  handle = mrh;
}

/**
 * Returns a <code>String</code> representation of the 
 * <code>MessageRetractionReturn</code>. 
 * <p>
 * Strangely missing from 1516.1.5 (DoDv2)
 * @return A {@link java.lang.String} with value
 * "&lt;retractionHandleIsValid&gt; &lt;handle&gt;" 
 */
public String
toString()
{//
  // return retractionHandleIsValid + " " + handle;
//}
/**
 * Returns true iff <code>this</code> and <code>other</code>
 * represent the same message retraction return.
 * @param other The <code>Object</code> to compare with
 * @return true iff supplied <code>other</code> is of type
 * <code>MessageRetractionReturn</code> and has same value
 */
/public boolean
// equals(Object other)
//
// if (other instanceof MessageRetractionReturn)
//
// MessageRetractionReturn mrrOther =
// (MessageRetractionReturn)other;
// if ((retractionHandleIsValid == false) && (mrrOther.retractionHandleIsValid == false))
//
// When retractionHandleIsValid is false, the payloads are ignored
// return true;  
// else if ((retractionHandleIsValid == true) && (mrrOther.retractionHandleIsValid == true))
//
// When retractionHandleIsValid is true, the payloads must match
// return handle.equals(mrrOther.handle);
//
// else
//
// mismatches retractionHandleIsValid fields
// return false;
//
//
// else
//
// Not the same classes
// return false;
//
// }
//
//
/**
 * Returns a hash code for <code>this</code>; two
 * <code>MessageRetractionReturn</code>s for which <code>equals()</code>
 * is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
// hashCode()
//
// return (retractionHandleValid ? handle.hashCode() : 7);
//
//end MessageRetractionReturn

214  DRDC Valcartier TR 2007-412
package hla.rti1516; // the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * Conveys the interfaces for all services that a federate
 * must supply and which may not execute in the federate's space.
 * This is used by the (4.4) {@link RTIambassador#joinFederationExecution joinFederationExecution} method.
 * 
 * DoD Interpretations of IEEE 1516-2000v2 adds the java.io.Serializable implementation.
 * 
 * The Java API is the only one which refers to the [{@link LogicalTimeFactory} and {@link LogicalTimeIntervalFactory}]
 * instances being passed to the <code>joinFederationExecution</code> method as "Mobile Federate Services" (Ada and C++
 * simply pass the instances as separate parameters). It is not clear
 * either what is meant by "which may
 * not execute" -does it mean they're not allowed to execute in the
 * federate's space, or does it mean
 * they could be invoked outside of its space? The latter seems more
 * likely.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public final class MobileFederateServices
    implements java.io.Serializable
{
    /**
     * The <code>LogicalTimeFactory</code> reference.
     */
    public LogicalTimeFactory _timeFactory;

    /**
     * The <code>LogicalTimeIntervalFactory</code> reference.
     */
    public LogicalTimeIntervalFactory _intervalFactory;
/**
 * Public constructor.
 * @param timeFactory An implementation of {@link LogicalTimeFactory}
 * @param intervalFactory An implementation of {@link LogicalTimeIntervalFactory}
 */

public MobileFederateServices(
    LogicalTimeFactory timeFactory,
    LogicalTimeIntervalFactory intervalFactory)
{
    _timeFactory = timeFactory;
    _intervalFactory = intervalFactory;
}

//end MobileFederateServices
package hla.rti1516;  //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Type-safe handle for an object class. Generally these are created
 * by the RTI and passed to the federate.
 * <p>
 * They are obtained from the (10.2) {link RTIambassador#getObjectClassHandle getobjectClassHandle} and
 * (10.16) {link RTIambassador#getKnownObjectClassHandle getKnownObjectClassHandle}
 * methods and are used by a variety of other methods.
 * <p>
 * They can also be obtained by using the
 * <code>ObjectClassHandleFactory</code>'s {link ObjectClassHandleFactory#decode decode} method on a <code>byte[</code> received as part of an attribute
 * update or interaction.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface ObjectClassHandle
    extends java.io.Serializable
{
    /**
     * Returns a <code>String</code> representation of the
     * <code>ObjectClassHandle</code>.
     * @return A {link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String toString();

    /**
     * Returns true iff <code>this</code> and
     * <code>otherObjectClassHandle</code> represent the same object class
     * handle.
     * @param otherObjectClassHandle the <code>Object</code> to compare
     * with
     * @return <code>true</code> iff supplied
     * <code>otherObjectClassHandle</code> is of type
     * <code>ObjectClassHandle</code> and has same value
     */
    public boolean equals(Object otherObjectClassHandle);

    /**
     * Returns a hash code for <code>this</code>; two
     * <code>ObjectClassHandle</code>'s for which <code>equals()</code> is
     * <code>true</code> should yield the same hash code.
     * @return An <code>int</code> hash code
     */
    public int hashCode();
/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of the <code>ObjectClassHandle</code>.  
 * @return The length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the specified <code>offset</code>.  
 * @param buffer the <code>byte[]</code> into which to encode <code>this</code>  
 * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
 */
public void encode(byte[] buffer, int offset);
//end ObjectClassHandle
public interface ObjectClassHandleFactory
    extends java.io.Serializable
{
    public ObjectClassHandle decode(byte[] buffer, int offset)
        throws CouldNotDecode, FederateNotExecutionMember;
}

// File: ObjectClassHandleFactory.java
package hla.rti2000v2;

/**
 * Interface corresponding to the
 * RTIambassador#getObjectClassHandleFactory
 * method.
 * Its one method creates a new
 * ObjectClassHandle from a
 * supplied byte[] representation,
 * itself received as an attribute or parameter value.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface ObjectClassHandleFactory
    extends java.io.Serializable
{
    /**
     * Creates a new ObjectClassHandle from the supplied
     * byte[] representation.
     * @param buffer A byte[] containing a representation
     *    of an ObjectClassHandle
     * @param offset Offset into the buffer at which the
     *    representation of the ObjectClassHandle begins
     * @return An ObjectClassHandle constructed from the
     *    buffer’s contents
     * @throws CouldNotDecode if the buffer could not be
     *    decoded into an instance of the factory’s target class
     * @throws FederateNotExecutionMember if the federate is not
     *    currently joined to a federation execution
     */
    public ObjectClassHandle decode(byte[] buffer, int offset)
        throws CouldNotDecode, FederateNotExecutionMember;
}

DRDC Valcartier TR 2007-412
public interface ObjectInstanceHandle
    extends java.io.Serializable {

        /**
         * Returns a <code>String</code> representation of the
         * <code>ObjectInstanceHandle</code>.
         * @return A <link java.lang.String> reflecting <code>this</code>
         * value
         */
        public String toString();

        /**
         * Returns true iff <code>this</code> and
         * <code>otherObjectInstanceHandle</code> represent the same object
         * instance handle.
         * @param otherObjectInstanceHandle the <code>Object</code> to
         * compare with
         * @return <code>true</code> iff supplied
         * <code>otherObjectInstanceHandle</code> is of type
         * <code>ObjectInstanceHandle</code> and has same value
         */
        public boolean equals(Object otherObjectInstanceHandle);
/**
 * Returns a hash code for <code>this</code>; two 
 * <code>ObjectInstanceHandle</code>s for which <code>equals()</code> is 
 * <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]</code> 
 * representation of the <code>ObjectInstanceHandle</code>. 
 * @return The length (in bytes) of the <code>byte[]</code> 
 * representation of <code>this</code>
 */
public int encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the 
 * specified <code>offset</code>. 
 * @param buffer the <code>byte[]</code> into which to encode 
 * <code>this</code> 
 * @param offset the offset into the <code>byte[]</code> at which 
 * to encode <code>this</code>
 */
public void encode(byte[] buffer, int offset);
} //end ObjectInstanceHandle
package hla.rti1516; //the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * Interface corresponding to the {@link RTIambassador#getObjectInstanceHandleFactory
 * getObjectInstanceHandleFactory} method.
 * Its one method creates a new {@link ObjectInstanceHandle} from a
 * supplied <code>byte[]</code> representation,
 * itself received as an attribute or parameter value.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public interface ObjectInstanceHandleFactory
  extends java.io.Serializable
/**
 * Creates a new {@link ObjectInstanceHandle} from the supplied
 * <code>byte[]</code> representation.
 * @param buffer A <code>byte[]</code> containing a representation
 * of an <code>ObjectInstanceHandle</code>
 * @param offset Offset into the <code>buffer</code> at which the
 * representation of the <code>ObjectInstanceHandle</code> begins
 * @return An <code>ObjectInstanceHandle</code> constructed from
 * the buffer's contents
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded into an instance of the factory's target class
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 */
public ObjectInstanceHandle
  decode(byte[] buffer,
          int offset)
  throws CouldNotDecode,
          FederateNotExecutionMember;
}
/**
 * An enumerated type (not a Java \{@link \java.util.Enumeration\})
 * representing the RTI-provided means of ordering messages
 * originating from multiple joined federates that are delivered to a
 * single joined federate.
 * Different categories of service are defined with different
 * characteristics regarding whether and how an RTI orders messages that
 * are to be delivered to a joined federate.
 * Ordering Type defaults are defined at the attribute and parameter
 * level by the POM Document Data; these may be overridden by their
 * owners.
 * The two OrderTypes are:
 * <ul>
 * <li><code>RECEIVE</code>: Messages are in an arbitrary order but
 * the general policy is "as soon as possible".
 * <li><code>TIMESTAMP</code>: An effort is made to deliver time-
 * stamped messages in order of increasing time-stamps
 * </ul>
 * Unlike \{@link TransportationType\}, the standard does not allow
 * specific RTIs to define additional OrderTypes.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public final class OrderType
    implements \java.io.Serializable
{
    // each instance's value
    private int _value;
    // initial value for enumeration
    private static final int lowestValue = 1;
    // begins at lowest
    private static int nextToAssign = lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of an \{code\}OrderType\{code\} must be
     * initialized with one of the defined static values.
     * @param otherOrderTypeValue must be a defined static value or
     * another instance.
     */
    public OrderType(OrderType otherOrderTypeValue)
    {
        _value = otherOrderTypeValue._value;
    }
}
/**
 * Package-only (default access) constructor. Used by the [decode decode] method.
 * @param value an <code>int</code> to assign to the instance; must be one of the static values
 */
OrderType(int value)
    throws RTIinternalError
{
    _value = value;
    if (!(_value < _lowestValue) || (value >= _nextToAssign))
        throw new RTIinternalError("OrderType: illegal value " +
     value);
}

/**
 * Private constructor; it is used to generate the static values.
 */
private OrderType()
{
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the
 * <code>OrderType</code>.  
 * @return A [java.lang.String] with value "OrderType(n)" where n is <code>this</code> value
 */
public String
toString()
{
    return "OrderType(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and
 * <code>otherOrderTypeValue</code> represent the same order type.
 * @param otherOrderTypeValue the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherOrderTypeValue</code> is of type <code>OrderType</code> and has same value
 */
public boolean
equals(Object otherOrderTypeValue)
{
    if (otherOrderTypeValue instanceof OrderType)
        return _value == ((OrderType)otherOrderTypeValue)._value;
    else
        return false;
}
/**
 * Returns a hash code for <code>this</code>; two <code>OrderType</code>s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode()
{
    return _value;
}

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of the <code>OrderType</code>.
 * @return The length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength()
{
    return 1;
}

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the specified <code>offset</code>.
 * @param buffer the <code>byte[]</code> into which to encode <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
 */
public void encode(byte[] buffer,
    int offset)
{
    buffer[offset] = (byte)_value;
}
/**
 * Creates an <code>OrderType</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>OrderType</code>
 * @param offset where in the <code>buffer</code> the <code>OrderType</code> representation begins
 * @return The <code>OrderType</code> that was encoded in the provided <code>buffer</code>  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory's target class
 */
public static OrderType decode(byte[] buffer,  
    int offset)  
    throws CouldNotDecode
{
    int val = buffer[offset];
    OrderType neo;
    try
    {
        neo = new OrderType(val);
    }
    catch (RTIinternalError e)
    {
        throw new CouldNotDecode(e.getMessage());
    }
    return neo;
}
The constant instances

** Receive Order (RO).
  A characteristic of no ordering guarantee for messages.
  RECEIVE messages will be received in an arbitrary order by the
  respective joined federate.
  A time stamp value will be provided with the message if one was
  specified when the message was sent,
  but that time stamp has no bearing on message receipt order.
*/
static public final OrderType
  RECEIVE = new OrderType();

/**
 * Time Stamp Order (TSO).
 * An ordering of messages provided by the runtime infrastructure
 (RTI) for joined federates
 * making use of time management services and messages containing
 time stamps.
 * TIMESTAMP messages are said to be delivered in TSO if, for any
 two messages M1 and M2
 * (time stamped with T1 and T2, respectively) that are delivered
 to a single joined federate
 * and where T1 < T2, then M1 is delivered before M2 (in real
 time).
 * Messages having the same time stamp will be delivered in an
 arbitrary order
 * (i.e., no tie-breaking mechanism is provided by the RTI).
 */
static public final OrderType
  TIMESTAMP = new OrderType();
} //end OrderType
public interface ParameterHandle

    extends java.io.Serializable

    /**
     * Returns a String representation of the ParameterHandle.
     * @return A java.lang.String reflecting this value
     */
    public String toString();

    /**
     * Returns true iff this and otherParameterHandle represent the same parameter handle.
     * Note that two handles may be equals but still refer to different parameters if used in different interaction class contexts.
     * @param otherParameterHandle the Object to compare with
     * @return true iff supplied otherParameterHandle is of type ParameterHandle and has same value
     */
    public boolean equals(Object otherParameterHandle);
/**
 * Returns a hash code for <code>this</code>; two <code>ParameterHandle</code>s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int
hashCode();

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of the <code>ParameterHandle</code>.
 * @return The length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int
encodedLength();

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the specified <code>offset</code>.
 * @param buffer the <code>byte[]</code> into which to encode <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
 */
public void
encode(byte[] buffer,
        int offset);
} //end ParameterHandle
public interface ParameterHandleFactory
    extends java.io.Serializable
{

    /**
     * Creates a new ParameterHandle from the supplied byte[] representation.
     * @param buffer A byte[] containing a representation of a ParameterHandle
     * @param offset Offset into the buffer at which the representation of the ParameterHandle begins
     * @return A ParameterHandle constructed from the buffer's contents
     * @throws CouldNotDecode if the buffer could not be decoded into an instance of the factory's target class
     * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
     */
    public ParameterHandle decode(byte[] buffer, int offset)
        throws CouldNotDecode, FederateNotExecutionMember;
}
Interpretations of IEEE 1516-2000v2

/**
 * Object used to transmit the set of parameters and their values involved in an interaction.
 * Keys are {@link ParameterHandle}s; values are byte[].
 * All {@link java.util.Map} operations are required, none optional.
 * Null mappings are not allowed.
 * <code>put(), putAll(), and remove()</code> should throw {@link IllegalArgumentException}
 * to enforce types of keys (@link ParameterHandle) and mappings (code>byte[]</code>).
 * <p>
 * Sent by the
 * (6.8/9.12) {@link RTIambassador#sendInteraction sendInteraction}
 * {@link RTIambassador#sendInteractionWithRegions [WithRegions]} (all forms) method.
 * Received by the (6.9) {@link FederateAmbassador#receiveInteraction receiveInteraction} callback (all forms).
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see ParameterHandleValueMapFactory
 */

public interface ParameterHandleValueMap
   extends java.lang.Cloneable,
           java.util.Map,
           java.io.Serializable
{
}

//end ParameterHandleValueMap
// File: ParameterHandleValueMapFactory.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
* Interface corresponding to the {@link RTIambassador#getParameterHandleValueMapFactory
getParameterHandleValueMapFactory} method.
* Its one method simply creates a new {@link ParameterHandleValueMap} object.
* @author IEEE
* @version 1516.1.5 (DoD v2)
*/
public interface ParameterHandleValueMapFactory
    extends java.io.Serializable
{
    /**
     * Creates a new {@link ParameterHandleValueMap} instance with the
     * specified initial capacity.
     * @param capacity Initial capacity (number of keys) of the
     * ParameterHandleValueMap</code>
     * @return A newly created ParameterHandleValueMap</code>
     */
    public ParameterHandleValueMap create(int capacity);
}
//end ParameterHandleValueMapFactory
package hla.rti1516; //the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * Record returned by the (10.31) \{@link RTIambassador#getRangeBounds
getRangeBounds\} method
 * and supplied to its (10.32) \{@link RTIambassador#setRangeBounds
setRangeBounds\} method.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 adds the
java.io.Serializable implementation and a constructor.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public final class
RangeBounds
    implements java.io.Serializable
{
    /**
    * Lower (included) bound of the range.
    */
    public long lower;

    /**
    * Upper (excluded) bound of the range.
    */
    public long upper;

    /**
    * Public constructor.
    * @param l The range’s lower bound (included)
    * @param u The range’s upper bound (excluded)
    */
    public RangeBounds(long l,
            long u)
    {
        lower = l;
        upper = u;
    }
/**
 * Returns true iff <code>this</code> and <code>other</code> represent the same set of bounds.
 * @param other The <code>Object</code> to compare with
 * @return true iff supplied <code>other</code> is of type <code>RangeBounds</code> and has same values
 */
public boolean equals(Object other)
{
    if ((other != null) && (other instanceof RangeBounds))
    {
        RangeBounds otherRangeBounds = (RangeBounds)other;
        return (lower == otherRangeBounds.lower) && (upper == otherRangeBounds.upper);
    }
    else
    {
        return false;
    }
}

/**
 * Returns a hash code for <code>this</code>; two <code>RangeBounds</code> for which <code>equals()</code> is true should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode()
{
    return (int) (lower + upper);
}
} //end RangeBounds
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * A designator for an RTI <code>Region</code> object.
 * <p>
 * A <b>region specification</b> is a set of <code>Range</code>s, each one of which is a continuous semi-open interval (defined by an included lower bound and an excluded upper bound) on a different <code>Dimension</code>.  
 * A <b>region realization</b> is a region specification that is associated with an instance attribute for update, with a sent interaction, or with a class attribute or interaction class for subscription.
 * The specification and realization may differ when a default range is specified for a dimension in the FOM Document Data (FDD), since if that dimension is not mentioned in the region specification, its default range is implicitly added to the region realization.
 * <p>
 * Region specifications are created, committed and deleted by the (9.2) {@link RTIambassador#createRegion createRegion}, (9.3) {@link RTIambassador#commitRegionModifications commitRegionModifications} and (9.4) {@link RTIambassador#deleteRegion deleteRegion} methods.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public interface RegionHandle
    extends java.io.Serializable
{
    /**
     * Returns a <code>String</code> representation of the <code>RegionHandle</code>.
     * @return A {@link java.lang.String} with value reflecting <code>this</code>
     */
    public String toString();

    /**
     * Returns true iff <code>this</code> and <code>otherRegionHandle</code> refer to the same Region.
     * @param otherRegionHandle The <code>Object</code> to compare with
     * @return true iff supplied <code>otherRegionHandle</code> is of type <code>RegionHandle</code> and has same value
     */
    public boolean equals(Object otherRegionHandle);
/**
 * Returns a hash code for <code>this</code>; two
 * <code>RegionHandle</code>s for which <code>equals()</code> is
 * <code>true</code> should yield the same hash code
 * (because they then refer to the same Region).
 * @return An <code>int</code> hash code
 */
public int
hashCode();

//end RegionHandle
package hla.rti1516; //the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * A Set of {@link RegionHandle}s.
 * All {@link java.util.Set} operations are required, none are
 * optional.
 * <code>add()</code> and <code>remove()</code> should throw
 * {@link IllegalArgumentException} if the argument is not a
 * <code>RegionHandle</code>.
 * <code>addAll()</code>, <code>removeAll()</code> and
 * <code>retainAll()</code> should throw
 * {@link IllegalArgumentException} if
 * the argument is not a <code>RegionHandleSet</code>.
 * <p>
 * Used with the (9.3) {link RTIambassador#commitRegionModifications
 * commitRegionModifications},
 * (9.10/9.11) {link
 * RTIambassador#unsubscribeInteractionClassWithRegions [un]} {link
 * RTIambassador#subscribeInteractionClassWithRegions
 * subscribeInteractionClassWithRegions} {link
 * RTIambassador#subscribeInteractionClassPassivelyWithRegions
 * subscribeInteractionClassPassivelyWithRegions}
 * and
 * (9.12) {link RTIambassador#sendInteractionWithRegions
 * sendInteractionWithRegions} (both forms) methods.
 * <p>
 * Also received by certain forms of the (6.7) {link
 * FederateAmbassador#reflectAttributeValue(ObjectInstanceHandle,
 * AttributeHandleValueMap,byte[],OrderType,TransportationType,
 * RegionHandleSet)}
 * and (6.9) {link
 * FederateAmbassador#receiveInteraction(InteractionClassHandle,
 * ParameterHandleValueMap,byte[],OrderType,TransportationType,
 * RegionHandleSet)}
 * callbacks if the Convey Region Designator Sets Switch is turned on
 * (this is done through the
 * <code>HLAmanager.HLAfederate.HLAadjust.HLAsetSwitches</code>
 * Management Object Model (MOM) interaction: there is no special API for
 * it).
 * These conveyed sets of <code>RegionHandle</code>s can be used with the
 * (10.31) {link RTIambassador#getRangeBounds getRangeBounds} and
 * (10.30) {link RTIambassador#getDimensionHandleSet
getDimensionHandleSet} methods.
 * All other RTIambassador <code>RegionHandle</code> methods expect a
 * federate-owned <code>Region</code>.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see RegionHandleSetFactory
 */
public interface RegionHandleSet
    extends java.lang.Cloneable,
            java.io.Serializable,
            java.util.Set
{
}
package hla.rti1516;  //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
   * Interface corresponding to the {@link RTIambassador#getRegionHandleSetFactory getRegionHandleSetFactory}
   * method.
   * Its one method simply creates a new {@link RegionHandleSet} object.
   * @author IEEE
   * @version 1516.1.5 (DoD v2)
   */
public interface RegionHandleSetFactory

   extends java.io.Serializable
{
   /**
      * Creates a new {@link RegionHandleSet} instance.
      * @return A newly created <code>RegionHandleSet</code>
      */
   public RegionHandleSet create();
}
//end RegionHandleSetFactory
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java {@link java.util.Enumeration})!
 * representing the resign policy adopted by the federate.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see hla.rti1516.RTIambassador#resignFederationExecution
 */
public final class ResignAction
    implements java.io.Serializable
{
    //each instance's value
    private int _value;
    //initial value for enumeration
    private static final int _lowestValue = 1;
    //begins at lowest
    private static int _nextToAssign = _lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a ResignAction must be initialized
     * with one of the defined static values.
     * @param otherResignActionValue must be a defined static value or
     * another instance.
     */
    public ResignAction(ResignAction otherResignActionValue)
    {
        _value = otherResignActionValue._value;
    }

    /**
     * Package-only (default access) constructor. Unused.
     * @param value to assign to the instance; must be one of the
     * static ones
     */
    ResignAction(int value)
        throws RTIinternalError
    {
        _value = value;
        if ((_value < _lowestValue) || (_value >= _nextToAssign))
            throw new RTIinternalError("ResignAction: illegal value " +
                                  value);
    }
}
/**
 * Private constructor; it is used to generate the static values.
 */
private ResignAction()
{
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the <code>ResignAction</code>.  
 * @return A <tt>{@link java.lang.String}</tt> with value "ResignAction(n)" where n is <code>this</code> value
 */
public String toString()
{
    return "ResignAction(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and <code>otherResignActionValue</code> represent the same resign action.
 * @param otherResignActionValue The <code>Object</code> to compare with
 * @return true iff supplied <code>otherResignActionValue</code> is of type <code>ResignAction</code> and has same value
 */
public boolean equals(Object otherResignActionValue)
{
    if (otherResignActionValue instanceof ResignAction)
        return _value == ((ResignAction)otherResignActionValue)._value;
    else
        return false;
}

/**
 * Returns a hash code for <code>this</code>; two <code>ResignAction</code>'s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode()
{
    return _value;
}
// The constant instances
/**
 * Unconditionally divest ownership of all owned instance attributes.
 */
static public final ResignAction UNCONDITIONALLY_DIVEST_ATTRIBUTES = new ResignAction();

/**
 * Delete all object instances for which the joined federate has the delete privilege.
 */
static public final ResignAction DELETE_OBJECTS = new ResignAction();

/**
 * Cancel all pending instance attribute ownership acquisitions.
 */
static public final ResignAction CANCEL.Pending.OWNERSHIP.ACQUISITIONS = new ResignAction();

/**
 * Perform DELETE_OBJECTS and then UNCONDITIONALLY_DIVEST_ATTRIBUTES.
 */
static public final ResignAction DELETE_OBJECTS_THEN_DIVEST = new ResignAction();

/**
 * Perform CANCEL_PENDING.OWNERSHIP.ACQUISITIONS, then DELETE_OBJECTS and then UNCONDITIONALLY_DIVEST_ATTRIBUTES.
 */
static public final ResignAction CANCEL_THEN_DELETE_THEN_DIVEST = new ResignAction();

/**
 * Perform no actions.
 */
static public final ResignAction NO_ACTION = new ResignAction();
}
package hla.rtiI516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java {@link java.util Enumeration}) representing the reason why the restore operation of a federate failed.
 * It is reported by the {@link FederateAmbassador#federationNotRestored federationNotRestored} callback.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see hla.rtiI516.FederateAmbassador#federationNotRestored federationNotRestored */

public final class RestoreFailureReason
    implements java.io.Serializable
{
    //each instance's value
    private int _value;
    //initial value for enumeration
    private static final int _lowestValue = 1;
    //begins at lowest
    private static int _nextToAssign = _lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a RestoreFailureReason must be initialized with one of the defined static values.
     * <p>
     * Here we change the parameter name from "otherResignActionValue" to "reason";
     * this is similar to the change DoD Interpretations of IEEE 1516-2000v2 applies
     * to the {@link SynchronizationPointFailureReason} constructor.
     * @param reason must be a defined static value or another instance.
     */
    public RestoreFailureReason(RestoreFailureReason reason) {
        _value = reason._value;
    }
}
/**
 * Package-only (default access) constructor. Unused.
 * @param value to assign to the instance; must be one of the
 * static ones
 */
RestorableFailureReason(int value)
  throws RTIinternalError
{
  _value = value;
  if ((value < _lowestValue) || (value >= _nextToAssign))
    throw new RTIinternalError("RestorableFailureReason: illegal value + value");
}

/**
 * Private constructor; it is used to generate the static values.
 */
private
RestorableFailureReason()
{
  _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the
 * <code>RestorableFailureReason</code>.  
 * @return A <code>String</code> with value 
 * "RestorableFailureReason(n)" where n is <code>this</code> value
 */
public String
toString()
{
  return "RestorableFailureReason(" + _value + ")";
}
/**
 * Returns true iff \texttt{this} and \texttt{otherRestoreFailureReasonValue} represent the same restore failure reason.
 * <p>
 * Here we change the parameter name from \texttt{otherResignActionValue} to \texttt{otherRestoreFailureReasonValue};
 * this follows the same rationale as the change DoD Interpretations of IEEE 1516-2000v2 applies
 * to the \texttt{@link SynchronizationPointFailureReason} constructor.
 * @param otherRestoreFailureReasonValue The \texttt{Object} to compare with
 * @return true iff supplied \texttt{otherRestoreFailureReasonValue} is of type \texttt{RestoreFailureReason} and has same value
 */
 public boolean equals (Object otherRestoreFailureReasonValue)
 {
    if (otherRestoreFailureReasonValue instanceof RestoreFailureReason)
       return value == ((RestoreFailureReason)otherRestoreFailureReasonValue)._value;
    else
       return false;
 }

/**
 * Returns a hash code for \texttt{this}; two \texttt{RestoreFailureReason}s for which \texttt{equals()} is \texttt{true} should yield the same hash code.
 * @return An \texttt{int} hash code
 */
 public int hashCode()
 {
    return _value;
 }
The constant instances

### The RTI was unable to restore.
```java
static public final RestoreFailureReason RTI_UNABLE_TO_RESTORE = new RestoreFailureReason();
```

### One or more federates have invoked the `federateRestoreNotComplete` method.
```java
static public final RestoreFailureReason FEDERATE_REPORTED_FAILURE = new RestoreFailureReason();
```

### One or more joined federates have resigned from the federation execution.
```java
static public final RestoreFailureReason FEDERATE_RESIGNED = new RestoreFailureReason();
```

### The RTI has detected failure at one or more of the joined federates.
```java
static public final RestoreFailureReason RTI_DETECTED_FAILURE = new RestoreFailureReason();
```
public final class RestoreStatus implements java.io.Serializable {

  // each instance's value
  private int _value;
  // initial value for enumeration
  private static final int _lowestValue = 1;
  // begins at lowest
  private static int _nextToAssign = _lowestValue;

  /**
   * This is the only public constructor.
   * Each user-defined instance of a RestoreStatus must be initialized with one of the defined static values.
   * @param otherRestoreStatusValue must be a defined static value or another instance.
   */
  public RestoreStatus(RestoreStatus otherRestoreStatusValue) {
    _value = otherRestoreStatusValue._value;
  }

  /**
   * Package-only (default access) constructor. Unused.
   * @param value to assign to the instance; must be one of the static ones
   */
  RestoreStatus(int value) throws RTIinternalError {
    _value = value;
    if ((value < _lowestValue) || (value >= _nextToAssign))
      throw new RTIinternalError("RestoreStatus: illegal value " + value);
  }
}
/**
 * Private constructor; it is used to generate the static values.
 */
private RestoreStatus()
{
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the <code>RestoreStatus</code>.
 * @return A &lt;code&gt;String</code> with value "RestoreStatus(n)" where n is <code>this</code> value
 */
public String toString()
{
    return "RestoreStatus(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and <code>otherRestoreStatusValue</code> represent the same restore status.
 * @param otherRestoreStatusValue The <code>Object</code> to compare with
 * @return true iff supplied <code>other</code> is of type <code>RestoreStatus</code> and has same value
 */
public boolean equals(Object otherRestoreStatusValue)
{
    if (otherRestoreStatusValue instanceof RestoreStatus)
        return value == ((RestoreStatus)otherRestoreStatusValue).value;
    else
        return false;
}

/**
 * Returns a hash code for <code>this</code>; two <code>RestoreStatus</code>s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode()
{
    return _value;
}
/The constant instances
/**
 * No restore in progress (federate in the Active state).
 */
static public final RestoreStatus
NO_RESTORE_IN_PROGRESS = new RestoreStatus();

/**
 * Federate in the Restore Request Pending state.
 */
static public final RestoreStatus
FEDERATE_RESTORE_REQUEST_PENDING = new RestoreStatus();

/**
 * Federate in the Waiting For Restore To Begin state.
 */
static public final RestoreStatus
FEDERATE_WAITING_FOR_RESTORE_TO_BEGIN = new RestoreStatus();

/**
 * Federate in the Prepared To Restore state.
 */
static public final RestoreStatus
FEDERATE_PREPARED_TO_RESTORE = new RestoreStatus();

/**
 * Federate in the Restoring state.
 */
static public final RestoreStatus
FEDERATE_RESTORING = new RestoreStatus();

/**
 * Federate in the Waiting For Federation To Restore state.
 */
static public final RestoreStatus
FEDERATE_WAITING_FOR_FEDERATION_TO_RESTORE = new RestoreStatus();
} //end RestoreStatus
package hla.rti1516;  //the package name was changed by DoD

/**
 * Memory Management Conventions for Parameters
 *
 * All Java parameters, including object references, are passed by value.
 * Therefore there is no need to specify further conventions for primitive types.
 * Unless otherwise noted, reference parameters adhere to the following convention:
 * The referenced object is created (or acquired) by the caller. The callee must
 * copy during the call anything it wishes to save beyond the completion of the
 * call.
 * Unless otherwise noted, a reference returned from a method represents a new
 * object created by the callee. The caller is free to modify the object whose
 * reference is returned.
 */
/**
 * The RTI presents this interface to the federate.
 * The RTI implementer must implement this.
 * <p>
 * As of DoD Interpretations of IEEE 1516-2000v2, none of the
 * RTIambassador methods may be called
 * with a <code>null</code> argument, with two exceptions: user-
 * supplied <code>tag</code> arguments may be <code>null</code>,
 * and the {link MobileFederateServices} argument of the {link
 * #joinFederationExecution} service
 * may be <code>null</code> (in the case where the federate does not
 * wish to supply the {link LogicalTimeFactory} and {link
 * LogicalTimeIntervalFactory}).
 * With the noted exceptions, if a <code>null</code> argument is
 * supplied, the RTI will throw a {link java.lang.NullPointerException}.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public interface RTIambassador
{
  // 4.2
  /**
   * Creates a federation execution.
   * @param federationExecutionName a {link java.lang.String}
   * holding the federation execution's identifier
   * @param fdd a {link java.net.URL} giving the path to the
   * federation execution's FOM Document Data (FDD)
   * @throws FederationExecutionAlreadyExists if the specified
   * federation execution already exists
   * @throws CouldNotOpenFDD if the FDD could not be found or opened
   * @throws ErrorReadingFDD if the FDD is corrupt or otherwise
   * unusable
   * @throws RTIinternalError if something else goes wrong
   * @see #destroyFederationExecution destroyFederationExecution
   */
  public void createFederationExecution(
      String federationExecutionName,
      java.net.URL fdd)
  throws FederationExecutionAlreadyExists,
      CouldNotOpenFDD,
      ErrorReadingFDD,
      RTIinternalError;
public void destroyFederationExecution(String federationExecutionName)
        throws FederatesCurrentlyJoined, FederationExecutionDoesNotExist, RTIinternalError;
// 4.4
/**
 * Joins the federate to the federation execution.
 * When a federate is instructed to save its state, the federate-specific persistent storage must use
 * the save [link FederationSave(String) label], the
 * [code]federateType[/code] supplied to this method and the [link FederateHandle] returned
 * by this method.
 * When the federation is later restored, there must be the same
 * number of federates of each
 * [code]federateType[/code] joined.
 * @param federateType a [link java.lang.String] descriptor used
 * to distinguish federate categories for federation save-and-restore
 * purposes
 * @param federationExecutionName a <code>String</code> giving the
 * federation execution's identifier
 * @param federateReference the [link FederateAmbassador]
 * interface of the joining federate
 * @param serviceReferences a [link MobileFederateServices] record
 * holding the federate-supplied [link LogicalTimeFactory] and [link LogicalTimeIntervalFactory] implementations (this parameter may be
 * <code>null</code>)
 * @return the joined federate's [link FederateHandle]
 * @throws FederationAlreadyExecutingMember if the federate (the
 * [link RTIambassador] instance) has already joined a federation
 * execution
 * @throws FederationExecutionDoesNotExist if the specified
 * federation execution does not exist
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #resignFederationExecution resignFederationExecution
 */
public FederateHandle
joinFederationExecution(String federateType, String federationExecutionName, FederateAmbassador federateReference, MobileFederateServices serviceReferences) throws FederationAlreadyExecutingMember, FederationExecutionDoesNotExist, SaveInProgress, RestoreInProgress, RTIinternalError;
/**
 * Resigns the federate from the federation execution.
 * <p>
 * The possible resignation policies are:
 * <ul>
 * <li> UNCONDITIONALLY_DIVEST_ATTRIBUTES: Unconditionally divest ownership of all owned instance attributes
 * <li> DELETE_OBJECTS: Delete all object instances for which the joined federate has the delete privilege
 * <li> CANCEL_PENDING_OWNERSHIP_ACQUISITIONS: Cancel all pending instance attribute ownership acquisitions
 * <li> DELETE_OBJECTS_THEN_DIVEST: Perform DELETE_OBJECTS and then UNCONDITIONALLY_DIVEST_ATTRIBUTES
 * <li> CANCEL_THEN_DELETE_THEN_DIVEST: Perform CANCEL_PENDING_OWNERSHIP_ACQUISITIONS, then DELETE_OBJECTS and then UNCONDITIONALLY_DIVEST_ATTRIBUTES
 * <li> NO_ACTION: Perform no actions
 * </ul>
 * When UNCONDITIONALLY_DIVEST_ATTRIBUTES occurs, the RTI will try to transfer ownership to any eligible federates (those that have discovered the object instances); this means ownership may be granted outright to federates that were already in the "Acquiring" or "Willing To Acquire" states, and that requestAttributeOwnershipAssumption callbacks may be sent as a consequence of this federate's resignation and of later object instance discoveries.
 * @param resignAction a {@link ResignAction} representing the federate's resignation policy
 * @throws OwnershipAcquisitionPending if there is an ownership acquisition pending for some object class (and the federate isn't cancelling them)
 * @throws FederateOwnsAttributes if the federate owns some instance attributes (and isn't deleting or divesting them)
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #joinFederationExecution
 */
public void resignFederationExecution(ResignAction resignAction)
    throws OwnershipAcquisitionPending,
            FederateOwnsAttributes,
            FederateNotExecutionMember,
            RTIinternalError;
4.6

** Registers a federation synchronization point. This form concerns all of the currently joined federates.

* Synchronization point labels may be pre-defined in the federation execution's FOM Document Data (FDD) but this isn't a requirement.

* @param synchronizationPointLabel a {@link java.lang.String} holding the synchronization point's identifier
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #synchronizationPointAchieved synchronizationPointAchieved
* @see FederateAmbassador#synchronizationPointRegistrationSucceeded
* @see FederateAmbassador#synchronizationPointRegistrationFailed
* @see FederateAmbassador#announceSynchronizationPoint
* @see FederateAmbassador#federationSynchronized
* /

public void registerFederationSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag)
    throws FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
/**
 * Registers a federation synchronization point and specifies a subset of the currently joined federates.
 * @param synchronizationPointLabel a String holding the synchronization point's identifier
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param synchronizationSet a FederateHandleSet holding the FederateHandle(s) of the federates concerned by the synchronization point
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #synchronizationPointAchieved synchronizationPointAchieved
 * @see FederateAmbassador#synchronizationPointRegistrationSucceeded synchronizationPointRegistrationSucceeded
 * @see FederateAmbassador#synchronizationPointRegistrationFailed synchronizationPointRegistrationFailed
 * @see FederateAmbassador#announceSynchronizationPoint announceSynchronizationPoint
 * @see FederateAmbassador#federationSynchronized federationSynchronized
 */

public void registerFederationSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag,
    FederateHandleSet synchronizationSet)
    throws FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
/**
 * Reports to the RTI that the specified synchronization point has been achieved.
 * @param synchronizationPointLabel a String holding the synchronization point's identifier
 * @throws SynchronizationPointLabelNotAnnounced if the RTI does not recognize the specified synchronizationPointLabel
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #registerFederationSynchronizationPoint
 * @see FederateAmbassador#synchronizationPointRegistrationSucceeded
 * @see FederateAmbassador#synchronizationPointRegistrationFailed
 * @see FederateAmbassador#announceSynchronizationPoint
 * @see FederateAmbassador#federationSynchronized
 */

public void synchronizationPointAchieved(String synchronizationPointLabel)
    throws SynchronizationPointLabelNotAnnounced,
               FederateNotExecutionMember,
               SaveInProgress,
               RestoreInProgress,
               RTIinternalError;
public void requestFederationSave(String label)
throws FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
public void requestFederationSave(String label, LogicalTime theTime)
    throws LogicalTimeAlreadyPassed, InvalidLogicalTime,
             FederateUnableToUseTime, FederateNotExecutionMember,
             SaveInProgress, RestoreInProgress, RTIinternalError;
// 4.13
/**
 * Signals the RTI that this federate has begun to save its state.
 * @throws SaveNotInitiated if a federation save was not previously
 * requested
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationSave requestFederationSave
 * @see #federateSaveComplete federateSaveComplete
 * @see #federateSaveNotComplete federateSaveNotComplete
 * @see #queryFederationSaveStatus queryFederationSaveStatus
 * @see FederateAmbassador#initiateFederateSave
 * @see FederateAmbassador#federationSaved federationSaved
 * @see FederateAmbassador#federationNotSaved federationNotSaved
 * @see FederateAmbassador#federationSaveStatusResponse
 */
public void federateSaveBegun()
throws SaveNotInitiated,
       FederateNotExecutionMember,
       RestoreInProgress,
       RTIinternalError;

// 4.14
/**
 * Signals the RTI that this federate has finished saving its
 * state.
 * @throws FederateHasNotBegunSave if the federate has not
 * previously invoked {link #federateSaveBegun federateSaveBegun}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationSave requestFederationSave
 * @see #federateSaveBegun federateSaveBegun
 * @see #federateSaveNotComplete federateSaveNotComplete
 * @see #queryFederationSaveStatus queryFederationSaveStatus
 * @see FederateAmbassador#initiateFederateSave
 * @see FederateAmbassador#federationSaved federationSaved
 * @see FederateAmbassador#federationNotSaved federationNotSaved
 * @see FederateAmbassador#federationSaveStatusResponse
 */
public void federateSaveComplete()
throws FederateHasNotBegunSave,
       FederateNotExecutionMember,
       RestoreInProgress,
       RTIinternalError;
/**
 * Signals the RTI that this federate was unable to save its state.
 * @throws FederateHasNotBegunSave if the federate has not
 * previously invoked [link #federateSaveBegun federateSaveBegun]
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationSave requestFederationSave
 * @see #federateSaveBegun federateSaveBegun
 * @see #federateSaveComplete federateSaveComplete
 * @see #queryFederationSaveStatus queryFederationSaveStatus
 * @see FederateAmbassador#initiateFederateSave
 * @see FederateAmbassador#federationSaved federationSaved
 * @see FederateAmbassador#federationNotSaved federationNotSaved
 * @see FederateAmbassador#federationSaveStatusResponse
 */
public void federateSaveNotComplete()
    throws FederateHasNotBegunSave,
            FederateNotExecutionMember,
            RestoreInProgress,
            RTIinternalError;

// 4.16
/**
 * Requests that the federation report on the status of the current
 * save.
 * @p
 * In the DoD Interpretations of IEEE 1516-2000v2, the pre-
 * condition "Save in progress" is dropped, and the exception
 * <code>SaveNotInProgress</code> is also deleted.
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationSave requestFederationSave
 * @see #federateSaveBegun federateSaveBegun
 * @see #federateSaveComplete federateSaveComplete
 * @see #federateSaveNotComplete federateSaveNotComplete
 * @see FederateAmbassador#initiateFederateSave
 * @see FederateAmbassador#federationSaved federationSaved
 * @see FederateAmbassador#federationNotSaved federationNotSaved
 * @see FederateAmbassador#federationSaveStatusResponse
 */
public void queryFederationSaveStatus()
    throws FederateNotExecutionMember,
            FederateNotExecutionMember,
            RestoreInProgress,
            RTIinternalError;
// 4.18
/**
 * Requests that the federation restore its current state to that
 * saved under the specified label.
 * <p>
 * The federate-specific persistent storage of a federation saved
 * state must use the save
 * <code>label</code>, the <code>federateType</code> supplied to
 * the {@link #joinFederationExecution joinFederationExecution}
 * method and the {@link FederateHandle} returned by the same
 * method.
 * For the restoration request to succeed, there must be the same
 * number of federates of
 * each <a href="#joinFederationExecution federateType">federateType</a> currently
 * joined.
 * Declaring a federate to be of a given <code>federateType</code>
 * is therefore equivalent
 * to asserting that it can be restored using the state information
 * saved by any other
 * federate of that <code>federateType</code>.
 * <p>
 * The RTI saves its own RTI-specific state information when a
 * federation save succeeds;
 * this information tracks the save <code>label</code>, the <a href="#joinFederationExecution federationExecutionName">federationExecutionName</a>,
 * the <a href="#createFederationExecution fdd">fdd</a> and the census of
 * joined federates
 * in number and <code>federateType</code>.
 * <p>
 * There is no requirement that a save taken by one RTI
 * implementation be restorable by another.
public void requestFederationRestore(String label)
    throws FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
/**
 * Signals the RTI that this federate has completed its restore operation.
 * @throws RestoreNotRequested if a federation restore was not previously requested
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationRestore requestFederationRestore
 * @see #federateRestoreNotComplete federateRestoreNotComplete
 * @see #queryFederationRestoreStatus queryFederationRestoreStatus
 * @see FederateAmbassador#requestFederationRestoreSucceeded requestFederationRestoreSucceeded
 * @see FederateAmbassador#requestFederationRestoreFailed requestFederationRestoreFailed
 * @see FederateAmbassador#federationRestoreBegun federationRestoreBegun
 * @see FederateAmbassador#initiateFederateRestore initiateFederateRestore
 * @see FederateAmbassador#federationRestored federationRestored
 * @see FederateAmbassador#federationNotRestored federationNotRestored
 * @see FederateAmbassador#federationRestoreStatusResponse federationRestoreStatusResponse
 */

public void federateRestoreComplete() throws RestoreNotRequested, FederateNotExecutionMember, SaveInProgress, RTIinternalError;
/**
 * Signals the RTI that this federate has failed in its restore
 * operation.
 * @throws RestoreNotRequested if a federation restore was not
 * previously requested
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestFederationRestore requestFederationRestore
 * @see #federateRestoreComplete federateRestoreComplete
 * @see #queryFederationRestoreStatus queryFederationRestoreStatus
 * @see FederateAmbassador#requestFederationRestoreSucceeded
 * @see FederateAmbassador#requestFederationRestoreFailed
 * @see FederateAmbassador#federationRestoreBegun
 * @see #initiateFederateRestore
 * @see #federationRestored federationRestored
 * @see #federationNotRestored federationNotRestored
 * @see #federationRestoreStatusResponse
 */

public void
federateRestoreNotComplete()
throws RestoreNotRequested,
       FederateNotExecutionMember,
       SaveInProgress,
       RTIinternalError;
Requests that the federation report on the status of the current restore operation.

In the DoD Interpretations of IEEE 1516-2000v2, the precondition "Restore in progress" is dropped, and the exception RestoreNotInProgress is also deleted.

The FederateHandle{s} used in the FederateHandleRestoreStatusPair[] returned by the federationRestoreStatusResponse callback are the pre-restore ones until all federates have invoked the RTIambassador#{federateRestoreComplete federateRestoreComplete} service. Once all federates have been issued the FederateAmbassador#{federationRestored federationRestored} callbacks, the post-restore FederateHandle{s} are used (and each federate's status will be RestoreStatus NO_RESTORE_IN_PROGRESS).

If this service is invoked between those two times, the FederateHandle{s} are unpredictable. A federate should therefore avoid invoking this service between the moment it invokes the federateRestoreComplete service and the moment it receives the federationRestored service or the federateNotRestored callbacks.

@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
@throws SaveInProgress if the federate is in one of the save-in-progress states
@throws RTIinternalError if something else goes wrong
@see #requestFederationRestore requestFederationRestore
@see #federateRestoreComplete federateRestoreComplete
@see #federateRestoreNotComplete federateRestoreNotComplete
@see FederateAmbassador#{requestFederationRestoreSucceeded requestFederationRestoreSucceeded}
@see FederateAmbassador#{requestFederationRestoreFailed requestFederationRestoreFailed}
@see FederateAmbassador#{federationRestoreBegun federationRestoreBegun}
@see FederateAmbassador#{initiateFederateRestore initiateFederateRestore}
@see FederateAmbassador#{federationRestored federationRestored}
@see FederateAmbassador#{federationNotRestored federationNotRestored}
@see FederateAmbassador#{federationRestoreStatusResponse federationRestoreStatusResponse}

public void queryFederationRestoreStatus() throws FederateNotExecutionMember, SaveInProgress, RTIinternalError;
public void publishObjectClassAttributes(
    ObjectClassHandle theClass,
    AttributeHandleSet attributeList)
    throws ObjectClassNotDefined,
            AttributeNotDefined,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
Signals to the RTI this federate’s intention to stop publishing any attributes of an object class.
* Note that this does <i>not</i> trigger [requestAttributeOwnershipAssumption](#)
* callbacks for object instance attributes that become unowned.
* @param theClass the [ObjectClassHandle] of the unpublished object class
* @throws ObjectClassNotDefined if <code>theClass</code> isn’t recognized by the RTI
* @throws OwnershipAcquisitionPending if there is an ownership acquisition pending for the specified object class
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #publishObjectClassAttributes publishObjectClassAttributes
* @see #unpublishObjectClassAttributes publishObjectClassAttributes
/*
public void unpublishObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotDefined,
    OwnershipAcquisitionPending,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Signals to the RTI this federate's intention to stop publishing certain attributes of an object class.
 * Note that this does <i>not</i> trigger [{@link FederateAmbassador#requestAttributeOwnershipAssumption requestAttributeOwnershipAssumption}]
 * callbacks for object instance attributes that become unowned.
 * @param theClass the {@link ObjectClassHandle} of the unpublished object class
 * @param attributeList an '{@link AttributeHandleSet} listing the unpublished attributes
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws OwnershipAcquisitionPending if there is an ownership acquisition pending for the specified object class attributes
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #publishObjectClassAttributes publishObjectClassAttributes
 * @see #unpublishObjectClass unpublishObjectClass
 */

public void unpublishObjectClassAttributes(
    ObjectClassHandle theClass,
    AttributeHandleSet attributeList)
throws ObjectClassNotDefined,
        AttributeNotDefined,
        OwnershipAcquisitionPending,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
public void publishInteractionClass(InteractionClassHandle theInteraction)
throws InteractionClassNotDefined,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIInternalError;

public void unpublishInteractionClass(InteractionClassHandle theInteraction)
throws InteractionClassNotDefined,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIInternalError;
// 5.6
/**
 * Subscribes the federate to certain attributes of an object class.
 * @param theClass the {@link ObjectClassHandle} of the subscribed object class
 * @param attributeList an {@link AttributeHandleSet} listing the subscribed attributes
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see subscribeObjectClassAttributesPassively
 * @see subscribeObjectClassAttributesWithRegions
 * @see unsubscribeObjectClass
 * @see unsubscribeObjectClassAttributes
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 * @see FederateAmbassador#attributesInScope
 * @see FederateAmbassador#attributesOutOfScope
 */
public void subscribeObjectClassAttributes(
    ObjectClassHandle theClass,
    AttributeHandleSet attributeList)
    throws ObjectClassNotDefined,
    AttributeNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
}
/**
 * Subscribes the federate to certain attributes of an object class without tripping the publishers' object class relevance advisories.
 * <p>
 * Note that if this federate uses this service to switch its subscription mode from active to passive and it happened to be the last active subscriber, the publisher may receive a
 * [/link FederateAmbassador#stopRegistrationForObjectClass stopRegistrationForObjectClass] callback (and possibly one or more [link FederateAmbassador#turnUpdatesOffForObjectInstance turnUpdatesOffForObjectInstance] callbacks).
 * @param theClass the [link ObjectClassHandle] of the subscribed object class
 * @param attributeList an [link AttributeHandleSet] listing the subscribed attributes
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeObjectClassAttributes subscribeObjectClassAttributes
 * @see #subscribeObjectClassAttributesWithRegions subscribeObjectClassAttributesWithRegions
 * @see #subscribeObjectClassAttributesPassivelyWithRegions subscribeObjectClassAttributesPassivelyWithRegions
 * @see #unsubscribeObjectClass unsubscribeObjectClass
 * @see #unsubscribeObjectClassAttributes unsubscribeObjectClassAttributes
 * @see #unsubscribeObjectClassAttributesWithRegions unsubscribeObjectClassAttributesWithRegions
 * @see #unsubscribeObjectClassAttributesPassivelyWithRegions unsubscribeObjectClassAttributesPassivelyWithRegions
 */

public void subscribeObjectClassAttributesPassively(
    ObjectClassHandle theClass,
    AttributeHandleSet attributeList)
    throws ObjectClassNotDefined,
            AttributeNotDefined,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;

DRDC Valcartier TR 2007-412 271
// 5.7
/**
 * Unsubscribes the federate from any attributes of an object class.
 * @param theClass the {@link ObjectClassHandle} of the unsubscribed object class
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeObjectClassAttributes
 * @see #subscribeObjectClassAttributesPassively
 * @see #subscribeObjectClassAttributesWithRegions
 * @see #subscribeObjectClassAttributesPassivelyWithRegions
 * @see #unsubscribeObjectClassAttributes
 * @see #unsubscribeObjectClassAttributesWithRegions
 */
public void unsubscribeObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Unsubscribes the federate from certain attributes of an object class.
 * @param theClass the {@link ObjectClassHandle} of the unsubscribed object
 * class
 * @param attributeList an {@link AttributeHandleSet} listing the unsubscribed attributes
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see subscribeObjectClassAttributes
 * @see subscribeObjectClassAttributesPassively
 * @see subscribeObjectClassAttributesWithRegions
 * @see subscribeObjectClassAttributesPassivelyWithRegions
 * @see unsubscribeObjectClass
 * @see unsubscribeObjectClassAttributesWithRegions
 */

public void unsubscribeObjectClassAttributes(
        ObjectClassHandle theClass,
        AttributeHandleSet attributeList)
throws ObjectClassNotDefined,
        AttributeNotDefined,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;

public void subscribeInteractionClass(InteractionClassHandle theClass) throws InteractionClassNotDefined, FederateServiceInvocationsAreBeingReportedViaMOM, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
/**
 * Subscribes the federate to an interaction without tripping the
 * publishers' interaction relevance advisories.
 * <p>
 * Note that if this federate uses this service to switch its
 * subscription mode from active to passive and it
 * happened to be the last active subscriber, the publisher may
 * receive a
 * <code>{@link FederateAmbassador#turnInteractionsOff turnInteractionsOff}</code> callback.
 * @param theClass the <code>@link InteractionClassHandle</code> of the
 * subscribed interaction
 * @throws InteractionClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws FederateServiceInvocationsAreBeingReportedViaMOM if service invocations are currently being reported via MOM interactions
 * and <code>theClass</code> is
 * <code>HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation</code>
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeInteractionClass subscribeInteractionClass
 * @see #subscribeInteractionClassWithRegions subscribeInteractionClassWithRegions
 * @see #subscribeInteractionClassPassivelyWithRegions subscribeInteractionClassPassivelyWithRegions
 * @see #unsubscribeInteractionClass unsubscribeInteractionClass
 * @see FederateAmbassador#receiveInteraction receiveInteraction
 */
public void subscribeInteractionClassPassively(
    InteractionClassHandle theClass)
throws InteractionClassNotDefined,
    FederateServiceInvocationsAreBeingReportedViaMOM,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
// 5.9
/**
 * Unsubscribes the federate from an interaction.
 * @param theClass the {@link InteractionClassHandle} of the unsubscribed interaction
 * @throws InteractionClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeInteractionClass subscribeInteractionClass
 */
public void unsubscribeInteractionClass(InteractionClassHandle theClass)
    throws InteractionClassNotDefined, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
6.2

/**
 * Reserves a federation-unique name for a future object instance registration.
 * <p>
 * Note that although only one federate can successfully reserve an object instance name,
 * the RTI does not prevent another federate from being the first to use it during registration.
 * The Pitch pRTI1516 implementation interprets the specification (which is a little ambiguous)
 * to mean that a name, once reserved, cannot be re-used ever, even if the registered object
 * is later deleted.
 * @param theObjectName a {@link java.lang.String} holding the proposed object instance name
 * @throws IllegalName if the <code>theObjectName</code> is ill-formed (begins with "HLA")
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #registerObjectInstance registerObjectInstance
 * @see #registerObjectInstanceWithRegions registerObjectInstanceWithRegions
 * @see #getObjectInstanceName getObjectInstanceName
 * @see FederateAmbassador#objectInstanceNameReservationSucceeded
 * @see FederateAmbassador#objectInstanceNameReservationFailed
 */

public void reserveObjectInstanceName(String theObjectName)
    throws IllegalName,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
** Registers a new instance of the specified object class with the RTI.
* Note that this does *not* trigger
  * requestAttributeOwnershipAssumption
  * callbacks for object instance attributes that are initially unowned.
  * @param theClass the {@link ObjectClassHandle} of the object instance being registered
  * @param theClass the {@link ObjectClassHandle} of the object instance being registered
* @return the registered object instance's {@link ObjectInstanceHandle}
  * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
  * @throws ObjectClassNotPublished if <code>theClass</code> isn't published by the federate
  * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
  * @throws SaveInProgress if the federate is in one of the save-in-progress states
  * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
  * @throws RTIinternalError if something else goes wrong
* @see #registerObjectInstanceWithRegions
* @see #deleteObjectInstance deleteObjectInstance
* @see FederateAmbassador#startRegistrationForObjectClass startRegistrationForObjectClass
* @see FederateAmbassador#stopRegistrationForObjectClass stopRegistrationForObjectClass
* @see FederateAmbassador#discoverObjectInstance discoverObjectInstance

```java
public ObjectInstanceHandle registerObjectInstance(
    ObjectClassHandle theClass)
throws ObjectClassNotDefined,
    ObjectClassNotPublished,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
```
/**
 * Registers a new instance of the specified object class with the RTI, using a previously reserved name.
 * Note that this does not trigger callbacks for object instance attributes that are initially unowned.
 * @param theClass the ObjectClassHandle of the object instance being registered
 * @param theObjectName a java.lang.String holding the previously reserved object instance name
 * @return the registered object instance's ObjectInstanceHandle
 * @throws ObjectClassNotDefined if theClass isn't recognized by the RTI
 * @throws ObjectClassNotPublished if theClass isn't published by the federate
 * @throws ObjectInstanceNameNotReserved if theObjectName has not been previously reserved
 * @throws ObjectInstanceNameInUse if theObjectName is already in use
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #registerObjectInstanceWithRegions
 * @see #reserveObjectInstanceName
 * @see #deleteObjectInstance
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 * @see FederateAmbassador#discoverObjectInstance
 */
public ObjectInstanceHandle registerObjectInstance(
    ObjectClassHandle theClass,
    String theObjectName)
    throws ObjectClassNotDefined,
        ObjectClassNotPublished,
        ObjectInstanceNameNotReserved,
        ObjectInstanceNameInUse,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
}
** Provides the RTI with updated values for certain instance attributes.
   * @param theObject the [ObjectInstanceHandle] of the object instance
   * @param theAttributes an [AttributeHandleValueMap] holding the updated values, keyed by attribute
   * @param userSuppliedTag a byte[] tag (this parameter may be null)
   * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered theObject
   * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
   * @throws AttributeNotOwned if an attribute is not owned by the federate
   * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
   * @throws SaveInProgress if the federate is in one of the save-in-progress states
   * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
   * @throws RTIinternalError if something else goes wrong
   * @see FederateAmbassador#turnUpdatesOnForObjectInstance
   * @see FederateAmbassador#turnUpdatesOffForObjectInstance
   * @see FederateAmbassador#reflectAttributeValues
   *
   public void updateAttributeValues(
      ObjectInstanceHandle theObject,
      AttributeHandleValueMap theAttributes,
      byte[] userSuppliedTag)
      throws ObjectInstanceNotKnown,
      AttributeNotDefined,
      AttributeNotOwned,
      FederateNotExecutionMember,
      SaveInProgress,
      RestoreInProgress,
      RTIinternalError;
/**
 * Provides the RTI with updated values for certain instance attributes at a specified time-stamp.
 * @param theObject the ObjectInstanceHandle of the object instance
 * @param theAttributes an AttributeHandleValueMap holding the updated values, keyed by attribute
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param theTime the LogicalTime at which the updated values become valid
 * @return the message's MessageRetractionReturn, should a retraction become necessary
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered theObject
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws AttributeNotOwned if an attribute is not owned by the federate
 * @throws InvalidLogicalTime if the specified LogicalTime is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #retract
 * @see FederateAmbassador#turnUpdatesOnForObjectInstance
 * @see FederateAmbassador#turnUpdatesOffForObjectInstance
 * @see FederateAmbassador#reflectAttributeValues
 */

public MessageRetractionReturn updateAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    LogicalTime theTime)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    InvalidLogicalTime,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Sends an interaction. Note that the sender may send a subset of
 * the interaction's available parameters.
 * @param theInteraction the [InteractionClassHandle] of the
 * interaction being sent
 * @param theParameters a [ParameterHandleValueMap] holding
 * the interaction values, keyed by parameter
 * @param userSuppliedTag a byte[] tag (this parameter
 * may be null)
 * @throws InteractionClassNotPublished if the federate does not
 * publish theInteraction
 * @throws InteractionClassNotDefined if
 * theInteraction isn't recognized by the RTI
 * @throws InteractionParameterNotDefined if one of
 * theParameters isn't recognized in the supplied context
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #sendInteractionWithRegions
 * @see FederateAmbassador.turnInteractionsOn
 * @see FederateAmbassador.turnInteractionsOff
 * @see FederateAmbassador.receiveInteraction(InteractionClassHandle,Parameter
 * HandleValueMap,byte[],OrderType,TransportationType)
 */
public void sendInteraction(InteractionClassHandle theInteraction,
ParameterHandleValueMap theParameters,
byte[] userSuppliedTag)
throws InteractionClassNotPublished,
InteractionClassNotDefined,
InteractionParameterNotDefined,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
/**
 * Sends an interaction at a specified time-stamp.
 * @param theInteraction the {@link InteractionClassHandle} of the
 * interaction being sent
 * @param theParameters a {@link ParameterHandleValueMap} holding
 * the interaction values, keyed by parameter
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @param theTime the {@link LogicalTime} at which the interaction
 * occurs
 * @return the message's {@link MessageRetractionReturn}, should a
 * retraction become necessary
 * @throws InteractionClassNotPublished if the federate does not
 * publish <code>theInteraction</code>
 * @throws InteractionClassNotDefined if <code>theInteraction</code>
 * isn't recognized by the RTI
 * @throws InteractionParameterNotDefined if one of
 * <code>theParameters</code> isn't recognized in the supplied context
 * @throws InvalidLogicalTime if the specified
 * <code>LogicalTime</code> is invalid
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #retract
 * @see #sendInteractionWithRegions
 * @see FederateAmbassador#turnInteractionsOn
 * @see FederateAmbassador#turnInteractionsOff
 * @see FederateAmbassador#receiveInteraction
 */

public MessageRetractionReturn
    sendInteraction(
        InteractionClassHandle theInteraction,
        ParameterHandleValueMap theParameters,
        byte[] userSuppliedTag,
        LogicalTime theTime)
    throws InteractionClassNotPublished,
              InteractionClassNotDefined,
              InteractionParameterNotDefined,
              InvalidLogicalTime,
              FederateNotExecutionMember,
              SaveInProgress,
              RestoreInProgress,
              RTIinternalError;
/ 6.10
**
* Notifies the RTI that an object instance is to be deleted.
* @param objectHandle the [@link ObjectInstanceHandle] of the
  object instance
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter
  may be <code>null</code>)
* @throws DeletePrivilegeNotHeld if the federate does not own the
  <code>HLAprivilegeToDeleteObject</code> attribute of the specified
  object instance
* @throws ObjectInstanceNotKnown if the RTI considers that the
  federate has not discovered <code>theObject</code>
* @throws FederateNotExecutionMember if the federate is not
  currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-
  progress states
* @throws RestoreInProgress if the federate is in one of the
  restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #registerObjectInstance registerObjectInstance
* @see #registerObjectInstanceWithRegions
* @see FederateAmbassador#removeObjectInstance
* @see FederateAmbassador#startRegistrationForObjectClass
* @see FederateAmbassador#stopRegistrationForObjectClass
*
  public void
deleteObjectInstance(
    ObjectInstanceHandle objectHandle,
    byte[] userSuppliedTag)
  throws DeletePrivilegeNotHeld,
    ObjectInstanceNotKnown,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
   * Notifies the RTI that an object instance is to be deleted at a
   * specified time-stamp.
   * @param objectHandle the {@link ObjectInstanceHandle} of the
   * object instance
   * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
   * may be <code>null</code>)
   * @param theTime the {@link LogicalTime} at which the object
   * instance is deleted
   * @return the message's {@link MessageRetractionReturn}, should a
   * retraction become necessary
   * @throws DeletePrivilegeNotHeld if the federate does not own the
   * <code>HLAprivilegeToDeleteObject</code> attribute of the specified
   * object instance
   * @throws ObjectInstanceNotKnown if the RTI considers that the
   * federate has not discovered <code>theObject</code>
   * @throws InvalidLogicalTime if the specified
   * <code>LogicalTime</code> is invalid
   * @throws FederateNotExecutionMember if the federate is not
   * currently joined to a federation execution
   * @throws SaveInProgress if the federate is in one of the save-in-
   * progress states
   * @throws RestoreInProgress if the federate is in one of the
   * restore-in-progress states
   * @throws RTIinternalError if something else goes wrong
   * @see #registerObjectInstance registerObjectInstance
   * @see #registerObjectInstanceWithRegions
   * @see #retract retract
   * @see FederateAmbassador#removeObjectInstance
   * @see FederateAmbassador#startRegistrationForObjectClass
   * @see FederateAmbassador#stopRegistrationForObjectClass
   */
   public MessageRetractionReturn
   deleteObjectInstance(
   ObjectInstanceHandle objectHandle,
   byte[] userSuppliedTag,
   LogicalTime theTime)
   throws DeletePrivilegeNotHeld,
   ObjectInstanceNotKnown,
   InvalidLogicalTime,
   FederateNotExecutionMember,
   SaveInProgress,
   RestoreInProgress,
   RTIinternalError;
/**
 * Notifies the RTI that the federate has "forgotten" all about a subscribed object instance and should therefore discover it anew.
 * @param objectHandle the [ObjectInstanceHandle] of the object instance
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered the object
 * @throws FederateOwnsAttributes if the federate owns some instance attributes of the specified object instance
 * @throws OwnershipAcquisitionPending if there is an ownership acquisition pending for some instance attribute of the specified object instance
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see FederateAmbassador#discoverObjectInstance
 * discoverObjectInstance
 */
 public void localDeleteObjectInstance(
     ObjectInstanceHandle objectHandle)
 throws ObjectInstanceNotKnown,
     FederateOwnsAttributes,
     OwnershipAcquisitionPending,
     FederateNotExecutionMember,
     SaveInProgress,
     RestoreInProgress,
     RTIinternalError;
// 6.13
/**
 * Notifies the RTI that future {updateAttributeValues} invocations should use the specified {TransportationType}.
 */
<p>
 * The attribute transportation types revert to their FDD-specified values once this federate loses ownership.
 * @param theObject the {ObjectInstanceHandle} of the object instance
 * @param theAttributes an {AttributeHandleSet} specifying the subject attributes
 * @param theType the {TransportationType} to which <code>theAttributes</code> should switch
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws AttributeNotOwned if an attribute is not owned by the federate
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #changeAttributeOrderType changeAttributeOrderType
 * @see #changeInteractionTransportationType
 */
public void changeAttributeTransportationType(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    TransportationType theType)
throws ObjectInstanceNotKnown,
        AttributeNotDefined,
        AttributeNotOwned,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;

/**
 * Notifies the RTI that future \{@link #sendInteraction sendInteraction\} invocations should use the specified \{@link TransportationType\}.
 * @param theClass the \{@link InteractionClassHandle\} of the subject interaction
 * @param theType the \{@link TransportationType\} to which \texttt{theClass} should switch
 * @throws InteractionClassNotDefined if \texttt{theClass} isn't recognized by the RTI
 * @throws InteractionClassNotPublished if the federate does not publish \texttt{theClass}
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see \#changeInteractionOrderType changeInteractionOrderType
 * @see \#changeAttributeTransportationType changeAttributeTransportationType
 */

public void changeInteractionTransportationType(
    InteractionClassHandle theClass,
    TransportationType theType)
    throws InteractionClassNotDefined,
            InteractionClassNotPublished,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
/**
 * Requests that attribute value updates be provided for the
 * specified instance attributes.
 * @param theObject the ObjectInstanceHandle of the object
 * instance
 * @param theAttributes an AttributeHandleSet specifying
 * the attributes for which value updates are requested
 * @param userSuppliedTag a byte[] tag (this parameter
 * may be null)
 * @throws ObjectInstanceNotKnown if the RTI considers that the
 * federate has not discovered theObject
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see requestAttributeValueUpdateWithRegions
 * @see FederateAmbassador#provideAttributeValueUpdate
 */
public void requestAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Requests that attribute value updates be provided for all
 * instance attributes of the specified object class.
 * @param theClass the ObjectClassHandle of the object
 * class being polled
 * @param theAttributes an AttributeHandleSet listing the
 * attributes being polled
 * @param userSuppliedTag a byte[] tag (this parameter
 * may be null)
 * @throws ObjectClassNotDefined if theClass isn't
 * recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see requestAttributeValueUpdateWithRegions
 * @see FederateAmbassador#provideAttributeValueUpdate
 */

public void
requestAttributeValueUpdate(
    ObjectClassHandle theClass,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
    throws ObjectClassNotDefined,
    AttributeNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
7.2

**Notifies the RTI that the federate immediately divests itself of ownership of the specified instance attributes.**

The RTI may issue [*link FederateAmbassador#requestAttributeOwnershipAssumption* requestAttributeOwnershipAssumption]

callbacks at eligible joined federates for object instance attributes that become unowned.

* @param theObject the [*link ObjectInstanceHandle*] of the object instance
* @param theAttributes an [*link AttributeHandleSet*] specifying the divested attributes
* @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered `<code>theObject</code>`
* @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
* @throws AttributeNotOwned if an attribute is not owned by the federate
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong

```
public void unconditionalAttributeOwnershipDivestiture(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
```
Notifies the RTI that the federate wishes to relinquish ownership of the specified instance attributes to any other willing federate(s).

* The RTI may issue \( \text{@link FederateAmbassador\#requestAttributeOwnershipAssumption requestAttributeOwnershipAssumption} \) callbacks at eligible joined federates for the specified object instance attributes.

* @param theObject the \( \text{@link ObjectInstanceHandle} \) of the object instance

* @param theAttributes an \( \text{@link AttributeHandleSet} \) specifying the attributes to divest

* @param userSuppliedTag a \( \text{byte[]} \) tag (this parameter may be \( \text{null} \))

* @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered \( \text{theObject} \)

* @throws AttributeNotDefined if an attribute could not be recognized within the supplied context

* @throws AttributeNotOwned if an attribute is not owned by the federate

* @throwsAttributeAlreadyBeingDivested if an attribute ownership divestiture request is already pending for one of the attribute instances

* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution

* @throws SaveInProgress if the federate is in one of the save-in-progress states

* @throws RestoreInProgress if the federate is in one of the restore-in-progress states

* @throws RTIinternalError if something else goes wrong

* @see FederateAmbassador\#requestDivestitureConfirmation requestDivestitureConfirmation

* @see \#confirmDivestiture confirmDivestiture

* @see \#cancelNegotiatedAttributeOwnershipDivestiture cancelNegotiatedAttributeOwnershipDivestiture

* @see \#unpublishObjectClassAttributes unpublishObjectClassAttributes

/**
 * @param theObject the \( \text{@link ObjectInstanceHandle} \) of the object instance
 * @param theAttributes an \( \text{@link AttributeHandleSet} \) specifying the attributes to divest
 * @param userSuppliedTag a \( \text{byte[]} \) tag (this parameter may be \( \text{null} \))
 */

public void negotiatedAttributeOwnershipDivestiture(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    AttributeAlreadyBeingDivested,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Completes the negotiated divestiture of the specified instance attributes.
 * <p>
 * In the DoD Interpretations of IEEE 1516-2000v2, the exception No Acquisition Pending is added.
 * @param theObject the {@link ObjectInstanceHandle} of the object instance
 * @param theAttributes an {@link AttributeHandleSet} specifying the confirmed attributes
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws AttributeNotOwned if an attribute is not owned by the federate
 * @throws AttributeDivestitureWasNotRequested if the divestiture was not previously requested by the federate
 * @throws NoAcquisitionPending if the RTI has not yet located candidate federates
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #negotiatedAttributeOwnershipDivestiture negotiatedAttributeOwnershipDivestiture
 * @see FederateAmbassador#requestDivestitureConfirmation requestDivestitureConfirmation
 */
public void
confirmDivestiture(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    NoAcquisitionPending,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
// 7.8
/**
 * Notifies the RTI that the federate wishes to acquire ownership
 * of the specified instance attributes.
 * @param theObject the {@link ObjectInstanceHandle} of the object
 * instance
 * @param desiredAttributes an {@link AttributeHandleSet}
 * specifying the desired attributes
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws ObjectInstanceNotKnown if the RTI considers that the
 * federate has not discovered <code>theObject</code>
 * @throws ObjectClassNotPublished if <code>theClass</code> isn't
 * published by the federate
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws AttributeNotPublished if an attribute is not published
 * by the federate
 * @throws FederateOwnsAttributes if the federate already owns some
 * of the specified instance attributes
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see FederateAmbassador#requestAttributeOwnershipRelease
 * requestAttributeOwnershipRelease
 * @see FederateAmbassador#attributeOwnershipAcquisitionNotification
 * attributeOwnershipAcquisitionNotification
 * @see #cancelAttributeOwnershipAcquisition
 * cancelAttributeOwnershipAcquisition
 */

public void
attributeOwnershipAcquisition(
    ObjectInstanceHandle theObject,
    AttributeHandleSet desiredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    FederateOwnsAttributes,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Notifies the RTI that the federate wishes to acquire ownership of the specified instance attributes,
 * but only if they are currently unowned or in the process of being divested.
 * <p>
 * Attributes made available through {@link #negotiatedAttributeOwnershipDivestiture} are eligible for acquisition in this manner.
 * Other federates aren't bothered by this call; the requesting federate receives {@link FederateAmbassador#attributeOwnershipUnavailable}
 * for those attributes which are currently owned by other federates, {@link FederateAmbassador#attributeOwnershipAcquisitionNotification}
 * for the unowned ones, and no response for those attributes which do not currently exist.
 * Note also that the owning federate may stay in the Completing Divestiture state indefinitely (i.e. it postpones emitting {@link #confirmDivestiture} indefinitely); the requesting federate won't receive any callbacks during that time.
 * It is not clear whether {@link #cancelAttributeOwnershipAcquisition} (whilst the requesting federate is awaiting a response)
 * would raise an {@link AttributeAcquisitionWasNotRequested} exception or not.
@param theObject the ObjectInstanceHandle of the object instance
@param desiredAttributes an AttributeHandleSet specifying the desired attributes
@throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered theObject
@throws ObjectClassNotPublished if theClass isn't published by the federate
@throws AttributeNotDefined if an attribute could not be recognized within the supplied context
@throws AttributeNotPublished if an attribute is not published by the federate
@throws FederateOwnsAttributes if the federate already owns some of the specified instance attributes
@throws AttributeAlreadyBeingAcquired if an unconditional attribute ownership acquisition request is already pending for an attribute instance
@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
@throws SaveInProgress if the federate is in one of the save-in-progress states
@throws RestoreInProgress if the federate is in one of the restore-in-progress states
@throws RTIinternalError if something else goes wrong
@see FederateAmbassador#attributeOwnershipAcquisitionNotification
@see FederateAmbassador#attributeOwnershipAcquisitionNotification
@see FederateAmbassador#attributeOwnershipUnavailable
@see FederateAmbassador#attributeOwnershipUnavailable
@see #cancelAttributeOwnershipAcquisition
cancelAttributeOwnershipAcquisition
*/
public void attributeOwnershipAcquisitionIfAvailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet desiredAttributes)
throws ObjectInstanceNotKnown,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    FederateOwnsAttributes,
    AttributeAlreadyBeingAcquired,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
* Notifies the RTI that the federate is willing to relinquish ownership of the specified instance attributes
* if any other federates are attempting to acquire them. The method concludes the divestiture immediately one way or another.
* @param theObject the {link ObjectInstanceHandle} of the object instance
* @param theAttributes an {link AttributeHandleSet} specifying the offered attributes
* @return an {link AttributeHandleSet} specifying the attributes successfully divested
* @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
* @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
* @throws AttributeNotOwned if an attribute is not owned by the federate
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see FederateAmbassador#requestAttributeOwnershipRelease requestAttributeOwnershipRelease
*/

public AttributeHandleSet
attributeOwnershipDivestitureIfWanted(
ObjectInstanceHandle theObject,
AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
AttributeNotDefined,
AttributeNotOwned,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
/**
 * Notifies the RTI that the federate is cancelling the outstanding
 * negotiatedAttributeOwnershipDivestiture of
 * the specified instance attributes.
 * @param theObject the ObjectInstanceHandle of the object instance
 * @param theAttributes an AttributeHandleSet specifying
 * the attributes for which divestiture is canceled
 * @throws ObjectInstanceNotKnown if the RTI considers that the
 * federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws AttributeNotOwned if an attribute is not owned by the
 * federate
 * @throws AttributeDivestitureWasNotRequested if the divestiture
 * was not previously requested by the federate
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see negotiatedAttributeOwnershipDivestiture
 */

public void cancelNegotiatedAttributeOwnershipDivestiture(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
              AttributeNotDefined,
              AttributeNotOwned,
              AttributeDivestitureWasNotRequested,
              FederateNotExecutionMember,
              SaveInProgress,
              RestoreInProgress,
              RTIinternalError;
/**
 * Notifies the RTI that the federate is cancelling the outstanding
 * {@link #attributeOwnershipAcquisition attributeOwnershipAcquisition}
 * of
 * the specified instance attributes.
 * @param theObject the {@link ObjectInstanceHandle} of the object
 * instance
 * @param theAttributes an {@link AttributeHandleSet} specifying
 * the attributes for which acquisition is canceled
 * @throws ObjectInstanceNotKnown if the RTI considers that the
 * federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws AttributeAlreadyOwned if the cancellation occurs too
 * late (i.e. it has already been granted ownership)
 * @throws AttributeAcquisitionWasNotRequested if there is no
 * pending attribute ownership acquisition to cancel
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #attributeOwnershipAcquisition
 * @see FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation
 * confirmAttributeOwnershipAcquisitionCancellation
 */
public void cancelAttributeOwnershipAcquisition(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Requests that the RTI report the ownership status of the specified instance attribute.
 * @param theObject the {ObjectInstanceHandle} of the object instance
 * @param theAttribute an {AttributeHandle} specifying the queried attribute
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see FederateAmbassador#informAttributeOwnership
 * @see FederateAmbassador#attributeIsNotOwned
 * @see FederateAmbassador#attributeIsOwnedByRTI
 */

public void queryAttributeOwnership(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute
) throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
// 7.18
/**
 * Requests that the RTI report whether or not the federate owns the specified instance attribute.
 * @param theObject the {@link ObjectInstanceHandle} of the object instance
 * @param theAttribute an {@link AttributeHandle} specifying the queried attribute
 * @return <code>true</code> iff <code>theAttribute</code> is owned by the federate
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 */
public boolean isAttributeOwnedByFederate(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;}
public void enableTimeRegulation(
    LogicalTimeInterval lookahead)
    throws TimeRegulationAlreadyEnabled,
            InvalidLookahead,
            InTimeAdvancingState,
            RequestForTimeRegulationPending,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;

/**
 * Requests that this federate become time-regulating (with the
 * specified lookahead), thereby enabling it to send time-stamped messages.
 * @param lookahead a [@link LogicalTimeInterval] specifying the
 * new lookahead
 * @throws TimeRegulationAlreadyEnabled if time regulation is
 * already enabled
 * @throws InvalidLookahead if theLookahead is invalid
 * @throws InTimeAdvancingState if the federate is in the time-
 * advancing state
 * @throws RequestForTimeRegulationPending if a time regulation
 * request is pending
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableTimeRegulation disableTimeRegulation
 * @see FederateAmbassador#timeRegulationEnabled
 */
// 8.4
/**
 * Requests that this federate no longer be time-regulating,
 * thereby making its future messages receive-ordered.
 * @throws TimeRegulationIsNotEnabled if time regulation is currently disabled
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #enableTimeRegulation enableTimeRegulation
 */
public void
disableTimeRegulation()
throws TimeRegulationIsNotEnabled,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;

// 8.5
/**
 * Requests that this federate become time-constrained.
 * @throws TimeConstrainedAlreadyEnabled if time constraint is already enabled
 * @throws InTimeAdvancingState if the federate is in the time-advancing state
 * @throws RequestForTimeConstrainedPending if a time constraint request is pending
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableTimeConstrained disableTimeConstrained
 * @see FederateAmbassador#timeConstrainedEnabled
timeConstrainedEnabled
*/
public void
enableTimeConstrained()
throws TimeConstrainedAlreadyEnabled,
InTimeAdvancingState,
RequestForTimeConstrainedPending,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
public void disableTimeConstrained() throws TimeConstrainedIsNotEnabled, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
**Requests an advance of the federate's logical time to the specified value,**
* thereby releasing queued messages of appropriate time-stamps.

The federate is guaranteeing that it will not later generate a
{@link OrderType TIMESTAMP} message with a time stamp less than or equal to
the specified logical time, even if its lookahead is zero.
A Time Advance Grant completes this request and indicates to the federate
that no additional {@link OrderType TIMESTAMP} messages will be later delivered
to it with time stamps less than or equal to the logical time of the grant.

@throws InvalidLogicalTime if the specified timeAdvanceRequestAvailable
LogicalTime is invalid
@throws LogicalTimeAlreadyPassed if the specified LogicalTime is in the federation's past
@throws InTimeAdvancingState if the federate is in the time-advancing state
@throws RequestForTimeRegulationPending if a time regulation request is pending
@throws RequestForTimeConstrainedPending if a time constraint request is pending
@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
@throws SaveInProgress if the federate is in one of the save-in-progress states
@throws RestoreInProgress if the federate is in one of the restore-in-progress states
@throws RTIinternalError if something else goes wrong
@see #timeAdvanceRequestAvailable timeAdvanceRequestAvailable
@see #nextMessageRequest nextMessageRequestAvailable
@see #flushQueueRequest flushQueueRequest
@see FederateAmbassador#timeAdvanceGrant timeAdvanceGrant

```java
public void timeAdvanceRequest(
    LogicalTime theTime)
throws InvalidLogicalTime,
LogicalTimeAlreadyPassed,
InTimeAdvancingState,
RequestForTimeRegulationPending,
RequestForTimeConstrainedPending,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
```
// 8.9
/**
 * Requests an advance of the federate's logical time to the
 * specified value,
 * thereby releasing queued messages of appropriate time-stamps,
 * whilst allowing
 * current-time messages to still be exchanged.
 * <p>
 * timeAdvanceRequestAvailable is similar to timeAdvanceRequest,
 * except that
 * the RTI does not guarantee delivery of <i>all</i> messages with
 * time stamps
 * <i>equal to</i> the logical time granted. The federate is also
 * allowed to
 * send additional messages with time stamps equal to the logical
 * time granted,
 * if the federate's actual lookahead is zero.
 * @param theTime the {link LogicalTime} to which the federate
 * wishes to advance
 * @throws InvalidLogicalTime if the specified
 * <code>LogicalTime</code> is invalid
 * @throws LogicalTimeAlreadyPassed if the specified
 * <code>LogicalTime</code> is in the federation's past
 * @throws InTimeAdvancingState if the federate is in the time-
 * advancing state
 * @throws RequestForTimeRegulationPending if a time regulation
 * request is pending
 * @throws RequestForTimeConstrainedPending if a time constraint
 * request is pending
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #timeAdvanceRequest timeAdvanceRequest
 * @see #nextMessageRequest nextMessageRequest
 * @see #nextMessageRequestAvailable nextMessageRequestAvailable
 * @see #flushQueueRequest flushQueueRequest
 * @see FederateAmbassador#timeAdvanceGrant timeAdvanceGrant
 */
public void
timeAdvanceRequestAvailable(
    LogicalTime theTime)
throws InvalidLogicalTime,
    LogicalTimeAlreadyPassed,
    InTimeAdvancingState,
    RequestForTimeRegulationPending,
    RequestForTimeConstrainedPending,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Notifies the RTI that the federate wishes to advance to the lesser LogicalTime of
 * the one specified and the next time-stamped message awaiting delivery.
 * <p>
 * nextMessageRequest is similar to timeAdvanceRequest except that the time advance granted
 * is a function of the time-stamp of the next timestamp message awaiting delivery.
 * @param theTime the LogicalTime beyond which the federate does not wish to advance
 * @throws InvalidLogicalTime if the specified LogicalTime is invalid
 * @throws LogicalTimeAlreadyPassed if the specified LogicalTime is in the federation's past
 * @throws InTimeAdvancingState if the federate is in the time-advancing state
 * @throws RequestForTimeRegulationPending if a time regulation request is pending
 * @throws RequestForTimeConstrainedPending if a time constraint request is pending
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #nextMessageRequestAvailable
 * @see #timeAdvanceRequest
 * @see #timeAdvanceRequestAvailable
 * @see #flushQueueRequest
 * @see FederateAmbassador#timeAdvanceGrant
 */

public void nextMessageRequest(
          LogicalTime theTime)
    throws InvalidLogicalTime,
           LogicalTimeAlreadyPassed,
           InTimeAdvancingState,
           RequestForTimeRegulationPending,
           RequestForTimeConstrainedPending,
           FederateNotExecutionMember,
           SaveInProgress,
           RestoreInProgress,
           RTIinternalError;
** nextMessageRequestAvailable is similar to timeAdvanceRequestAvailable except that the time advance granted is a function of the time-stamp of the next Timestamp message awaiting delivery.

* @param theTime the LogicalTime beyond which the federate does not wish to advance
* @throws InvalidLogicalTime if the specified LogicalTime is invalid
* @throws LogicalTimeAlreadyPassed if the specified LogicalTime is in the federation's past
* @throws InTimeAdvancingState if the federate is in the time-advancing state
* @throws RequestForTimeRegulationPending if a time regulation request is pending
* @throws RequestForTimeConstrainedPending if a time constraint request is pending
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIInternalError if something else goes wrong
* @see #nextMessageRequest nextMessageRequest
* @see #timeAdvanceRequest timeAdvanceRequest
* @see #timeAdvanceRequestAvailable timeAdvanceRequestAvailable
* @see #flushQueueRequest flushQueueRequest
* @see FederateAmbassador#timeAdvanceGrant timeAdvanceGrant
*/

public void nextMessageRequestAvailable(
        LogicalTime theTime)
        throws InvalidLogicalTime,
                LogicalTimeAlreadyPassed,
                InTimeAdvancingState,
                RequestForTimeRegulationPending,
                RequestForTimeConstrainedPending,
                FederateNotExecutionMember,
                SaveInProgress,
                RestoreInProgress,
                RTIInternalError;
// 8.12
/**
* Notifies the RTI that the federate wishes to advance to the lesser [link LogicalTime] of
* the one specified and the next time-stamped message awaiting delivery, but should nevertheless
* receive all currently-queued time-stamped messages awaiting delivery.
* <p>
* flushQueueRequest is similar to nextMessageRequest except that some anticipated messages may be received.
* As a consequence, later time-stamped messages may not be received in the "correct" order.
* @param theTime the [link LogicalTime] beyond which the federate does not wish to advance
* @throws InvalidLogicalTime if the specified <code>LogicalTime</code> is invalid
* @throws LogicalTimeAlreadyPassed if the specified <code>LogicalTime</code> is in the federation's past
* @throws InTimeAdvancingState if the federate is in the time-advancing state
* @throws RequestForTimeRegulationPending if a time regulation request is pending
* @throws RequestForTimeConstrainedPending if a time constraint request is pending
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #nextMessageRequest
* @see #nextMessageRequestAvailable
* @see #timeAdvanceRequest
* @see #timeAdvanceRequestAvailable
* @see FederateAmbassador#timeAdvanceGrant
*/
public void
flushQueueRequest(
    LogicalTime theTime)
throws InvalidLogicalTime,
    LogicalTimeAlreadyPassed,
    InTimeAdvancingState,
    RequestForTimeRegulationPending,
    RequestForTimeConstrainedPending,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Instructs the RTI to deliver \(\text{@link OrderType RECEIVE}\) messages when the federate is in either of the
 * Time Advancing and Time Granted states.
 * @throws AsynchronousDeliveryAlreadyEnabled if asynchronous delivery is already enabled
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableAsynchronousDelivery disableAsynchronousDelivery
 */
public void enableAsynchronousDelivery()
throws AsynchronousDeliveryAlreadyEnabled,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;

/**
 * Instructs the RTI to deliver \(\text{@link OrderType RECEIVE}\) messages only when the federate is in the
 * Time Advancing state. This is only applicable to time-constrained federates.
 * @throws AsynchronousDeliveryAlreadyDisabled if asynchronous delivery is already disabled
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableAsynchronousDelivery disableAsynchronousDelivery
 */
public void disableAsynchronousDelivery()
throws AsynchronousDeliveryAlreadyDisabled,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
// 8.16
/**
 * Requests the federate's current GALT (Greatest Available Logical Time).
 * @return a [TimeQueryReturn] containing the requested logical time and a validity guard boolean
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 */
public TimeQueryReturn queryGALT() throws FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;

// 8.17
/**
 * Requests the federate's current logical time.
 * @return the requested [LogicalTime]
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 */
public LogicalTime queryLogicalTime() throws FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
public TimeQueryReturn queryLITS() throws FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;

// 8.19
/**
 * Requests a change to the federate's lookahead.
 * If the new lookahead is smaller than the current one, the change will come about gradually.
 * The federate's lookahead will shrink every time its logical time advances so that the federate's "future" remains at a constant logical time, until the specified lookahead value is reached.
 * @param theLookahead a LogicalTimeInterval specifying the requested new lookahead
 * @throws TimeRegulationIsNotEnabled if time regulation is currently disabled
 * @throws InvalidLookahead if theLookahead is invalid
 * @throws InTimeAdvancingState if the federate is in the time-advancing state
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 */
public void modifyLookahead(LogicalTimeInterval theLookahead) throws TimeRegulationIsNotEnabled, InvalidLookahead, InTimeAdvancingState, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
// 8.20
/**
 * Requests the federate's current lookahead.
 * @return the requested \{LogicalTimeInterval\}
 * @throws TimeRegulationIsNotEnabled if time regulation is currently disabled
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #modifyLookahead modifyLookahead
 */
public LogicalTimeInterval queryLookahead()
throws TimeRegulationIsNotEnabled, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;

// 8.21
/**
 * Notifies the RTI that a message previously sent by the joined federate is to be retracted.
 * @param theHandle the \{MessageRetractionHandle\} of the message to retract
 * @throws InvalidMessageRetractionHandle if the \<code\>MessageRetractionHandle\</code\> is invalid
 * @throws TimeRegulationIsNotEnabled if time regulation is currently disabled
 * @throws MessageCanNoLongerBeRetracted the message specified can no longer be retracted
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see FederateAmbassador#requestRetraction requestRetraction
 */
public void retract(
    MessageRetractionHandle theHandle)
throws InvalidMessageRetractionHandle, TimeRegulationIsNotEnabled, MessageCanNo LongerBeRetracted, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
** Notifies the RTI that future [updateAttributeValues] invocations should use the specified [orderType].

* <p>

* The attribute ordering types revert to their FDD-specified values once this federate loses ownership.

* @param theObject the [ObjectInstanceHandle] of the object instance

* @param theAttributes an [AttributeHandleSet] specifying the subject attributes

* @param theType the [orderType] to which <code>theAttributes</code> should switch

* @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>

* @throws AttributeNotDefined if an attribute could not be recognized within the supplied context

* @throws AttributeNotOwned if an attribute is not owned by the federate

* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution

* @throws SaveInProgress if the federate is in one of the save-in-progress states

* @throws RestoreInProgress if the federate is in one of the restore-in-progress states

* @throws RTIinternalError if something else goes wrong

* @see #changeAttributeTransportationType

changeAttributeOrderType

* @see #changeInteractionOrderType changeInteractionOrderType

*/

public void changeAttributeOrderType(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    OrderType theType)
throws ObjectInstanceNotKnown,
    AttributeNotDefined,
    AttributeNotOwned,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
// 8.24
/**
 * Notifies the RTI that future \@link #sendInteraction sendInteraction invocations should use the specified \@link OrderType].
 * @param theClass the \@link InteractionClassHandle] of the subject interaction
 * @param theType the \@link OrderType] to which
 * <code>theClass</code> should switch
 * @throws InteractionClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws InteractionClassNotPublished if the federate does not publish <code>theClass</code>
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see \#changeAttributeOrderType changeAttributeOrderType
 * @see \#changeInteractionTransportationType changeInteractionTransportationType
 */
public void changeInteractionOrderType(
    InteractionClassHandle theClass,
    OrderType theType)
throws InteractionClassNotDefined,
    InteractionClassNotPublished,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
9.2

**
* Creates a region template that has the specified dimensions.
* Before the region can be used for update or subscription, [@link #setRangeBounds setRangeBounds] must
* be invoked at least once for each dimension specified. Only then can [@link #commitRegionModifications commitRegionModifications]
* be invoked to turn the region template into a region specification.
* @param dimensions a [@link DimensionHandleSet] specifying the
* region’s dimensions
* @return the [@link RegionHandle] of the newly created region
* template
* @throws InvalidDimensionHandle if one of the [@link DimensionHandle]s is invalid
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #deleteRegion deleteRegion
*/

public RegionHandle createRegion(
    DimensionHandleSet dimensions)
throws InvalidDimensionHandle,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
Notifies the RTI that pending dimension range modifications should be applied to the regions.

[code]commitRegionModifications[/code] turns region templates into region specifications and updates region realizations.

@throws InvalidRegion if one of the region templates has not had all of its ranges set
@throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
@throws SaveInProgress if the federate is in one of the save-in-progress states
@throws RestoreInProgress if the federate is in one of the restore-in-progress states
@throws RTIinternalError if something else goes wrong
@see #getRangeBounds
@see #setRangeBounds

public void
commitRegionModifications(
    RegionHandleSet regions)
throws InvalidRegion,
    RegionNotCreatedByThisFederate,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Deletes the specified region.
 * Before a region can be deleted, it must be unassociated from any
 * subscriptions or updates.
 * @param theRegion the \{@link RegionHandle\} to delete (template or
 * specification)
 * @throws InvalidRegion if the \{@link RegionHandle\} is invalid
 * @throws RegionNotCreatedByThisFederate if the federate does not
 * own (did not create) one of the regions in the set
 * @throws RegionInUseForUpdateOrSubscription if the region is
 * still in use
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #createRegion createRegion
 */

public void deleteRegion(RegionHandle theRegion)
throws InvalidRegion,
        RegionNotCreatedByThisFederate,
        RegionInUseForUpdateOrSubscription,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
// 9.5
/**
 * Registers a new instance of the specified object class with the RTI
 * and simultaneously associates its attributes with distribution regions.
 * @param theClass the ObjectClassHandle of the object instance being registered
 * @param attributesAndRegions an AttributeSetRegionSetPairList listing the attributes and regions being simultaneously associated
 * @return the registered object instance's ObjectInstanceHandle
 * @throws ObjectClassNotDefined if theClass isn't recognized by the RTI
 * @throws ObjectClassNotPublished if theClass isn't published by the federate
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws AttributeNotPublished if an attribute is not published by the federate
 * @throws InvalidRegion if the RegionHandle is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #registerObjectInstance registerObjectInstance
 * @see #deleteObjectInstance deleteObjectInstance
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 */
public ObjectInstanceHandle registerObjectInstanceWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions)
    throws ObjectClassNotDefined,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Registers a new instance of the specified object class with the RTI, using a previously reserved name and simultaneously associates its attributes with distribution regions.
 * @param theClass the ObjectClassHandle of the object instance being registered
 * @param attributesAndRegions an AttributeSetRegionSetPairList listing the attributes and regions being simultaneously associated
 * @param theObject a String holding the previously reserved object instance name
 * @return the registered object instance's ObjectInstanceHandle
 * @throws ObjectClassNotDefined if theClass isn't recognized by the RTI
 * @throws ObjectClassNotPublished if theClass isn't published by the federate
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws AttributeNotPublished if an attribute is not published by the federate
 * @throws InvalidRegion if the RegionHandle is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute
 * @throws ObjectInstanceNameNotReserved if theObject has not been previously reserved
 * @throws ObjectInstanceNameInUse if theObject is already in use
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see registerObjectInstance
 * @see reserveObjectInstanceName
 * @see deleteObjectInstance
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 */
public ObjectInstanceHandle
registerObjectInstanceWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions,
    String theObject)
throws ObjectClassNotDefined,
    ObjectClassNotPublished,
    AttributeNotDefined,
    AttributeNotPublished,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    ObjectInstanceNameNotReserved,
    ObjectInstanceNameInUse,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
**Associates distribution regions with the specified object instance attributes.**

@param theObject the `ObjectInstanceHandle` of the object instance

@throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered theObject

@throws AttributeNotDefined if an attribute could not be recognized within the supplied context

@throws InvalidRegion if the `RegionHandle` is invalid or the region is still a template

@throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set

@throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute

@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution

@throws SaveInProgress if the federate is in one of the save-in-progress states

@throws RestoreInProgress if the federate is in one of the restore-in-progress states

@throws RTIinternalError if something else goes wrong

@see #unassociateRegionsForUpdates unassociateRegionsForUpdates

*  

```java
public void associateRegionsForUpdates(
    ObjectInstanceHandle theObject,
    AttributeSetRegionSetPairList attributesAndRegions)
    throws ObjectInstanceNotKnown,
            AttributeNotDefined,
            InvalidRegion,
            RegionNotCreatedByThisFederate,
            InvalidRegionContext,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
```
/**
 * Unassociates distribution regions from the specified object instance attributes.
 * <p>
 * Regions must be unassociated from any distributions and subscriptions before they can be deleted.
 * @param theObject the {ObjectInstanceHandle} of the object instance
 * @param attributesAndRegions an {AttributeSetRegionSetPairList} listing the attributes and regions to unassociate
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws InvalidRegion if the {RegionHandle} is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #associateRegionsForUpdates associateRegionsForUpdates
 */

public void
unassociateRegionsForUpdates(
    ObjectInstanceHandle theObject,
    AttributeSetRegionSetPairList attributesAndRegions) throws ObjectInstanceNotKnown,
        AttributeNotDefined,
        InvalidRegion,
        RegionNotCreatedByThisFederate,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
public void subscribeObjectClassAttributesWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions)
throws ObjectClassNotDefined,
    AttributeNotDefined,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
/**
 * Subscribes the federate to certain attributes of an object class within specified regions,
 * without tripping the publishers' object class relevance advisories.
 * @param theClass the [ObjectClassHandle] of the subscribed object class
 * @param attributesAndRegions an [AttributeSetRegionSetPairList] listing the attributes and regions being simultaneously associated
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws InvalidRegion if the [RegionHandle] is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see subscribeObjectClassAttributes
 * @see subscribeObjectClassAttributesPassively
 * @see subscribeObjectClassAttributesWithRegions
 * @see unsubscribeObjectClass
 * @see unsubscribeObjectClassAttributes
 * @see unsubscribeObjectClassAttributesWithRegions
 */
public void subscribeObjectClassAttributesPassivelyWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions)
throws ObjectClassNotDefined,
    AttributeNotDefined,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
Unsubscribes the federate from certain attributes of an object class within specified regions.

@param theClass the ObjectClassHandle of the unsubscribed object class
@param attributesAndRegions an AttributeSetRegionSetPairList listing the attributes and regions being simultaneously unassociated

@throws ObjectClassNotDefined if theClass isn't recognized by the RTI
@throws AttributeNotDefined if an attribute could not be recognized within the supplied context
@throws InvalidRegion if the RegionHandle is invalid or the region is still a template
@throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
@throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
@throws SaveInProgress if the federate is in one of the save-in-progress states
@throws RestoreInProgress if the federate is in one of the restore-in-progress states
@throws RTIinternalError if something else goes wrong

@see #subscribeObjectClassAttributes
@see #subscribeObjectClassAttributesPassively
@see #subscribeObjectClassAttributesWithRegions
@see #subscribeObjectClassAttributesPassivelyWithRegions
@see #unsubscribeObjectClass
@see #unsubscribeObjectClassAttributes

```
public void unsubscribeObjectClassAttributesWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions)
throws ObjectClassNotDefined,
        AttributeNotDefined,
        InvalidRegion,
        RegionNotCreatedByThisFederate,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;
```
// 9.10
/**
 * Subscribes the federate to an interaction within specified regions.
 * @param theClass the {@link InteractionClassHandle} of the subscribed interaction
 * @param regions the {@link RegionHandleSet} describing the subscription regions
 * @throws InteractionClassNotDefined if <code>theClass</code> isn't recognized by the RTI
 * @throws InvalidRegion if the {@link RegionHandle} is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute
 * @throws FederateServiceInvocationsAreBeingReportedViaMOM if service invocations are currently being reported via MOM interactions and <code>theClass</code> is <code>HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation</code>
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeInteractionClass subscribeInteractionClass
 * @see #subscribeInteractionClassPassively subscribeInteractionClassPassively
 * @see #subscribeInteractionClassPassivelyWithRegions subscribeInteractionClassPassivelyWithRegions
 * @see #unsubscribeInteractionClass unsubscribeInteractionClass
 */
public void
subscribeInteractionClassWithRegions(
    InteractionClassHandle theClass,
    RegionHandleSet regions)
throws InteractionClassNotDefined,
        InvalidRegion,
        RegionNotCreatedByThisFederate,
        InvalidRegionContext,
        FederateServiceInvocationsAreBeingReportedViaMOM,
        FederateNotExecutionMember,
        SaveInProgress,
        RestoreInProgress,
        RTIinternalError;

/**
 * Subscribes the federate to an interaction within specified regions,
 * without tripping the publishers' interaction relevance advisories.
 * @param theClass the InteractionClassHandle of the subscribed interaction
 * @param regions the RegionHandleSet describing the subscription regions
 * @throws InteractionClassNotDefined if theClass isn't recognized by the RTI
 * @throws InvalidRegion if the RegionHandle is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the regions are not a subset of the available dimensions of some attribute
 * @throws FederateServiceInvocationsAreBeingReportedViaMOM if service invocations are currently being reported via MOM interactions and theClass is HLAManager.HLAFederate.HLAreport.HLAreportServiceInvocation
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see subscribeInteractionClass
 * @see subscribeInteractionClassPassively
 * @see subscribeInteractionClassWithRegions
 * @see unsubscribeInteractionClass
 */

public void subscribeInteractionClassPassivelyWithRegions(
    InteractionClassHandle theClass,
    RegionHandleSet regions)
throws InteractionClassNotDefined,
InvalidRegion,
RegionNotCreatedByThisFederate,
InvalidRegionContext,
FederateServiceInvocationsAreBeingReportedViaMOM,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
/**
 * Unsubscribes the federate from an interaction within specified regions.
 * @param theClass the InteractionClassHandle of the unsubscribed interaction
 * @param regions the RegionHandleSet describing the subscription regions
 * @throws InteractionClassNotDefined if theClass isn't recognized by the RTI
 * @throws InvalidRegion if the RegionHandle is invalid or the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not own (did not create) one of the regions in the set
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #subscribeInteractionClassWithRegions
 */

public void unsubscribeInteractionClassWithRegions(
    InteractionClassHandle theClass,
    RegionHandleSet regions)
    throws InteractionClassNotDefined,
            InvalidRegion,
            RegionNotCreatedByThisFederate,
            FederateNotExecutionMember,
            SaveInProgress,
            RestoreInProgress,
            RTIinternalError;
//9.12
/**
 * Sends an interaction within specified regions.
 * @param theInteraction the {@link InteractionClassHandle} of the
 * interaction being sent
 * @param theParameters a {@link ParameterHandleValueMap} holding
 * the interaction values, keyed by parameter
 * @param regions the {@link RegionHandleSet} describing the
 * distribution regions
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws InteractionClassNotDefined if
 * theInteraction isn't recognized by the RTI
 * @throws InteractionClassNotPublished if the federate does not
 * publish theInteraction
 * @throws InteractionParameterNotDefined if one of
 * theParameters isn't recognized in the supplied context
 * @throws InvalidRegion if the RegionHandle is invalid or
 * the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not
 * own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the
 * regions are not a subset of the available dimensions of some attribute
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #sendInteraction
 * @see FederateAmbassador#receiveInteraction(InteractionClassHandle,Parameter
 * HandleValueMap,byte[],OrderType,TransportationType,RegionHandleSet)
 */

public void
sendInteractionWithRegions(
    InteractionClassHandle theInteraction,
    ParameterHandleValueMap theParameters,
    RegionHandleSet regions,
    byte[] userSuppliedTag)
    throws InteractionClassNotDefined,
           InteractionClassNotPublished,
           InteractionParameterNotDefined,
           InvalidRegion,
           RegionNotCreatedByThisFederate,
           InvalidRegionContext,
           FederateNotExecutionMember,
           SaveInProgress,
           RestoreInProgress,
           RTIinternalError;
/**
 * Sends a time-stamped interaction within specified regions.
 * @param theInteraction the {@link InteractionClassHandle} of the
 * interaction being sent
 * @param theParameters a {@link ParameterHandleValuePairMap} holding
 * the interaction values, keyed by parameter
 * @param regions the {@link RegionHandleSet} describing the
distribution regions
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @param theTime the {@link LogicalTime} at which the interaction
 * occurs
 * @return the message's {@link MessageRetractionReturn}, should a
 * retraction become necessary
 * @throws InvalidLogicalTime if the specified
 * <code>LogicalTime</code> is invalid
 * @throws InteractionClassNotDefined if
 * <code>theInteraction</code> isn't recognized by the RTI
 * @throws InteractionClassNotPublished if the federate does not
 * publish <code>theInteraction</code>
 * @throws InteractionParameterNotDefined if one of
 * <code>theParameters</code> isn't recognized in the supplied context
 * @throws InvalidRegion if the {@link RegionHandle} is invalid or
 * the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not
 * own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the
 * regions are not a subset of the available dimensions of some attribute
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #sendInteraction sendInteraction
 * @see FederateAmbassador#receiveInteraction(InteractionClassHandle,Parameter
 * HandleValuePairMap,byte[],OrderType,TransportationType,LogicalTime,OrderTy
 * pe,RegionHandleSet)
 * @see #retract retract
 */
public MessageRetractionReturn
sendInteractionWithRegions(
    InteractionClassHandle theInteraction,
    ParameterHandleValueMap theParameters,
    RegionHandleSet regions,
    byte[] userSuppliedTag,
    LogicalTime theTime)
throws InteractionClassNotDefined,
    InteractionClassNotPublished,
    InteractionParameterNotDefined,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    InvalidLogicalTime,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
// 9.13
/**
 * Requests that attribute value updates be provided for the
 * specified instance attributes
 * if they fall within the specified regions.
 * @param theClass the {@link ObjectClassHandle} of the object
 * class being polled
 * @param attributesAndRegions an {@link AttributeSetRegionSetPairList} listing the attributes and regions
 * being polled
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws ObjectClassNotDefined if <code>theClass</code> isn't
 * recognized by the RTI
 * @throws AttributeNotDefined if an attribute could not be
 * recognized within the supplied context
 * @throws InvalidRegion if the {@link RegionHandle} is invalid or
 * the region is still a template
 * @throws RegionNotCreatedByThisFederate if the federate does not
 * own (did not create) one of the regions in the set
 * @throws InvalidRegionContext if the dimensions of one of the
 * regions are not a subset of the available dimensions of some attribute
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-
 * progress states
 * @throws RestoreInProgress if the federate is in one of the
 * restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #requestAttributeValueUpdate requestAttributeValueUpdate
 */
public void requestAttributeValueUpdateWithRegions(
    ObjectClassHandle theClass,
    AttributeSetRegionSetPairList attributesAndRegions,
    byte[] userSuppliedTag)
    throws ObjectClassNotDefined,
    AttributeNotDefined,
    InvalidRegion,
    RegionNotCreatedByThisFederate,
    InvalidRegionContext,
    FederateNotExecutionMember,
    SaveInProgress,
    RestoreInProgress,
    RTIinternalError;
public ObjectClassHandle
getObjectClassHandle(String theName)
throws NameNotFound,
FederateNotExecutionMember,
RTIinternalError;

public String
getObjectClassName(ObjectClassHandle theHandle)
throws InvalidObjectClassHandle,
FederateNotExecutionMember,
RTIinternalError;
/**
 * Requests the {@link AttributeHandle} of the specified class' attribute bearing the specified name.
 * @param whichClass the {@link ObjectClassHandle} of the class to which the attribute belongs
 * @param theName a {@link java.lang.String} holding the attribute name
 * @return the requested {@link AttributeHandle}
 * @throws InvalidObjectClassHandle if the {@link ObjectClassHandle} was not recognized
 * @throws NameNotFound if the name could not be found in the FOM Document Data
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getAttributeName
 */

public AttributeHandle
getAttributeHandle(String whichClass, String theName)
throws InvalidObjectClassHandle, NameNotFound, FederateNotExecutionMember, RTIinternalError;

/**
 * Requests the name of the specified attribute of the specified class.
 * @param whichClass the {@link ObjectClassHandle} of the class to which the attribute belongs
 * @param theHandle an {@link AttributeHandle}
 * @return the requested name as a {@link java.lang.String}
 * @throws InvalidObjectClassHandle if the {@link ObjectClassHandle} was not recognized
 * @throws InvalidAttributeHandle if the {@link AttributeHandle} was not recognized within the specified context
 * @throws AttributeNotDefined if an attribute could not be recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getAttributeHandle
 */

public String
getAttributeName(ObjectClassHandle whichClass, AttributeHandle theHandle)
throws InvalidObjectClassHandle, InvalidAttributeHandle, AttributeNotDefined, FederateNotExecutionMember, RTIinternalError;
public InteractionClassHandle
getInteractionClassHandle(
String theName)
throws NameNotFound,
FederateNotExecutionMember,
RTIinternalError;

public String
getInteractionClassName(
InteractionClassHandle theHandle)
throws InvalidInteractionClassHandle,
FederateNotExecutionMember,
RTIinternalError;
// 10.8
/**
 * Requests the \{@link ParameterHandle\} of the interaction class' parameter bearing the specified (fully qualified) name.
 * @param whichClass the \{@link InteractionClassHandle\} to which the parameter belongs
 * @param theName a \{@link java.lang.String\} holding the name of the parameter
 * @return the requested \{@link parameterHandle\}
 * @throws InvalidInteractionClassHandle if the \{@link InteractionClassHandle\} was not recognized
 * @throws NameNotFound if the name could not be found in the FOM
 * Document Data
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getParameterName getParameterName
 */
public ParameterHandle
getParameterHandle(
    InteractionClassHandle whichClass,
    String theName)
throws InvalidInteractionClassHandle,
    NameNotFound,
    FederateNotExecutionMember,
    RTIinternalError;

// 10.9
/**
 * Requests the name of the interaction class' parameter bearing the specified \{@link ParameterHandle\}.
 * @param whichClass the \{@link InteractionClassHandle\} to which the parameter belongs
 * @param theHandle the parameter's \{@link ParameterHandle\}
 * @return the requested name as a \{@link java.lang.String\}
 * @throws InvalidInteractionClassHandle if the \{@link InteractionClassHandle\} was not recognized
 * @throws InvalidParameterHandle if the \{@link parameterHandle\} was not recognized within the specified context
 * @throws InteractionParameterNotDefined if \texttt{theHandle} isn't recognized in the supplied context
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getParameterHandle getParameterHandle
 */
public String
getParameterName(
    InteractionClassHandle whichClass,
    ParameterHandle theHandle)
throws InvalidInteractionClassHandle,
    InvalidParameterHandle,
    InteractionParameterNotDefined,
    FederateNotExecutionMember,
    RTIinternalError;
/**
 * Requests the [ObjectInstanceHandle] of the object instance bearing the specified name.
 * @param theName a [java.lang.String] holding the name of the object instance
 * @return the requested [ObjectInstanceHandle]
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getObjectInstanceName
 */

public ObjectInstanceHandle
getObjectInstanceHandle(
    String theName)
throws ObjectInstanceNotKnown,
    FederateNotExecutionMember,
    RTIinternalError;


/**
 * Requests the name of the object instance bearing the specified [ObjectInstanceHandle].
 * @param theHandle the [ObjectInstanceHandle] of the object instance
 * @return the requested name as a [java.lang.String]
 * @throws ObjectInstanceNotKnown if the RTI considers that the federate has not discovered <code>theObject</code>
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getObjectInstanceHandle
 */

public String
getObjectInstanceName(
    ObjectInstanceHandle theHandle)
throws ObjectInstanceNotKnown,
    FederateNotExecutionMember,
    RTIinternalError;
public DimensionHandle getDimensionHandle(String theName)
    throws NameNotFound, FederateNotExecutionMember, RTIinternalError;

public String getDimensionName(DimensionHandle theHandle)
    throws InvalidDimensionHandle, FederateNotExecutionMember, RTIinternalError;
// 10.14
/**
 * Requests the upper bound of the specified dimension. All
dimensions have zero as their lower bound.
 * @param theHandle the {link DimensionHandle} of the dimension
 * @return the requested upper bound as a <code>long</code>
 * @throws InvalidDimensionHandle if one of the {link
 * DimensionHandle}s is invalid
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public long
defineDimensionUpperBound(
    DimensionHandle theHandle)
throws InvalidDimensionHandle,
    FederateNotExecutionMember,
    RTIinternalError;

// 10.15
/**
 * Requests the available dimensions of the specified class
attribute.
 * @param whichClass the {link ObjectClassHandle}
 * @param theHandle the {link AttributeHandle}
 * @return the requested {link DimensionHandleSet}
 * @throws InvalidObjectClassHandle if the {link
ObjectClassHandle} was not recognized
 * @throws InvalidAttributeHandle if the {link AttributeHandle}
was not recognized within the specified context
 * @throws AttributeNotDefined if an attribute could not be
recognized within the supplied context
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public DimensionHandleSet
getAvailableDimensionsForClassAttribute(
    ObjectClassHandle whichClass,
    AttributeHandle theHandle)
throws InvalidObjectClassHandle,
    InvalidAttributeHandle,
    AttributeNotDefined,
    FederateNotExecutionMember,
    RTIinternalError;
// 10.16
/**
 * Requests the {link ObjectClassHandle} of the class the
 * specified object instance is known under.
 * @param theObject the {link ObjectInstanceHandle} of the object
 * instance
 * @return the requested {link ObjectClassHandle}
 * @throws ObjectInstanceNotKnown if the RTI considers that the
 * federate has not discovered <code>theObject</code>
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getObjectClassHandle
 * @see #getObjectClassName
 */
public ObjectClassHandle
getKnownObjectClassHandle(
    ObjectInstanceHandle theObject)
throws ObjectInstanceNotKnown,
    FederateNotExecutionMember,
    RTIinternalError;

// 10.17
/**
 * Requests the available dimensions of the specified interaction.
 * @param theHandle the {link InteractionClassHandle}
 * @return the requested {link DimensionHandleSet}
 * @throws InvalidInteractionClassHandle if the {link
 * InteractionClassHandle} was not recognized
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public DimensionHandleSet
getAvailableDimensionsForInteractionClass(
    InteractionClassHandle theHandle)
throws InvalidInteractionClassHandle,
    FederateNotExecutionMember,
    RTIinternalError;
// 10.18
/**
 * Requests the [@link TransportationType] bearing the specified
 * name, as defined in the FDD Transportation Table.
 * @param theName a [@link java.lang.String] holding the
 * transportation type’s name
 * @return the requested [@link TransportationType]
 * @throws InvalidTransportationName if the name is not recognized
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getTransportationName getTransportationName
 */
public TransportationType
getTransportationType(
    String theName)
throws InvalidTransportationName,
    FederateNotExecutionMember,
    RTIinternalError;

// 10.19
/**
 * Requests the name of the specified [@link TransportationType],
 * as defined in the FDD Transportation Table.
 * @param theType the [@link TransportationType] for which to
 * return the name
 * @return the requested name as a [@link java.lang.String]
 * @throws InvalidTransportationType if the [@link TransportationType] is not recognized
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 * @see #getTransportationType getTransportationType
 */
public String
getTransportationName(
    TransportationType theType)
throws InvalidTransportationType,
    FederateNotExecutionMember,
    RTIinternalError;
// 10.20
/**
 * Requests the {@link OrderType} bearing the specified name, as
defined in the IEEE 1516 specification.
 * @param theName a {@link String} holding the ordering type’s name
 * @return the requested {@link OrderType}
 * @throws InvalidOrderName if the name is not recognized
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public OrderType
getOrderType(String theName)
throws InvalidOrderName,
FederateNotExecutionMember,
RTIinternalError;

// 10.21
/**
 * Requests the name of the specified {@link OrderType}, as defined
in the IEEE 1516 specification.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 changes the parameter
from "theHandle" to "theType".
 * @param theType the {@link OrderType} for which to return the
name
 * @return the requested name as a {@link String}
 * @throws InvalidOrderType if the {@link OrderType} is not
recognized
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public String
getOrderName(OrderType theType)
throws InvalidOrderType,
FederateNotExecutionMember,
RTIinternalError;
// 10.22
/**
 * Turns the object class relevance advisory switch on for the federate.
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws ObjectClassRelevanceAdvisorySwitchIsOn if the advisory switch is already on
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableObjectClassRelevanceAdvisorySwitch
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 */
public void enableObjectClassRelevanceAdvisorySwitch()
throws FederateNotExecutionMember,
          ObjectClassRelevanceAdvisorySwitchIsOn,
          SaveInProgress,
          RestoreInProgress,
          RTIinternalError;

// 10.23
/**
 * Turns the object class relevance advisory switch off for the federate.
 * @throws ObjectClassRelevanceAdvisorySwitchIsOff if the advisory switch is already off
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #enableObjectClassRelevanceAdvisorySwitch
 * @see FederateAmbassador#startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 */
public void disableObjectClassRelevanceAdvisorySwitch()
throws ObjectClassRelevanceAdvisorySwitchIsOff,
          FederateNotExecutionMember,
          SaveInProgress,
          RestoreInProgress,
          RTIinternalError;
// 10.24
/**
 * Turns the attribute relevance advisory switch on for the federate.
 * @throws AttributeRelevanceAdvisorySwitchIsOn if the advisory switch is already on
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableAttributeRelevanceAdvisorySwitch
 */
public void enableAttributeRelevanceAdvisorySwitch()
throws AttributeRelevanceAdvisorySwitchIsOn,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;

// 10.25
/**
 * Turns the attribute relevance advisory switch off for the federate.
 * @throws AttributeRelevanceAdvisorySwitchIsOff if the advisory switch is already off
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #enableAttributeRelevanceAdvisorySwitch
 */
public void disableAttributeRelevanceAdvisorySwitch()
throws AttributeRelevanceAdvisorySwitchIsOff,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
// 10.26
/**
 * Turns the attribute scope advisory switch on for the federate.
 * @throws AttributeScopeAdvisorySwitchIsOn if the advisory switch is already on
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #disableAttributeScopeAdvisorySwitch
 * @see FederateAmbassador#attributesOutOfScope
 * @see FederateAmbassador#attributesInScope
 */
public void
enableAttributeScopeAdvisorySwitch()
throws AttributeScopeAdvisorySwitchIsOn,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;

// 10.27
/**
 * Turns the attribute scope advisory switch off for the federate.
 * @throws AttributeScopeAdvisorySwitchIsOff if the advisory switch is already off
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see #enableAttributeScopeAdvisorySwitch
 * @see FederateAmbassador#attributesOutOfScope
 * @see FederateAmbassador#attributesInScope
 */
public void
disableAttributeScopeAdvisorySwitch()
throws AttributeScopeAdvisorySwitchIsOff,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
**// 10.28**

```
**
* Turns the interaction relevance advisory switch on for the federate.
* @throws InteractionRelevanceAdvisorySwitchIsOn if the advisory switch is already on
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #disableInteractionRelevanceAdvisorySwitch
disableInteractionRelevanceAdvisorySwitch
* @see FederateAmbassador#turnInteractionsOn turnInteractionsOn
* @see FederateAmbassador#turnInteractionsOff turnInteractionsOff
*/
public void
enableInteractionRelevanceAdvisorySwitch() throws InteractionRelevanceAdvisorySwitchIsOn, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
```

**// 10.29**

```
**
* Turns the interaction relevance advisory switch off for the federate.
* @throws InteractionRelevanceAdvisorySwitchIsOff if the advisory switch is already off
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #enableInteractionRelevanceAdvisorySwitch
disableInteractionRelevanceAdvisorySwitch
* @see FederateAmbassador#turnInteractionsOn turnInteractionsOn
* @see FederateAmbassador#turnInteractionsOff turnInteractionsOff
*/
public void
disableInteractionRelevanceAdvisorySwitch() throws InteractionRelevanceAdvisorySwitchIsOff, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
```
// 10.30
/**
 * Requests the [@link DimensionHandleSet] of the specified region.
 * The specified region must either have been [@link #createRegion created] by the invoking federate
 * or conveyed to it in a [@link RegionHandleSet] callback argument.
 * @param region the [@link RegionHandle] of the region template, specification, or realization
 * @return A [@link DimensionHandleSet] specifying the dimensions of the region
 * @throws InvalidRegion if the [@link RegionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws SaveInProgress if the federate is in one of the save-in-progress states
 * @throws RestoreInProgress if the federate is in one of the restore-in-progress states
 * @throws RTIinternalError if something else goes wrong
 * @see FederateAmbassador#reflectAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],OrderType,TransportationType,RegionHandleSet)
 * @see FederateAmbassador#reflectAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,RegionHandleSet)
 * @see FederateAmbassador#receiveInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],OrderType,TransportationType,RegionHandleSet)
 * @see FederateAmbassador#receiveInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet)
 */
public DimensionHandleSet
getDimensionHandleSet(
RegionHandle region)
throws InvalidRegion,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
public RangeBounds
getRangeBounds(
    RegionHandle region,
    DimensionHandle dimension)
throws InvalidRegion,
RegionDoesNotContainSpecifiedDimension,
FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
public void setRangeBounds(RegionHandle region, DimensionHandle dimension, RangeBounds bounds)
    throws InvalidRegion, RegionNotCreatedByThisFederate, RegionDoesNotContainSpecifiedDimension, InvalidRangeBound, FederateNotExecutionMember, SaveInProgress, RestoreInProgress, RTIinternalError;
// 10.33
/**
 * Projects a {@link FederateHandle} onto the <code>Federates</code> dimension (defined by the Management Object Model (MOM)) for <code>Region</code> specification purposes.
 * @param federateHandle the <code>FederateHandle</code> to project onto the <code>Federates</code> dimension
 * @return a <code>long</code> representing the federate's coordinate along the <code>Federates</code> dimension
 * @throws InvalidFederateHandle if the specified federate handle is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public long normalizeFederateHandle(
    FederateHandle federateHandle)
throws InvalidFederateHandle,
    FederateNotExecutionMember,
    RTIinternalError;

// 10.34
/**
 * Projects a {@link serviceGroup} onto the <code>ServiceGroups</code> dimension (defined by the Management Object Model (MOM)) for <code>Region</code> specification purposes.
 * @param group the <code>ServiceGroup</code> to project onto the <code>ServiceGroups</code> dimension
 * @return a <code>long</code> representing the group's coordinate along the <code>ServiceGroups</code> dimension
 * @throws InvalidServiceGroup if the specified service group designator is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong
 */
public long normalizeServiceGroup(
    ServiceGroup group)
throws InvalidServiceGroup,
    FederateNotExecutionMember,
    RTIinternalError;
In the DoD Interpretations of IEEE 1516-2000v2, this service is deleted, along with the InitializePreviouslyInvoked exception (which it was the only one to throw). @throws RTIinternalError if something else goes wrong

```
//public java.util.Properties
initializeRTI(
    java.util.Properties properties)
//throws InitializePreviouslyInvoked,
//BadInitializationParameter,
//RTIinternalError;
```

In the DoD Interpretations of IEEE 1516-2000v2, this service is deleted, along with the InitializeNeverInvoked and SomeFederateJoinedToAnExecution exceptions (which it was the only one to throw). @throws RTIinternalError if something else goes wrong

```
//public void
finalizeRTI()
//throws InitializeNeverInvoked,
//SomeFederateJoinedToAnExecution,
//RTIinternalError;
```

Requests that the RTI invoke a single federate callback or, if there are none pending, that the method time out after the specified real time has elapsed. If callbacks are disabled, the method times out after the specified real-time interval has elapsed but nevertheless returns <code>true</code> even though no callbacks are invoked. @param seconds A <code>double</code> specifying the time out in real-time seconds (with a precision of at least 1 ms) @return <code>true</code> iff there was at least one callback pending @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution @throws RTIinternalError if something else goes wrong

```
public boolean
 evokeCallback(
    double seconds)
throws FederateNotExecutionMember,
    RTIinternalError;
```
public boolean evokeMultipleCallbacks(
    double minimumTime,
    double maximumTime)
throws FederateNotExecutionMember, RTIinternalError;
** 10.39

* Instructs the RTI to deliver callbacks when the [@link #evokeCallback evokeCallback} and [* [@link #evokeMultipleCallbacks evokeMultipleCallbacks] methods are invoked. This is the default federate state.
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #disableCallbacks disableCallbacks
* @see #evokeCallback evokeCallback
* @see #evokeMultipleCallbacks evokeMultipleCallbacks
*
public void
enableCallbacks()
throws FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;

** 10.40

* Instructs the RTI to withhold callbacks when the [@link #evokeCallback evokeCallback} and [* [@link #evokeMultipleCallbacks evokeMultipleCallbacks] methods are invoked. These methods then time out as instructed but nevertheless return <code>true</code> to indicate callbacks are pending.
* @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
* @throws SaveInProgress if the federate is in one of the save-in-progress states
* @throws RestoreInProgress if the federate is in one of the restore-in-progress states
* @throws RTIinternalError if something else goes wrong
* @see #enableCallbacks enableCallbacks
* @see #evokeCallback evokeCallback
* @see #evokeMultipleCallbacks evokeMultipleCallbacks
*
public void
disableCallbacks()
throws FederateNotExecutionMember,
SaveInProgress,
RestoreInProgress,
RTIinternalError;
/**
 * Supplies the RTIambassador's {@link AttributeHandleFactory}, which can be used to
decode callback arguments.
 *<p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
to each of the get*Factory services.
 * @return The RTIambassador's {@link AttributeHandleFactory}
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 */
public AttributeHandleFactory
getAttributeHandleFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's {@link AttributeHandleSetFactory}, which can be used to
prepare method arguments.
 *<p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
to each of the get*Factory services.
 * @return The RTIambassador's {@link AttributeHandleSetFactory}
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 */
public AttributeHandleSetFactory
getAttributeHandleSetFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's {@link AttributeHandleValueMapFactory}, which can be used to
prepare method arguments.
 *<p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
to each of the get*Factory services.
 * @return The RTIambassador's {@link AttributeHandleValueMapFactory}
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 */
public AttributeHandleValueMapFactory
getAttributeHandleValueMapFactory()
throws FederateNotExecutionMember;
/**
 * Supplies the RTIambassador's AttributeSetRegionSetPairListFactory, which can be used to
 * prepare method arguments.
 * @return The RTIambassador's AttributeSetRegionSetPairListFactory
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */
public AttributeSetRegionSetPairListFactory
getAttributeSetRegionSetPairListFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's DimensionHandleFactory, which can be used to
 * decode callback arguments.
 * @return The RTIambassador's DimensionHandleFactory
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */
public DimensionHandleFactory
getDimensionHandleFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's DimensionHandleSetFactory, which can be used to
 * prepare method arguments.
 * @return The RTIambassador's DimensionHandleSetFactory
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */
public DimensionHandleSetFactory
getDimensionHandleSetFactory()
throws FederateNotExecutionMember;
/**
 * Supplies the RTIambassador's {@link FederateHandleFactory},
 * which can be used to
 * decode callback arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws
 * FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's {@link FederateHandleFactory}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 */
 public FederateHandleFactory
 getFederateHandleFactory()
 throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's {@link FederateHandleSetFactory},
 * which can be used to
 * prepare method arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws
 * FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's {@link FederateHandleSetFactory}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 */
 public FederateHandleSetFactory
 getFederateHandleSetFactory()
 throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's {link InteractionClassHandleFactory},
 * which can be used to
 * decode callback arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws
 * FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's {link InteractionClassHandleFactory}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 */
 public InteractionClassHandleFactory
 getInteractionClassHandleFactory()
 throws FederateNotExecutionMember;
/**
 * Supplies the RTIambassador's [ObjectClassHandleFactory], which can be used to
 * decode callback arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's [ObjectClassHandleFactory]
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */

public ObjectClassHandleFactory
getObjectClassHandleFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's [ObjectInstanceHandleFactory], which can be used to
 * decode callback arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's [ObjectInstanceHandleFactory]
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */

public ObjectInstanceHandleFactory
getObjectInstanceHandleFactory()
throws FederateNotExecutionMember;

/**
 * Supplies the RTIambassador's [ParameterHandleFactory], which can be used to
 * decode callback arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws FederateNotExecutionMember"
 * to each of the get*Factory services.
 * @return The RTIambassador's [ParameterHandleFactory]
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 */

public ParameterHandleFactory
getParameterHandleFactory()
throws FederateNotExecutionMember;
/**
 * Supplies the RTIambassador's [link ParameterHandleValueMapFactory]{@link}, which can be used to
 * prepare method arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws
 * FederateNotExecutionMember" to each of the get*Factory services.
 * @return The RTIambassador's [link ParameterHandleValueMapFactory]{@link}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * */
 public ParameterHandleValueMapFactory
 getParameterHandleValueMapFactory()
 throws FederateNotExecutionMember;

 /**
 * Supplies the RTIambassador's [link RegionHandleSetFactory]{@link}, which can be used to
 * prepare method arguments.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 added "throws
 * FederateNotExecutionMember" to each of the get*Factory services.
 * @return The RTIambassador's [link RegionHandleSetFactory]{@link}
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * */
 public RegionHandleSetFactory
 getRegionHandleSetFactory()
 throws FederateNotExecutionMember;

 /**
 * Identifies the RTI's version.
 * @return The [link java.lang.String] "1516.1.5"
 */
 public String
 getHLAversion();

 //end RTIambassador
// File: SaveFailureReason.java
package hla.rti1516; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java \{@link java.util Enumeration\})
 * representing the reason why the save operation of a federate failed.
 * It is reported by the \{@link FederateAmbassador#federationNotSaved
 federationNotSaved\} callback.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see hla.rti1516.FederateAmbassador#federationNotSaved
 */
public final class SaveFailureReason
    implements java.io.Serializable
{
    //each instance's value
    private int _value;
    //initial value for enumeration
    private static final int _lowestValue = 1;
    //begins at lowest
    private static int _nextToAssign = _lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a SaveFailureReason must be
     * initialized with one of the defined static values.
     * @param otherSaveFailureReasonValue must be a defined static
     * value or another instance.
     */
    public SaveFailureReason(SaveFailureReason otherSaveFailureReasonValue)
    {
        _value = otherSaveFailureReasonValue._value;
    }

    /**
     * Package-only (default access) constructor. Unused.
     * @param value to assign to the instance; must be one of the
     * static ones
     */
    public SaveFailureReason(int value)
        throws RTIinternalError
    {
        _value = value;
        if ((value < _lowestValue) || (value >= _nextToAssign))
            throw new RTIinternalError("SaveFailureReason: illegal value
" + value);
    }
}
/**
 * Private constructor; it is used to generate the static values.
 */
private
SaveFailureReason()
{
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the <code>SaveFailureReason</code>.
 * @return A <code>String</code> with value "SaveFailureReason(n)" where n is this value
 */
public String
toString()
{
    return "SaveFailureReason(" + _value + ")";
}

/**
 * Returns true iff this and otherSaveFailureReasonValue represent the same save failure reason.
 * @param otherSaveFailureReasonValue The Object to compare with
 * @return true iff supplied otherSaveFailureReasonValue is of type <code>SaveFailureReason</code> and has same value
 */
public boolean
equals(Object otherSaveFailureReasonValue)
{
    if (otherSaveFailureReasonValue instanceof SaveFailureReason)
        return _value == ((SaveFailureReason)otherSaveFailureReasonValue)._value;
    else
        return false;
}

/**
 * Returns a hash code for this; two <code>SaveFailureReason</code>s for which equals() is true should yield the same hash code.
 */
public int
hashCode()
{
    return _value;
}
The constant instances

* The RTI was unable to save.

```java
static public final SaveFailureReason RTI_UNABLE_TO_SAVE = new SaveFailureReason();
```

* One or more joined federates have invoked the `federateSaveNotComplete` method.

```java
static public final SaveFailureReason FEDERATE_REPORTED_FAILURE = new SaveFailureReason();
```

* One or more joined federates have resigned from the federation execution.

```java
static public final SaveFailureReason FEDERATE_RESIGNED = new SaveFailureReason();
```

* The RTI has detected failure at one or more of the joined federates.

```java
static public final SaveFailureReason RTI_DETECTED_FAILURE = new SaveFailureReason();
```

* The time stamp specified by the `requestFederationSave(String,LogicalTime)` method cannot be honored, due to possible race conditions in the distributed calculation of GALT (Greatest Available Logical Time).

```java
static public final SaveFailureReason SAVE_TIME_CANNOT_BE_HONORED = new SaveFailureReason();
```
package hla.rti\texttt{516}; //the package name was changed by DoD Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java \texttt{\{\noindent java.util.Enumeration\}}) representing the save status of a federate during a federation save operation.
 * It is contained in the \texttt{\{\noindent FederateHandleSaveStatusPair\}} argument of the \texttt{\{\noindent FederateAmbassador\#federationSaveStatusResponse\}} callback.
 * @author IEEE
 * @version 1516.1.5 (DoD V2)
 */
public final class SaveStatus
    implements java.io.Serializable
{
    //each instance's value
    private int \_value;
    //initial value for enumeration
    private static final int \_lowestValue = 1;
    //begins at lowest
    private static int \_nextToAssign = \_lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a \texttt{\{\noindent SaveStatus\}} must be initialized with one of the defined static values.
     * @param otherSaveStatusValue must be a defined static value or another instance.
     */
    public SaveStatus(SaveStatus otherSaveStatusValue)
    {
        \_value = otherSaveStatusValue.\_value;
    }

    /**
     * Package-only (default access) constructor. Unused.
     * @param value to assign to the instance; must be one of the static ones
     */
    SaveStatus(int value)
        throws RTIinternalError
    {
        \_value = value;
        if ((value < \_lowestValue) || (value >= \_nextToAssign))
            throw new RTIinternalError("SaveStatus: illegal value " + value);
    }
}
/**
 * Private constructor; it is used to generate the static values.
 */
private
SaveStatus()
{
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the
 <code>SaveStatus</code>.
 * @return A <code>String</code> with value "SaveStatus(n)"
 where n is <code>this</code> value
 */
public String
toString()
{
    return "SaveStatus(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and
 <code>otherSaveStatusValue</code> represent the same save status.
 * @param otherSaveStatusValue The <code>Object</code> to compare
 with
 * @return true iff supplied <code>other</code> is of type
 <code>SaveStatus</code> and has same value
 */
public boolean
equals(Object otherSaveStatusValue)
{
    if (otherSaveStatusValue instanceof SaveStatus)
        return _value == ((SaveStatus)otherSaveStatusValue)._value;
    else
        return false;
}

/**
 * Returns a hash code for <code>this</code>; two
 <code>SaveStatus</code>es for which <code>equals()</code> is
 <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int
hashCode()
{
    return _value;
}
// The constant instances
/**
 * No save in progress (federate in Active or Restore Request Pending states).
 */
static public final SaveStatus NO_SAVE_IN_PROGRESS = new SaveStatus();

/**
 * Federate in the Instructed To Save state.
 */
static public final SaveStatus FEDERATE_INSTRUCTED_TO_SAVE = new SaveStatus();

/**
 * Federate in the Saving state.
 */
static public final SaveStatus FEDERATE_SAVING = new SaveStatus();

/**
 * Federate in the Waiting For Federation To Save state.
 */
static public final SaveStatus FEDERATE_WAITING_FOR_FEDERATION_TO_SAVE = new SaveStatus();
} // end SaveStatus
// File: ServiceGroup.java
package hla.rti1516;  //the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java {@link java.util.Enumeration}![)
 * representing the seven HLA service groups (Federation management;
 * Declaration management;
 * Object management; Ownership management; Time management;
 * Data distribution management; and Support services).
 * "<p>
 * DoD Interpretations of IEEE 1516-2000v2 changes one of the constant
 * names from "SUPPORT SERVICES" to "SUPPORT_SERVICES".
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 * @see hla.rti1516.RTIambassador#normalizeServiceGroup
 * normalizeServiceGroup
 */
public final class ServiceGroup
    implements java.io.Serializable
{
    //each instance's value
    private int _value;
    //initial value for enumeration: fedn mgt is chapter 4
    private static final int lowestValue = 4;  //_lowestValue = 0 may
    be more compatible with the MOM
    //begins at lowest
    private static int nextToAssign = lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a <code>ServiceGroup</code> must
     * be initialized with one of the defined static values.
     * @param otherServiceGroupValue must be a defined static value or
     * another instance.
     */
    public ServiceGroup(ServiceGroup otherServiceGroupValue)
    {
        _value = otherServiceGroupValue._value;
    }

    /**
     * Package-only (default access) constructor. Unused.
     * @param value to assign to the instance; must be one of the
     * static ones
     */
    ServiceGroup(int value)
        throws RTIinternalError
    {
        _value = value;
        if ((_value < _lowestValue) || (_value >= _nextToAssign))
            throw new RTIinternalError("ServiceGroup: illegal value " +
                value);
    }
private ServiceGroup() {
    _value = _nextToAssign++;
}

/**
 * Returns a <code>String</code> representation of the
 * <code>ServiceGroup</code>.
 * @return A {@link java.lang.String} with value "ServiceGroup(n)" where n is <code>this</code> value
 */
public String toString() {
    return "ServiceGroup(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and <code>otherServiceGroupValue</code> represent the same service group.
 * @param otherServiceGroupValue The <code>Object</code> to compare with
 * @return true iff supplied <code>otherServiceGroupValue</code> is of type <code>ServiceGroup</code> and has same value
 */
public boolean equals(Object otherServiceGroupValue) {
    if (otherServiceGroupValue instanceof ServiceGroup)
        return _value == ((ServiceGroup)otherServiceGroupValue)._value;
    else
        return false;
}

/**
 * Returns a hash code for <code>this</code>; two <code>ServiceGroup</code>s for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode() {
    return _value;
}
The constant instances

/**
 * Methods and callbacks referring to the creation, dynamic control, modification and deletion of a federation execution.
 */
static public final ServiceGroup FEDERATION_MANAGEMENT = new ServiceGroup();

/**
 * Methods and callbacks dealing with the intent of federates to generate or consume information.
 */
static public final ServiceGroup DECLARATION_MANAGEMENT = new ServiceGroup();

/**
 * Methods and callbacks dealing with the registration, modification and deletion of object instances, as well as the sending and receipt of interactions.
 */
static public final ServiceGroup OBJECT_MANAGEMENT = new ServiceGroup();

/**
 * Methods and callbacks dealing with the ownership of instance attributes among joined federates.
 * The ability to transfer ownership of instance attributes among joined federates supports the cooperative modeling of object instances across a federation.
 */
static public final ServiceGroup OWNERSHIP_MANAGEMENT = new ServiceGroup();

/**
 * Methods and callbacks dealing with the ordering of the delivery of messages throughout the federation execution.
 * Use of these mechanisms permits messages sent by different joined federates to be delivered in a consistent order to any joined federate in the federation execution that is to receive those messages.
 */
static public final ServiceGroup TIME_MANAGEMENT = new ServiceGroup();

/**
 * Methods and callbacks dealing with the reduction in the transmission and reception of irrelevant data. Whereas DECLARATION_MANAGEMENT services provide information on data relevance at the class attribute and interaction class levels, DATA_DISTRIBUTION_MANAGEMENT services add the capability to further refine the data requirements at the instance attribute and specific interaction levels.
 */
static public final ServiceGroup DATA_DISTRIBUTION_MANAGEMENT = new ServiceGroup();
/**
 * Miscellaneous services utilized by joined federates for performing such actions as
 * name-to-handle and handle-to-name transformation, setting advisory switches and
 * manipulating regions.
 */
static public final ServiceGroup SUPPORT_SERVICES = new ServiceGroup();
//end ServiceGroup
// File: TimeQueryReturn.java
package hla.rti1516; // the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * Record returned by (8.16) `@link RTIambassador#queryGALT queryGALT`
 * and (8.18) `@link RTIambassador#queryLITS queryLITS`.
 * It consists of a guard boolean `<code>timeIsValid</code>` and a
 * payload `<code>LogicalTime</code> <code>time</code>.
 * <p>
 * DoD Interpretations of IEEE 1516-2000v2 adds the
 * java.io.Serializable implementation and a constructor.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */

public final class TimeQueryReturn
    implements java.io.Serializable
{
    /**
     * Whether the other field is valid or not.
     */
    public boolean timeIsValid;

    /**
     * Payload Logical Time.
     */
    public LogicalTime time;

    /**
     * Public constructor.
     * @param tiv Whether the <code>LogicalTime</code> field is valid or not
     * @param It The <code>LogicalTime</code> field
     */
    public TimeQueryReturn(boolean tiv, LogicalTime lt)
    {
        timeIsValid = tiv;
        time = lt;
    }

    /**
     * Returns a <code>String</code> representation of the
     * <code>TimeQueryReturn</code>.
     * @return A `{@link java.lang.String} with value
     * 
     */
    public String toString()
    {
        return "" + timeIsValid + " " + time;
    }
}
/**
 * Returns true iff <code>this</code> and <code>other</code> represent the same time query return.
 * @param other The <code>Object</code> to compare with
 * @return true iff supplied <code>other</code> is of type <code>TimeQueryReturn</code> and has same value
 */
public boolean equals(Object other) {
    if (other instanceof TimeQueryReturn) {
        TimeQueryReturn tqrOther = (TimeQueryReturn) other;
        if ((timeIsValid == false) && (tqrOther.timeIsValid == false)) {
            // When timeIsValid is false, the payloads are ignored
            return true;
        } else if ((timeIsValid == true) && (tqrOther.timeIsValid == true)) {
            // When timeIsValid is true, the payloads must match
            return time.equals(tqrOther.time);
        } else {
            // Mismatched timeIsValid fields
            return false;
        }
    } else {
        // Not the same classes
        return false;
    }
}

/**
 * Returns a hash code for <code>this</code>; two <code>TimeQueryReturn</code>s for which <code>equals()</code> is true should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int hashCode() {
    return (timeIsValid ? time.hashCode() : 7);
}

// End TimeQueryReturn
// File: TransportationType.java
package hla.rti1516; //the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * An enumerated type (not a Java {@link java.utilEnumeration})
 * representing the RTI-provided means of message transmission between
 * joined federates.
 * Transportation Type defaults are defined at the attribute and
 * parameter level by the FOM Document Data; these may be overridden by
 * their owners.
 * The two core TransportationTypes are:
 * * `<ul><li>`<code>HLAreliable</code>`</li>`: provides reliable delivery of data in
 * the sense that TCP/IP delivers its data reliably
 * * `<ul><li>`<code>HLAbestEffort</code>`</li>`: makes an effort to deliver data in
 * the sense that UDP provides best-effort delivery
 * </ul>`
 * Additional `<code>TransportationType</code>`s may be provided by
 * specific RTIs.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public class TransportationType
    implements java.io.Serializable
{
    /**
     * Each instance's value.
     */
    protected int _value;

    //initial value for enumeration
    private static final int _lowestValue = 1;

    /**
     * The enumeration begins at the lowest value.
     */
    protected static int _nextToAssign = _lowestValue;

    /**
     * This is the only public constructor.
     * Each user-defined instance of a `<code>TransportationType</code>`
     * must be initialized with one of the defined static values.
     * @param otherTransportationTypeValue must be a defined static
     * value or another instance.
     */
    public TransportationType(TransportationType
        otherTransportationTypeValue)
    {
        _value = otherTransportationTypeValue._value;
    }
/**
 * Class and subclass constructor; it is used to generate the
 * static values.
 * Because this class is RTI-extendable, the constructor is
 * protected instead of private.
 */
protected TransportationType()
{
    _value = _nextToAssign++;
}

/**
 * Package-only (default access) constructor. Used by the
 * @link TransportationType#decode decode method.
 * @param value an <code>int</code> to assign to the instance; must
 * be one of the static values
 */
TransportationType(int value)
    throws RTIinternalError
{
    _value = value;
    if ((value < _lowestValue) || (value >= _nextToAssign))
        throw new RTIinternalError("TransportationType: illegal value
" + value);
}

/**
 * Returns a <code>String</code> representation of the
 * <code>TransportationType</code>.
 * @return A <code>java.lang.String</code> with value
 * "TransportationType(n)" where n is <code>this</code> value
 */
public String
toString()
{
    return "TransportationType(" + _value + ")";
}

/**
 * Returns true iff <code>this</code> and
 * <code>otherTransportationTypeValue</code> represent the same
 * transportation type.
 * @param otherTransportationTypeValue The <code>Object</code> to
 * compare with
 * @return <code>true</code> iff supplied
 * <code>otherTransportationTypeValue</code> is of type
 * <code>TransportationType</code> and has same value
 */
public boolean
equals(Object otherTransportationTypeValue)
{
    if (otherTransportationTypeValue instanceof TransportationType)
        return _value == ((TransportationType)otherTransportationTypeValue)._value;
    else
        return false;
}
/**
 * Returns a hash code for <code>this</code>; two
 * <code>TransportationType</code>s for which <code>equals()</code> is
 * <code>true</code> should yield the same hash code.
 * @return An <code>int</code> hash code
 */
public int
hashCode()
{
    return _value;
}

/**
 * Returns the length (in bytes) of the <code>byte[]</code>
 * representation of the <code>TransportationType</code>.
 * @return The length (in bytes) of the <code>byte[]</code>
 * representation of <code>this</code>
 */
public int
encodedLength()
{
    return 1;
}

/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the
 * specified <code>offset</code>.  
 * @param buffer the <code>byte[]</code> into which to encode
 * <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which
 * to encode <code>this</code>
 */
public void
encode(byte[] buffer,
        int offset)
{
    buffer[offset] = (byte)_value;
}
/**
 * Creates a <code>TransportationType</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>TransportationType</code>
 * @param offset where in the <code>buffer</code> the <code>TransportationType</code> representation begins
 * @return The <code>TransportationType</code> that was encoded in the provided <code>buffer</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded into an instance of the factory's target class
 */
 public static TransportationType decode (byte[] buffer, 
                               int offset) 
                               throws CouldNotDecode 
 { 
     int val = buffer[offset]; 
     TransportationType neo; 
     try 
     { 
         neo = new TransportationType(val); 
     } 
     catch (RTIinternalError e) 
     { 
         throw new CouldNotDecode(e.getMessage()); 
     } 
     return neo; 
 }

//The two core static instances
/**
 * Provides reliable delivery of data in the sense that TCP/IP delivers its data reliably.
 * Speed is sacrificed for reliability.
 */
 static public final TransportationType HLA_RELIABLE = new TransportationType();

/**
 * Makes an effort to deliver data in the sense that UDP provides best-effort delivery.
 * Reliability is sacrificed for speed.
 */
 static public final TransportationType HLA_BEST_EFFORT = new TransportationType();

//end TransportationType
// File: RTIexception.java
package hla.rti1516; // the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * Superclass of all exceptions thrown by the RTI.
 * All RTI exceptions must be caught or specified.
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public class RTIexception
    extends Exception
{
    /**
     * Constructs a new exception with the specified detail message.
     * The cause is not initialized, and may subsequently be initialized
     * by a call to {@link java.lang Throwable#initCause(java.lang.Throwable)}.
     * @param msg a @link java.lang.String holding the detail message, which can be later retrieved by the @link java.lang Throwable#getMessage getMessage method
     */
    public RTIexception(String msg)
    {
        super(msg);
    }
}
//end RTIexception
The various RTIexception descendents follow a single format. Using the template listed below, the expressions <exception_name> and <exception_description> are to be replaced successively with the values given by the table that follows the template. The proposed exception InvalidLogicalTimeInterval is listed separately.

```
// File: <exception_name>.java
package hla.rti1516; // the package name was changed by DoD
Interpretations of IEEE 1516-2000v2

/**
 * <exception_description>
 * @author IEEE
 * @version 1516.1.5 (DoD v2)
 */
public final class <exception_name>
    extends RTIexception
{
    /**
     * Constructs a new exception with the specified detail message.
     * The cause is not initialized, and may subsequently be initialized
     * by a call to {@link java.lang.Throwable#initCause(java.lang.Throwable)}.
     * @param msg a {@link java.lang.String} holding the detail message, which can be later retrieved by the
     *            {@link java.lang.Throwable#getMessage} method
     */
    public <exception_name> (String msg)
    {
        super(msg);
    }
}
```

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsynchronousDeliveryAlreadyDisabled</td>
<td>Exception thrown by the (8.15) <code>RTIambassador#disableAsynchronousDelivery</code> method when asynchronous delivery is already disabled for the federate.</td>
</tr>
<tr>
<td>AsynchronousDeliveryAlreadyEnabled</td>
<td>Exception thrown by the (8.14) <code>RTIambassador#enableAsynchronousDelivery</code> method when asynchronous delivery is already enabled for the federate.</td>
</tr>
<tr>
<td>AttributeAcquisitionWasNotCanceled</td>
<td>Exception that should be thrown by the (7.15) <code>FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation</code> callback if the federate wishes to repudiate the attribute ownership acquisition cancellation.</td>
</tr>
<tr>
<td>AttributeAcquisitionWasNotRequested</td>
<td>Exception thrown by the (7.14) <code>RTIambassador#cancelAttributeOwnershipAcquisition</code> method when the attribute ownership acquisition request to cancel was not previously made. It should also be thrown by the (7.7) <code>FederateAmbassador#attributeOwnershipAcquisitionNotification</code> and (7.10) <code>FederateAmbassador#attributeOwnershipUnavailable</code> callbacks when the attribute ownership acquisition being granted or denied (respectively) is not recognised as having been previously requested.</td>
</tr>
<tr>
<td>AttributeAlreadyBeingAcquired</td>
<td>Exception thrown by the (7.9) <code>RTIambassador#attributeOwnershipAcquisitionIfAvailable</code> method when an unconditional attribute ownership acquisition request is pending for the specified attribute instances.</td>
</tr>
<tr>
<td>Exception Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AttributeAlreadyBeingDivested</td>
<td>Exception thrown by the (7.3) [link RTIambassador#negotiatedAttributeOwnershipDivestiture negotiatiedAttributeOwnershipDivestiture] method when an attribute ownership divestiture request is already pending for the specified attribute instances.</td>
</tr>
<tr>
<td>AttributeAlreadyOwned</td>
<td>Exception thrown by the (7.14) [link RTIambassador#cancelAttributeOwnershipAcquisition cancelAttributeOwnershipAcquisition] method when the federate's attribute ownership acquisition cancellation occurs too late (i.e. it has already been granted ownership). It should also be thrown by the (7.4) [link FederateAmbassador#requestAttributeOwnershipAssumption requestAttributeOwnershipAssumption], (7.7) [link FederateAmbassador#attributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotification], (7.10) [link FederateAmbassador#attributeOwnershipUnavailable attributeOwnershipUnavailable] and (7.15) [link FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation confirmAttributeOwnershipAcquisitionCancellation] callbacks when the attribute ownership acquisition being offered, granted, denied or cancelled (respectively) conflicts with the federate's perceived ownership.</td>
</tr>
<tr>
<td>AttributeDivestitureWasNotRequested</td>
<td>Exception thrown by the (7.14) [link RTIambassador#cancelAttributeOwnershipAcquisition cancelAttributeOwnershipAcquisition] method when the federate's attribute ownership acquisition cancellation occurs too late (i.e. it has already been granted ownership). It should also be thrown by the (7.4) [link FederateAmbassador#requestAttributeOwnershipAssumption requestAttributeOwnershipAssumption], (7.7) [link FederateAmbassador#attributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotification], (7.10) [link FederateAmbassador#attributeOwnershipUnavailable attributeOwnershipUnavailable] and (7.15) [link FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation confirmAttributeOwnershipAcquisitionCancellation] callbacks when the attribute ownership acquisition being offered, granted, denied or cancelled (respectively) conflicts with the federate's perceived ownership.</td>
</tr>
</tbody>
</table>
**AttributeNotDefined**

Exception thrown when the specified attribute could not be recognised within the supplied context. The following RTIambassador methods throw it: <ul>
- @link RTIambassador#publishObjectClassAttributes publishObjectClassAttributes <li>@link RTIambassador#unpublishObjectClassAttributes unpublishObjectClassAttributes</li>
- @link RTIambassador#subscribeObjectClassAttributes subscribeObjectClassAttributes <li>@link RTIambassador#subscribeObjectClassAttributesPassively subscribeObjectClassAttributesPassively.unsubscribe</li>
- @link RTIambassador#updateAttributeValues updateAttributeValues</ul>

380 DR DC Valcartier TR 2007-412
AttributeNotOwned

Exception thrown when the specified attribute instance(s) was(were) not owned by the federate. The following RTIambassador methods throw it: <ul>
<li>{@link RTIambassador#updateAttributeValues updateAttributeValues} (both forms)</li>
<li>{@link RTIambassador#changeAttributeTransportationType changeAttributeTransportationType} / {@link RTIambassador#changeAttributeOrderType changeAttributeOrderType}</li>
<li>{@link RTIambassador#unconditionalAttributeOwnershipDivestiture unconditional}</li>
<li>{@link RTIambassador#attributeOwnershipDivestitureIfWanted attributeOwnershipDivestitureIfWanted} / {@link RTIambassador#negotiatedAttributeOwnershipDivestiture negotiated}</li>
<li>{@link RTIambassador#cancelNegotiatedAttributeOwnershipDivestiture cancelNegotiatedAttributeOwnershipDivestiture}</li>
<li>{@link RTIambassador#confirmDivestiture confirmDivestiture}</li>
</ul>

It should also be thrown by the following FederateAmbassador callbacks when the federate repudiates ownership of the specified attribute instances: <ul>
<li>{@link FederateAmbassador#provideAttributeValueUpdate provideAttributeValueUpdate}</li>
<li>{@link FederateAmbassador#turnUpdatesOnForObjectInstance turnUpdatesOn}</li>
<li>{@link FederateAmbassador#turnUpdatesOffForObjectInstance turnUpdatesOff}</li>
<li>{@link FederateAmbassador#requestDivestitureConfirmation requestDivestitureConfirmation}</li>
<li>{@link FederateAmbassador#requestAttributeOwnershipRelease requestAttributeOwnershipRelease}</li>
</ul>

AttributeNotPublished

Exception thrown by the (7.8/7.9) {@link RTIambassador#attributeOwnershipAcquisition attributeOwnershipAcquisition} and (9.5) {@link RTIambassador#registerObjectInstanceWithRegions registerObjectInstanceWithRegions} methods when some of the attributes to acquire or register aren't published.

It should also be thrown by the (7.4) {@link FederateAmbassador#requestAttributeOwnershipAssumption requestAttributeOwnershipAssumption} and (7.7) {@link FederateAmbassador#attributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotification} callbacks when some of the offered or granted attributes aren't published in the federate's opinion.
AttributeNotRecognized

Exception that should be thrown when the specified attributes aren't recognized by the federate in the supplied object class context. The following FederateAmbassador callbacks should throw it:

- `FederateAmbassador#reflectAttributeValues`
- `FederateAmbassador#attributesInScope`
- `FederateAmbassador#attributesOutOfScope`
- `FederateAmbassador#provideAttributeValueUpdate`
- `FederateAmbassador#turnUpdatesOnForObjectInstance`
- `FederateAmbassador#turnUpdatesOffForObjectInstance`
- `FederateAmbassador#requestAttributeOwnershipAssumption`
- `FederateAmbassador#requestDivestitureConfirmation`
- `FederateAmbassador#attributeOwnershipAcquisitionNotification`
- `FederateAmbassador#attributeOwnershipUnavailable`
- `FederateAmbassador#requestAttributeOwnershipRelease`
- `FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation`
- `FederateAmbassador#informAttributeOwnership`
- `FederateAmbassador#attributeIsNotOwned`
- `FederateAmbassador#attributeIsOwnedByRTI`

AttributeNotSubscribed

Exception that should be thrown by the (6.7) `FederateAmbassador#reflectAttributeValues` (all six forms) and (6.15/6.16) `FederateAmbassador#attributesInScope` / `FederateAmbassador#attributesOutOfScope` callbacks when the specified attributes aren't published by the federate in its opinion.

AttributeRelevanceAdvisorySwitchIsOff

Exception thrown by the (10.25) `RTIambassador#disableAttributeRelevanceAdvisorySwitch` method when the attribute relevance advisory switch is already off.
<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeRelevanceAdvisorySwitchIsOn</td>
<td>Exception thrown by the (10.24) [@link RTIambassador#enableAttributeRelevanceAdvisorySwitch enableAttributeRelevanceAdvisorySwitch] method when the attribute relevance advisory switch is already on.</td>
</tr>
<tr>
<td>AttributeScopeAdvisorySwitchIsOff</td>
<td>Exception thrown by the (10.27) [@link RTIambassador#disableAttributeScopeAdvisorySwitch disableAttributeScopeAdvisorySwitch] method when the attribute scope advisory switch is already off.</td>
</tr>
<tr>
<td>AttributeScopeAdvisorySwitchIsOn</td>
<td>Exception thrown by the (10.26) [@link RTIambassador#enableAttributeScopeAdvisorySwitch enableAttributeScopeAdvisorySwitch] method when the attribute scope advisory switch is already on.</td>
</tr>
<tr>
<td>CouldNotDecode</td>
<td>Exception thrown by the <code>&lt;code&gt;decode&lt;/code&gt;</code> method of a factory class when it fails to decode the specified byte buffer into an instance of the factory's target class. The following factory classes throw it: &lt;ul&gt; &lt;li&gt;{@link AttributeHandleFactory} &lt;/li&gt; &lt;li&gt;{@link DimensionHandleFactory} &lt;/li&gt; &lt;li&gt;{@link FederateHandleFactory} &lt;/li&gt; &lt;li&gt;{@link InteractionClassHandleFactory} &lt;/li&gt; &lt;li&gt;{@link LogicalTimeFactory} &lt;/li&gt; &lt;li&gt;{@link LogicalTimeIntervalFactory} &lt;/li&gt; &lt;li&gt;{@link ObjectClassHandleFactory} &lt;/li&gt; &lt;li&gt;{@link ObjectInstanceHandleFactory} &lt;/li&gt; &lt;li&gt;{@link ParameterHandleFactory} &lt;/li&gt; &lt;li&gt;{@link OrderType} &lt;/li&gt; &lt;li&gt;{@link TransportationType} &lt;/li&gt; &lt;/ul&gt;</td>
</tr>
<tr>
<td>CouldNotDiscover</td>
<td>Exception that should be thrown by the (6.5) [@link FederateAmbassador#discoverObjectInstance discoverObjectInstance] callback when it fails for some reason other than an unrecognized object class.</td>
</tr>
<tr>
<td>CouldNotInitiateRestore</td>
<td>Exception that should be thrown by the (4.21) [@link FederateAmbassador#initiateFederateRestore initiateFederateRestore] callback when it fails for some reason other than an unrecognized save label.</td>
</tr>
<tr>
<td>Exception Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CouldNotOpenFDD</td>
<td>Exception thrown by the RTIambassador#createFederationExecution method when it fails to reach or open the specified FOM Document Data (FDD) file.</td>
</tr>
<tr>
<td>Delete Privilege Not Held</td>
<td>Exception thrown by the RTIambassador#createFederationExecution method when it fails to reach or open the specified FOM Document Data (FDD) file.</td>
</tr>
<tr>
<td>ErrorReadingFDD</td>
<td>Exception thrown by the RTIambassador#createFederationExecution method when the contents of the FOM Document Data (FDD) file prove unsuitable.</td>
</tr>
<tr>
<td>Federate Already Execution Member</td>
<td>Exception thrown by the RTIambassador#joinFederationExecution method when the federate (as represented by the RTIambassador instance) is already joined to any federation execution (not just the specified one).</td>
</tr>
<tr>
<td>Federate Has Not Begun Save</td>
<td>Exception thrown by the RTIambassador.federateSave methods if the federate is not in the Saving state.</td>
</tr>
<tr>
<td>Federate Internal Error</td>
<td>Exception that should be thrown by all of the RTIambassador's callbacks when something goes wrong unless a more specific exception is appropriate.</td>
</tr>
<tr>
<td>Federate Not Execution Member</td>
<td>Exception thrown by nearly all of the RTIambassador's methods if the federate isn't joined.</td>
</tr>
<tr>
<td>Exception Name</td>
<td>Exception Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FederateOwnsAttributes</td>
<td>Exception thrown by the (4.5) <code>{@link RTIambassador#resignFederationExecution resignFederationExecution}</code>, (6.12) <code>{@link RTIambassador#localDeleteObjectInstance localDeleteObjectInstance}</code> and (7.8/7.9) <code>{@link RTIambassador#attributeOwnershipAcquisition attributeOwnershipAcquisition}</code> methods if the federate, in the first case, owns some instance attributes and hasn't specified a disposal policy; or, in the latter cases, owns some of the attributes it is trying to locally delete or acquire.</td>
</tr>
<tr>
<td>FederatesCurrentlyJoined</td>
<td>Exception thrown by the (4.3) <code>{@link RTIambassador#destroyFederationExecution destroyFederationExecution}</code> method if the federation execution isn't empty.</td>
</tr>
<tr>
<td>FederateServiceInvocationsAreBeingReportedViaMOM</td>
<td>Exception thrown by the (5.8/9.10) <code>{@link RTIambassador#subscribeInteractionClass subscribeInteractionClass}</code>, <code>{@link RTIambassador#subscribeInteractionClassPassively subscribeInteractionClassPassively}</code>, <code>{@link RTIambassador#subscribeInteractionClassWithRegions subscribeInteractionClassWithRegions}</code> methods if service invocations are currently being reported via MOM interactions and the specified interaction class is <code>HLAmanager.HLAfederate.HLAreport.HLAreportServiceInvocation</code>. Such a subscription would cause an infinite loop (the first invocation would trigger a report of itself and so on).</td>
</tr>
<tr>
<td>FederateUnableToUseTime</td>
<td>Exception thrown by the (4.11) <code>{@link RTIambassador#requestFederationSave(String,LogicalTime) requestFederationSave(String,LogicalTime)}</code> (time-stamped form) method if the specified time-stamp, although not in the federate's past, is nevertheless too soon to be achievable.</td>
</tr>
<tr>
<td>FederationExecutionAlreadyExists</td>
<td>Exception thrown by the (4.2) <code>{@link RTIambassador#createFederationExecution createFederationExecution}</code> method if the specified federation execution already exists.</td>
</tr>
</tbody>
</table>
FederationExecutionDoesNotExist

Exception thrown by the (4.3) {@link RTIambassador#destroyFederationExecution destroyFederationExecution} and (4.4) {@link RTIambassador#joinFederationExecution joinFederationExecution} methods if the specified federation execution does not exist.

IllegalName

Exception thrown by the (6.2) {@link RTIambassador#reserveObjectInstanceName reserveObjectInstanceName} method if the specified object instance name is ill-formed.

Object instance names may not be the empty String (""), nor begin with "HLA" (including all case variations such as "hla" or "Hla"), nor equal "na"; there are no other restrictions.

Names are constructed from a combination of letters (a..z and A..Z), digits (0..9), hyphens and underscores. The period is used to qualify a class; it is the "path" character linking a super-class with its sub-class. These rules apply to object and interaction classes, attributes and parameters, datatypes (including enumerators and values), record fields, dimensions, transportation types, synchronization point labels and OMT note labels. It is not made clear anywhere whether these rules apply to object instance names, although this is probably a safe assumption.

IllegalTimeArithmetic

Exception thrown by the LogicalTime {@link LogicalTime#add add} and {@link LogicalTime#subtract subtract} methods if adding (subtracting) the specified LogicalTimeInterval would result in a LogicalTime lying after (before) the final (initial) LogicalTime.
### InteractionClassNotDefined

Exception thrown by the (5.4/5.5) `{@link RTIambassador#publishInteractionClass publish}` / `{@link RTIambassador#unpublishInteractionClass unpublish}` InteractionClass, (5.8/9.10) `{@link RTIambassador#subscribeInteractionClass subscribeInteractionClass}` @link RTIambassador#subscribeInteractionClassPassively [Passively], (5.9/9.11) `{@link RTIambassador#unsubscribeInteractionClass unsubscribeInteractionClass}` @link RTIambassador#unsubscribeInteractionClassWithRegions [WithRegions], (6.8/9.12) `{@link RTIambassador#sendInteraction sendInteraction}` / `{@link RTIambassador#sendInteractionWithRegions [WithRegions] sendInteractionWithRegions [WithRegions]} (all forms) and (6.14/8.24) `{@link RTIambassador#changeInteractionTransportationType Transportation} / `{@link RTIambassador#changeInteractionOrderType order} Type methods when the specified interaction class isn’t recognized.

### InteractionClassNotPublished

Exception thrown by the (6.8/9.12) `{@link RTIambassador#sendInteraction sendInteraction}` / `{@link RTIambassador#sendInteractionWithRegions [WithRegions] sendInteractionWithRegions [WithRegions]}` and (6.14/8.24) `{@link RTIambassador#changeInteractionTransportationType Transportation} / `{@link RTIambassador#changeInteractionOrderType order} Type methods when the specified interaction class isn’t currently published by the federate.

*It should also be thrown by the (5.12/5.13) FederateAmbassador.turnInteractions `on` / `off` callbacks if the federate denies publishing the specified interaction class.*

### InteractionClassNotRecognized

Exception that should be thrown by the (6.9) `{@link FederateAmbassador#receiveInteraction receiveInteraction}` (all forms) callback if the federate does not recognize the supplied interaction class.

### InteractionClassNotSubscribed

Exception that should be thrown by the (6.9) `{@link FederateAmbassador#receiveInteraction receiveInteraction}` (all forms) callback if the federate denies subscribing to the supplied interaction class.
<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InteractionParameterNotDefined</td>
<td>Exception thrown by the (6.8/9.12) <code>{@link RTIambassador#sendInteraction sendInteraction}</code> <code>{@link RTIambassador#sendInteractionWithRegions sendInteractionWithRegions}</code> (both forms) and (10.9) <code>{@link RTIambassador#getParameterName getParameterName}</code> methods when (at least one of) the specified parameter(s) isn’t defined in the context of the specified interaction class.</td>
</tr>
<tr>
<td>InteractionParameterNotRecognized</td>
<td>Exception that should be thrown by the (6.9) <code>{@link FederateAmbassador#receiveInteraction receiveInteraction}</code> (all forms) callback if at least one of the supplied parameters isn’t recognized in the context of the specified interaction class.</td>
</tr>
<tr>
<td>InteractionRelevanceAdvisorySwitchIsOff</td>
<td>Exception thrown by the (10.29) <code>{@link RTIambassador#disableInteractionRelevanceAdvisorySwitch disableInteractionRelevanceAdvisorySwitch}</code> method when the interaction relevance advisory switch is already off.</td>
</tr>
<tr>
<td>InteractionRelevanceAdvisorySwitchIsOn</td>
<td>Exception thrown by the (10.28) <code>{@link RTIambassador#enableInteractionRelevanceAdvisorySwitch enableInteractionRelevanceAdvisorySwitch}</code> method when the interaction relevance advisory switch is already on.</td>
</tr>
<tr>
<td>InTimeAdvancingState</td>
<td>Exception thrown by the (8.2) <code>{@link RTIambassador#enableTimeRegulation enableTimeRegulation}</code>, (8.5) <code>{@link RTIambassador#enableTimeConstrained enableTimeConstrained}</code>, (8.8/8.9) <code>{@link RTIambassador#timeAdvanceRequest timeAdvanceRequest}</code> <code>{@link RTIambassador#timeAdvanceRequestAvailable timeAdvanceRequestAvailable}</code> <code>{@link RTIambassador#nextMessageRequest nextMessageRequest}</code> <code>{@link RTIambassador#nextMessageRequestAvailable nextMessageRequestAvailable}</code> <code>{@link RTIambassador#flushQueueRequest flushQueueRequest}</code> <code>{@link RTIambassador#flushQueueRequestAvailable flushQueueRequestAvailable}</code> and (8.10) <code>{@link RTIambassador#modifyLookahead modifyLookahead}</code> methods when the federate is in the Time Advancing state.</td>
</tr>
<tr>
<td>InvalidAttributeHandle</td>
<td>Exception thrown by the (10.5) <code>{@link RTIambassador#getAttributeName getAttributeName}</code> and (10.15) <code>{@link RTIambassador#getAvailableDimensionsForClassAttribute getAvailableDimensionsForClassAttribute}</code> methods when the specified attribute handle doesn’t exist (in any object class context).</td>
</tr>
</tbody>
</table>
InvalidDimensionHandle

Exception thrown by the (9.2) {@link RTIambassador#createRegion createRegion}, (10.13) {@link RTIambassador#getDimensionName getDimensionName} and (10.14) {@link RTIambassador#getDimensionUpperBound getDimensionUpperBound} methods when (one of) the specified dimension handle(s) doesn't exist.

InvalidFederateHandle

Exception thrown by the (10.33) {@link RTIambassador#normalizeFederateHandle normalizeFederateHandle} method when the specified federate handle is invalid.

Note that when the (4.6) {@link RTIambassador#registerFederationSynchronizationPoint(String,byte[],FederateHandleSet) method is supplied a {@link FederateHandleSet} that includes one or more invalid federate handles, it does not throw this exception. Instead the RTI invokes the {@link FederateAmbassador#synchronizationPointRegistrationFailed synchronizationPointRegistrationFailed} callback with the reason <code>SYNCHRONIZATION_SET_MEMBER_NOT_JOINED</code>.

InvalidInteractionClassHandle

Exception thrown by the (10.7) {@link RTIambassador#getInteractionClassName getInteractionClassName}, (10.8) {@link RTIambassador#getParameterHandle getParameterHandle}, (10.9) {@link RTIambassador#getParameterName getParameterName} and (10.17) {@link RTIambassador#getAvailableDimensionsForInteractionClass getAvailableDimensionsForInteractionClass} methods when the specified interaction class handle is invalid.
InvalidLogicalTime

Exception thrown by the (4.11) { @link RTIambassador#requestFederationSave(String,LogicalTime) }, (6.6)
{ @link RTIambassador#updateAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],LogicalTime) }, (6.8)
{ @link RTIambassador#sendInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],LogicalTime) }, (9.12)
{ @link RTIambassador#sendInteractionWithRegions(InteractionClassHandle,ParameterHandleValueMap,RegionHandleSet,byte[],LogicalTime) }, (6.10)
{ @link RTIambassador#deleteObjectInstance(ObjectInstanceHandle,byte[],LogicalTime) }, (8.8/8.9)
{ @link RTIambassador#timeAdvanceRequest timeAdvanceRequest }, (8.10/8.11)
{ @link RTIambassador#nextMessageRequest nextMessageRequest }, (8.12)
{ @link RTIambassador#flushQueueRequest flushQueueRequest } methods when the specified time-stamp is invalid.

It should also be thrown by the (4.12) { @link FederateAmbassador#initiateFederateSave(String,LogicalTime) }, (6.7)
{ @link FederateAmbassador#reflectAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle) }, (6.9)
{ @link FederateAmbassador#reflectAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet) }, (6.11)
{ @link FederateAmbassador#receiveInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[],OrderType,TransportationType,LogicalTime,OrderType,MessageRetractionHandle,RegionHandleSet) }, (6.12)
{ @link FederateAmbassador#removeObjectInstance(ObjectInstanceHandle,byte[],OrderType,LogicalTime,OrderType,MessageRetractionHandle) }, (8.3)
{ @link FederateAmbassador#timeRegulationEnabled timeRegulationEnabled } and (8.13)
{ @link FederateAmbassador#timeConstrainedEnabled timeConstrainedEnabled } and
{ @link FederateAmbassador#timeAdvanceGrant timeAdvanceGrant } callbacks if the federate considers the time-stamp invalid.

InvalidLookahead

Exception thrown by the (8.2) { @link RTIambassador#enableTimeRegulation enableTimeRegulation } and (8.19)
{ @link RTIambassador#modifyLookahead modifyLookahead } methods when the supplied lookahead { @link LogicalTimeInterval } is invalid.
### InvalidMessageRetractionHandle

Exception thrown by the (8.21) [@link RTIambassador#retract retract] method when the supplied [@link MessageRetractionHandle] is invalid.

### InvalidObjectClassHandle

Exception thrown by the (10.3) [@link RTIambassador#getObjectClassName getObjectClassName], (10.4) [@link RTIambassador#getAttributeHandle getAttributeHandle], (10.5) [@link RTIambassador#getAttributeName getAttributeName] and (10.15) [@link RTIambassador#getAvailableDimensionsForClassAttribute getAvailableDimensionsForClassAttribute] methods when the supplied [@link ObjectClassHandle] is invalid.

### InvalidOrderName

Exception thrown by the (10.20) [@link RTIambassador#getOrderType getOrderType] method when the supplied order type name is invalid.

As specified in [@link OrderType], the two possible order type names are <code>RECEIVE</code> and <code>TIMESTAMP</code>.

### InvalidOrderType

Exception thrown by the (10.21) [@link RTIambassador#getOrderName getOrderName] method when the supplied [@link OrderType] is invalid.

### InvalidParameterHandle

Exception thrown by the (10.9) [@link RTIambassador#getParameterName getParameterName] method when the supplied [@link ParameterHandle] is invalid.

### InvalidRangeBound

Exception thrown by the (10.32) [@link RTIambassador#setRangeBounds setRangeBounds] method when the supplied [@link RangeBounds] is invalid.
InvalidRegion

Exception thrown by the (9.3) \{@link RTIambassador#commitRegionModifications commitRegionModifications\}, (9.4) \{@link RTIambassador#deleteRegion deleteRegion\}, (9.5) \{@link RTIambassador#registerObjectInstanceWithRegions registerObjectInstanceWithRegions\} (both forms), (9.6) \{@link RTIambassador#associateRegionsForUpdates associateRegionsForUpdates\}, (9.7) \{@link RTIambassador#unassociateRegionsForUpdates unassociateRegionsForUpdates\}, (9.8) \{@link RTIambassador#subscribeObjectClassAttributesWithRegions subscribeObjectClassAttributesWithRegions\} \{@link RTIambassador#subscribeObjectClassAttributesPassivelyWithRegions subscribeObjectClassAttributesPassivelyWithRegions\} [Passively\}], (9.9) \{@link RTIambassador#unsubscribeObjectClassAttributesWithRegions unsubscribeObjectClassAttributesWithRegions\}, (9.10) \{@link RTIambassador#subscribeInteractionClassWithRegions subscribeInteractionClassWithRegions\} \{@link RTIambassador#subscribeInteractionClassPassivelyWithRegions subscribeInteractionClassPassivelyWithRegions\} [Passively\}], (9.11) \{@link RTIambassador#unsubscribeInteractionClassWithRegions unsubscribeInteractionClassWithRegions\}, (9.12) \{@link RTIambassador#sendInteractionWithRegions sendInteractionWithRegions\} (both forms), (9.13) \{@link RTIambassador#requestAttributeValueUpdateWithRegions requestAttributeValueUpdateWithRegions\}, (10.30) \{@link RTIambassador#getDimensionHandleSet getDimensionHandleSet\}, (10.31) \{@link RTIambassador#getRangeBounds getRangeBounds\} and (10.32) \{@link RTIambassador#setRangeBounds setRangeBounds\} methods when the supplied \{@link RegionHandle RegionHandle\} (which for most methods is included in a \{@link AttributeSetRegionSetPairList\}) is invalid.

In particular, \{@link RTIambassador#commitRegionModifications commitRegionModifications\} throws this exception if any of a region template's specified dimensions have not had their RangeBounds set beforehand.
### InvalidRegionContext

Exception thrown by the (9.5) `{@link RTIambassador#registerObjectInstanceWithRegions registerObjectInstanceWithRegions}` (both forms), (9.6) `{@link RTIambassador#associateRegionsForUpdates associateRegionsForUpdates}`, (9.8) `{@link RTIambassador#subscribeObjectClassAttributesWithRegions subscribeObjectClassAttributesWithRegions}`, (9.10) `{@link RTIambassador#subscribeObjectClassAttributesPassivelyWithRegions subscribeObjectClassAttributesPassivelyWithRegions [Passively]}], (9.12) `{@link RTIambassador#subscribeInteractionClassWithRegions subscribeInteractionClassWithRegions}`, (9.14) `{@link RTIambassador#subscribeInteractionClassPassivelyWithRegions subscribeInteractionClassPassivelyWithRegions [Passively]}}, (9.16) `{@link RTIambassador#sendInteractionWithRegions sendInteractionWithRegions} (both forms) and (9.13) `{@link RTIambassador#requestAttributeValueUpdateWithRegions requestAttributeValueUpdateWithRegions} methods when the specified dimensions of one of the `<code>Region</code>`s (supplied by a `{@link RegionHandleSet} or `{@link AttributeSetRegionSetPairList} argument) are not a subset of the available dimensions of the specified class attributes (as described by the FOM Document Data).

### InvalidServiceGroup

Exception thrown by the (10.34) `{@link RTIambassador#normalizeServiceGroup normalizeServiceGroup}` method when the specified `{@link ServiceGroup} is invalid.

### InvalidTransportationName

Exception thrown by the (10.18) `{@link RTIambassador#getTransportationType getTransportationType}` method when the specified transportation type name is invalid.

As specified in `{@link TransportationType}, the two core transportation type names are `<code>HLAreliable</code>` and `<code>HLABestEffort</code>`; additional `<code>TransportationType</code>`s may be provided by specific RTIs.

### InvalidTransportationType

Exception thrown by the (10.19) `{@link RTIambassador#getTransportationName getTransportationName}` method when the specified `{@link TransportationType} is invalid.

### JoinedFederateIsNotInTimeAdvancingState

Exception that should be thrown by the (8.13) `{@link FederateAmbassador#timeAdvanceGrant timeAdvanceGrant} callback if the federate does not consider itself in the Time Advancing state.
LogicalTimeAlreadyPassed

Exception thrown by the (4.11) \{@link RTIambassador#requestFederationSave requestFederationSave\}, (8.8/8.9) \{@link RTIambassador#timeAdvanceRequest timeAdvanceRequest\} \{@link RTIambassador#timeAdvanceRequestAvailable timeAdvanceRequestAvailable \{Available\}\}, (8.10/8.11) \{@link RTIambassador#nextMessageRequest nextMessageRequest\} \{@link RTIambassador#nextMessageRequestAvailable timeAdvanceRequestAvailable \{Available\}\} and (8.12) \{@link RTIambassador#flushQueueRequest flushQueueRequest\} methods when the specified \{@link LogicalTime\} is in the federation's past.

MessageCanNo LongerBeRetracted

Exception thrown by the (8.21) \{@link RTIambassador#retract retract\} method when the message associated with the specified \{@link MessageRetractionHandle\} can no longer be retracted. A federate in the Time Granted state can only retract messages with time stamps larger than the federate's current logical time plus its actual lookahead. A federate in the Time Advancing state can only retract messages with time stamps larger than the logical time specified in the federate's most recent (8.8) \{@link RTIambassador#timeAdvanceRequest timeAdvanceRequest\} plus its actual lookahead.

NameNotFound

Exception thrown by the (10.2) \{@link RTIambassador#getObjectClassHandle getObjectClassHandle\}, (10.4) \{@link RTIambassador#getAttributeHandle getAttributeHandle\}, (10.6) \{@link RTIambassador#getInteractionClassHandle getInteractionClassHandle\}, (10.8) \{@link RTIambassador#getParameterHandle getParameterHandle\} and (10.12) \{@link RTIambassador#getDimensionHandle getDimensionHandle\} methods when the specified name isn't recognized.

NoAcquisitionPending

Exception thrown by the (7.6) \{@link RTIambassador#confirmDivestiture confirmDivestiture\} method when, although there is a negotiated divestiture request pending for the specified attributes, the (7.5) \{@link FederateAmbassador#requestDivestitureConfirmation requestDivestitureConfirmation\} callback has not yet occurred. The exception was missing from Annex B but not from the 7.6 clause.

Pitch pRTI 1516 version 2.3 states that "This is deliberately not a final class" without explaining why.
<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Exception Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoRequestToEnableTimeConstrainedWasPending</td>
<td>Exception that should be thrown by the (8.6) <code>{@link FederateAmbassador#timeConstrainedEnabled timeConstrainedEnabled}</code> callback if the federate wants to repudiate its request to become time-constrained.</td>
</tr>
<tr>
<td>NoRequestToEnableTimeRegulationWasPending</td>
<td>Exception that should be thrown by the (8.3) <code>{@link FederateAmbassador#timeRegulationEnabled timeRegulationEnabled}</code> callback if the federate wants to repudiate its request to become time-regulating.</td>
</tr>
<tr>
<td>ObjectClassNotDefined</td>
<td>Exception thrown by the <code>{@link RTIambassador#publishObjectClassAttributes publishObjectClassAttributes}</code>, <code>{@link RTIambassador#unpublishObjectClass unpublishObjectClass}</code>, <code>{@link RTIambassador#unpublishObjectClassAttributes unpublishObjectClassAttributes}</code>, <code>{@link RTIambassador#subscribeObjectClassAttributes subscribeObjectClassAttributes}</code> <code>{@link RTIambassador#subscribeObjectClassAttributesPassively subscribeObjectClassAttributesPassively [Passively]}</code>, <code>{@link RTIambassador#unsubscribeObjectClassunsubscribeObjectClass}</code>, <code>{@link RTIambassador#unsubscribeObjectClassAttributes unsubscribeObjectClassAttributes}</code> <code>{@link RTIambassador#unsubscribeObjectClassAttributesWithRegions unsubscribeObjectClassAttributesWithRegions [WithRegions]}</code>, <code>{@link RTIambassador#registerObjectInstance registerObjectInstance}</code> <code>{@link RTIambassador#registerObjectInstanceWithRegions registerObjectInstanceWithRegions [WithRegions]} (all forms), </code>{@link RTIambassador#requestAttributeValueUpdate(ObjectClassHandle,AttributeHandleSet,byte[]) requestAttributeValueUpdate(ObjectClassHandle,AttributeHandleSet,byte[])<code>and</code>{@link RTIambassador#requestAttributeValueUpdateWithRegions requestAttributeValueUpdateWithRegions} methods when the supplied `{@link ObjectClassHandle} isn't recognized.</td>
</tr>
</tbody>
</table>
ObjectClassNotPublished

Exception thrown by the (6.4/9.5) `{@link RTIambassador#registerObjectInstance registerObjectInstance} (@link RTIambassador#registerObjectInstanceWithRegions [WithRegions]) (all forms) and (7.8/7.9) `{@link RTIambassador#attributeOwnershipAcquisition attributeOwnershipAcquisition} (@link RTIambassador#attributeOwnershipAcquisitionIfAvailable [IfAvailable])` methods when the supplied object class isn’t currently published by the federate.

It should also be thrown by the (5.10) `{@link FederateAmbassador#startRegistrationForObjectClass start} / (5.11) `{@link FederateAmbassador#stopRegistrationForObjectClass stop}` RegistrationForObjectClass callbacks if the federate does not publish the specified object class.

ObjectClassNotRecognized

Exception that should be thrown by the (6.5) `{@link FederateAmbassador#discoverObjectInstance discoverObjectInstance} callback if the federate does not recognize the specified object class.

ObjectClassRelevanceAdvisorySwitchIsOff

Exception thrown by the (10.23) `{@link RTIambassador#disableObjectClassRelevanceAdvisorySwitch disableObjectClassRelevanceAdvisorySwitch} method when the object class relevance advisory switch is already off.

ObjectClassRelevanceAdvisorySwitchIsOn

Exception thrown by the (10.22) `{@link RTIambassador#enableObjectClassRelevanceAdvisorySwitch enableObjectClassRelevanceAdvisorySwitch} method when the object class relevance advisory switch is already on.

ObjectNameInUse

Exception thrown by the (6.4) `{@link RTIambassador#registerObjectInstance(ObjectClassHandle, String)} and (9.5) `{@link RTIambassador#registerObjectInstanceWithRegions(ObjectClassHandle, AttributeSetRegionSetPairList, String)}` methods when the specified object instance name has already been used. An object instance name may not be re-used during the lifetime of the federation execution even if the object instance is deleted.
ObjectInstanceNameNotReserved

Exception thrown by the (6.4) {@link RTIambassador#registerObjectInstance(ObjectClassHandle,String)} and (9.5) {@link RTIambassador#registerObjectInstanceWithRegions(ObjectClassHandle,AttributeSetRegionSetPairList,String)} methods when the specified object instance name could not be reserved for some reason other than its having already been used.

Object instance names may not begin with "HLA" (including any case variations such as "Hla"). For details on name construction, see the [@link IllegalName] exception.

An object instance name may not be re-used during the lifetime of the federation execution even if the object instance is deleted.
### Exception Not Known

Exception thrown by the [@link RTIambassador#updateAttributeValues updateAttributeValues] (both forms), [@link RTIambassador#localDeleteObjectInstance localDeleteObjectInstance] {local}, [@link RTIambassador#deleteObjectInstance deleteObjectInstance] (all forms), changeAttribute {@link RTIambassador#changeAttributeTransportationType changeAttributeTransportationType} Transportation] / [@link RTIambassador#changeAttributeOrderType changeAttributeOrderType] Order] Type, [@link RTIambassador#requestAttributeValueUpdate requestAttributeValueUpdate] (ObjectInstanceHandle, AttributeHandleSet, byte[]), [@link RTIambassador#unconditionalAttributeOwnershipDivestiture unconditionalAttributeOwnershipDivestiture], [@link RTIambassador#negotiatedAttributeOwnershipDivestiture negotiatedAttributeOwnershipDivestiture], [@link RTIambassador#confirmDivestiture confirmDivestiture], [@link RTIambassador#attributeOwnershipAcquisition attributeOwnershipAcquisition], [@link RTIambassador#attributeOwnershipAcquisitionIfAvailable attributeOwnershipAcquisitionIfAvailable], [@link RTIambassador#attributeOwnershipDivestitureIfWanted attributeOwnershipDivestitureIfWanted], [@link RTIambassador#cancelNegotiatedAttributeOwnershipDivestiture cancelNegotiatedAttributeOwnershipDivestiture], [@link RTIambassador#cancelAttributeOwnershipAcquisition cancelAttributeOwnershipAcquisition], [@link RTIambassador#queryAttributeOwnership queryAttributeOwnership], [@link RTIambassador#isAttributeOwnedByFederate isAttributeOwnedByFederate], [@link RTIambassador#associateRegionsForUpdates associateRegionsForUpdates], [@link RTIambassador#unassociateRegionsForUpdates unassociateRegionsForUpdates], [@link RTIambassador#getObjectInstance getObjectInstance], [@link RTIambassador#getObjectInstanceHandle getObjectInstanceHandle] / [@link RTIambassador#getObjectInstanceName getObjectInstanceName] Name and [@link RTIambassador#getNameObjectClassHandle getNameObjectClassHandle] methods when the RTI considers that the specified [@link RTIambassador#getObjectInstanceHandle getObjectInstanceHandle] hasn’t been discovered by the federate.

(continued next page)
It should also be thrown by the `{@link FederateAmbassador#reflectAttributeValues reflectAttributeValues}` (all forms), `{@link FederateAmbassador#removeObjectInstance removeObjectInstance} (both forms), attributes `{@link FederateAmbassador#attributesInScope InScope} / `{@link FederateAmbassador#attributesOutOfScope OutOf} Scope, `{@link FederateAmbassador#provideAttributeValueUpdate provideAttributeValueUpdate}, turnUpdates `{@link FederateAmbassador#turnUpdatesOnForObjectInstance turnUpdatesOnForObjectInstance} / `{@link FederateAmbassador#turnUpdatesOffForObjectInstance turnUpdatesOffForObjectInstance} ForObjectInstance, requestAttributeOwnership `{@link FederateAmbassador#requestAttributeOwnership requestAttributeOwnership} / `{@link FederateAmbassador#requestAttributeOwnershipRelease requestAttributeOwnershipRelease}, requestDivestitureConfirmation `{@link FederateAmbassador#requestDivestitureConfirmation requestDivestitureConfirmation}, attributeOwnershipAcquisitionNotification `{@link FederateAmbassador#attributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotification}, attributeOwnershipUnavailable `{@link FederateAmbassador#attributeOwnershipUnavailable attributeOwnershipUnavailable}, confirmAttributeOwnershipAcquisitionCancellation `{@link FederateAmbassador#confirmAttributeOwnershipAcquisitionCancellation confirmAttributeOwnershipAcquisitionCancellation}, informAttributeOwnership `{@link FederateAmbassador#informAttributeOwnership informAttributeOwnership}, attributeIsNotOwned `{@link FederateAmbassador#attributeIsNotOwned attributeIsNotOwned} and `{@link FederateAmbassador#attributeIsOwnedByRTI attributeIsOwnedByRTI} callbacks if the federate denies having previously discovered the object instance.

**OwnershipAcquisitionPending**

Exception thrown by the (4.5) `{@link RTIambassador#resignFederationExecution resignFederationExecution}`, (5.3) `{@link RTIambassador#unpublishObjectClass unpublishObjectClass}`, `{@link RTIambassador#unpublishObjectClassAttributes unpublishObjectClassAttributes} and (6.12) `{@link RTIambassador#localDeleteObjectInstance localDeleteObjectInstance}` methods when there is at least one ownership acquisition pending for the specified object class (or any object class in the resignFederationExecution case).

**RegionDoesNotContainSpecifiedDimension**

Exception thrown by the (10.31) `{@link RTIambassador#getRangeBounds getRangeBounds}` and (10.32) `{@link RTIambassador#setRangeBounds setRangeBounds}` methods when the specified `{@link DimensionHandle}` is not bound to the specified `{@link RegionHandle}`.
<table>
<thead>
<tr>
<th><strong>exception_name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>RegionInUseForUpdateOrSubscription</td>
</tr>
<tr>
<td>Exception thrown by the [9.4] {@link RTI ambassador#deleteRegion deleteRegion} method when the specified &lt;code&gt;Region&lt;/code&gt; is still in use by the federation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>exception_description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>RegionNotCreatedByThisFederate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>exception_name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestForTimeConstrainedPending</td>
</tr>
<tr>
<td>Exception thrown by the [8.5] {@link RTI ambassador#enableTimeConstrained enableTimeConstrained}, [8.8/8.9] {@link RTI ambassador#timeAdvanceRequest timeAdvanceRequest} {@link RTI ambassador#timeAdvanceRequestAvailable timeAdvanceRequestAvailable [Available]}, [8.10/8.11] {@link RTI ambassador#nextMessageRequest nextMessageRequest} {@link RTI ambassador#nextMessageRequestAvailable nextMessageRequestAvailable [Available]} and [8.12] {@link RTI ambassador#flushQueueRequest flushQueueRequest} methods when an {@link RTI ambassador#enableTimeConstrained enableTimeConstrained} request is pending.</td>
</tr>
</tbody>
</table>
<exception_name>

RequestForTimeRegulationPending

Exception thrown by the (8.2) \{@link RTIambassador#enableTimeRegulation enableTimeRegulation\}, (8.8/8.9) \{@link RTIambassador#timeAdvanceRequest timeAdvanceRequest\} \{@link RTIambassador#timeAdvanceRequestAvailable timeAdvanceRequestAvailable \[Available\]\}, (8.10/8.11) \{@link RTIambassador#nextMessageRequest nextMessageRequest\} \{@link RTIambassador#nextMessageRequestAvailable timeAdvanceRequestAvailable \[Available\]\} and (8.12) \{@link RTIambassador#flushQueueRequest flushQueueRequest\} methods when an \{@link RTIambassador#enableTimeRegulation enableTimeRegulation\} request is pending.

RestoreInProgress

Exception thrown by nearly all of the \{@link RTIambassador\}'s methods when a federation restore is in progress. The methods that do not throw it are \{@link RTIambassador#createFederationExecution create\} / \{@link RTIambassador#destroyFederationExecution destroy\} / \{@link RTIambassador#resignFederationExecution resign\} FederationExecution, federateRestore \{@link RTIambassador#federateRestoreNotComplete federateRestoreNotComplete \[Not\]\} \{@link RTIambassador#federateComplete federateComplete \[Complete\]\} FederationExecution, \{@link RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus\}, all of the various <code>get</code> methods, normalize \{@link RTIambassador#normalizeFederateHandle normalizeFederateHandle\} / \{@link RTIambassador#normalizeServiceGroup normalizeServiceGroup\} ServiceGroup, \{@link RTIambassador#evokeCallback evokeCallback\} and \{@link RTIambassador#evokeMultipleCallbacks evokeMultipleCallbacks\}.

RestoreNotRequested

Exception thrown by the RTIambassador's (4.22) federateRestore \{@link RTIambassador#federateRestoreNotComplete federateRestoreNotComplete \[Not\]\} \{@link RTIambassador#federateComplete federateComplete \[Complete\]\} methods when a federation restore was not previously requested.

RTIinternalError

Exception thrown by all of the \{@link RTIambassador\}'s methods when something goes wrong and none of the more specific exceptions is appropriate.
<table>
<thead>
<tr>
<th><strong>&lt;exception_name&gt;</strong></th>
<th><strong>&lt;exception_description&gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveInProgress</td>
<td></td>
</tr>
<tr>
<td>Exception thrown by nearly all of the [@link RTIambassador][]'s methods when a federation save is in progress. The methods that do not throw it are [@link RTIambassador#createFederationExecution create] / [@link RTIambassador#destroyFederationExecution destroy] / [@link RTIambassador#resignFederationExecution resign] FederationExecution, federateSave [@link RTIambassador#federateSaveBegun Begun] / [@link RTIambassador#federateSaveNotComplete [Not]] / [@link RTIambassador#federateSaveComplete Complete] / [@link RTIambassador#queryFederationSaveStatus queryFederationSaveStatus], all of the various &lt;code&gt;get&lt;/code&gt; methods, normalize [@link RTIambassador#normalizeFederateHandle FederateHandle] / [@link RTIambassador#normalizeServiceGroup ServiceGroup], [@link RTIambassador#evokeCallback evokeCallback] and [@link RTIambassador#evokeMultipleCallbacks evokeMultipleCallbacks].</td>
<td></td>
</tr>
<tr>
<td>SaveNotInitiated</td>
<td></td>
</tr>
<tr>
<td>Exception thrown by the (4.13) [@link RTIambassador#federateSaveBegun federateSaveBegun] method when a federation save was not previously requested.</td>
<td></td>
</tr>
<tr>
<td>SpecifiedSaveLabelDoesNotExist</td>
<td></td>
</tr>
<tr>
<td>Exception that should be thrown by the (4.21) [@link FederateAmbassador#initiateFederateRestore initiateFederateRestore] callback if the federate cannot find the specified save label.</td>
<td></td>
</tr>
<tr>
<td>SynchronizationPointLabelNotAnnounced</td>
<td></td>
</tr>
<tr>
<td>Exception thrown by the (4.9) [@link RTIambassador#synchronizationPointAchieved synchronizationPointAchieved] method when the specified synchronization point was not previously announced.</td>
<td></td>
</tr>
<tr>
<td>TimeConstrainedAlreadyEnabled</td>
<td></td>
</tr>
<tr>
<td>Exception thrown by the (8.5) [@link RTIambassador#enableTimeConstrained enableTimeConstrained] method when time-constraint is already enabled.</td>
<td></td>
</tr>
<tr>
<td>TimeConstrainedIsNotEnabled</td>
<td></td>
</tr>
<tr>
<td>Exception thrown by the (8.7) [@link RTIambassador#disableTimeConstrained disableTimeConstrained] method when time-constraint is already disabled.</td>
<td></td>
</tr>
<tr>
<td>Exception Name</td>
<td>Exception Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TimeRegulationAlreadyEnabled</td>
<td>Exception thrown by the (8.2) <code>{@link RTIambassador#enableTimeRegulation enableTimeRegulation}</code> method when time-regulation is already enabled.</td>
</tr>
<tr>
<td>TimeRegulationIsNotEnabled</td>
<td>Exception thrown by the (8.4) <code>{@link RTIambassador#disableTimeRegulation disableTimeRegulation}</code>, (8.19/8.20) <code>{@link RTIambassador#modifyLookahead modifyLookahead}</code>, <code>{@link RTIambassador#queryLookahead queryLookahead}</code> and (8.21) <code>{@link RTIambassador#retract retract}</code> methods when time-regulation is already disabled.</td>
</tr>
<tr>
<td>UnableToPerformSave</td>
<td>Exception that should be thrown by the (4.12) <code>{@link FederateAmbassador#initiateFederateSave initiateFederateSave}</code> (both forms) callback if the federate considers itself unable to proceed into the Saving state. This exception should be thrown if a quick check by the federate leads it to conclude that an attempt to save is doomed to failure. Normally, the federate will start its save operation and later report success or failure through the (4.14) <code>{@link RTIambassador#federateSaveNotComplete [Not]} </code>{@link RTIambassador#federateSaveComplete complete}` methods.</td>
</tr>
<tr>
<td>UnknownName</td>
<td>Exception that should be thrown by the (6.3) <code>{@link FederateAmbassador#objectInstanceNameReservationSucceeded objectInstanceNameReservationSucceeded}</code> and <code>{@link FederateAmbassador#objectInstanceNameReservationFailed objectInstanceNameReservationFailed}</code> callbacks if the federate wishes to repudiate the name.</td>
</tr>
</tbody>
</table>
// File: InvalidLogicalTimeInterval.java
package hla.rti1516; //the package name was changed by DoD

Interpretations of IEEE 1516-2000v2

/**
 * NOTE: This exception is NOT part of IEEE 1516-2000 nor
 * specified by the DoD Interpretations; it is tentatively added
 * based on an analysis of apparent inconsistencies between the
 * Java, Ada and C++ specification clauses and normative annexes.
 * 
 * Exception thrown by the <code>LogicalTimeInterval.setTo</code>
 * and <code>LogicalTimeInterval.subtract</code> methods when
 * the supplied <code>LogicalTimeInterval</code> is invalid or
 * would result in an invalid one.
 * @author Daniel U. Thibault
 * @version 1516.1.5 (DoD v2+1)
 */
public final class InvalidLogicalTimeInterval
extends RTIexception
{
    /**
     * Constructs a new exception with the specified detail message.
     * The cause is not initialized, and may subsequently be
     * initialized by a call to
     * <code>java.lang.Throwable.initCause(java.lang.Throwable)</code>.
     * @param msg a <code>java.lang.String</code> holding the detail
     * message, which can be later retrieved by the <code>getMessage</code> method
     */
    public InvalidLogicalTimeInterval (String msg)
    {
        super(msg);
    }
}

//end InvalidLogicalTimeInterval
Annex C – DRDC HLA 1516 OMT Supporting Classes

The ca.gc.drdc_rddc.hla.rti1516.omt package implements the 1516.2-2000 Object Model Template (OMT) datatypes and their predefined encodings, as laid out in 4.12. Although a federation is free to define arbitrary data representation schemes for federate-to-federate interaction parameters and object attribute values, the RTI-owned objects and the RTI-issued interactions follow the 1516.2 predefined encodings. The standard currently provides no support whatsoever, but this may hopefully change with later editions. In the meantime, it is hoped the supporting classes laid out herein will prove flexible enough to fill the need.

HLA recognises five broad categories of datatypes: basic, simple, enumerated, array and record. In what follows, we’ll examine each set of classes briefly, and outline how user-defined classes should be implemented to follow the design patterns established here. The mandated datatypes and those added by the Management Object Model (MOM) were discussed in Annex A’s Encoding/Decoding Values sub-heading.

Basic datatypes

The basic datatypes are the underpinnings of all others, and generally map directly to a given language’s primitive types. They represent integers, floating-point numbers and characters, in various byte widths and byte sexes (big or little endian). They also form an almost closed set, in the sense that a federation will rarely have to declare additional basic datatypes.

The basic datatypes map readily to Java’s primitive types. We captured the functionality of the datatypes in the HLA datatype interface, a subset of which is the HLA encodable interface. This facilitates the construction of the more elaborate datatypes later on.

In order to facilitate the construction of user-defined array datatypes, we felt it necessary to add the HLA null basic datatype. It is trivial to represent, having no width (zero bits) and thus taking up no room in the byte[] representations being ferried around by the RTI.

Simple datatypes

The implementation of the simple datatypes was mostly a matter of casting as broad a net of constructors as possible. Intermediate abstract classes HLA octetPair, HLA integer16, HLA integer32, HLA integer64, HLA float32, and HLA float64 capture the behaviour (and underlying storage) common to the big-endian (BE) and little-endian (LE) forms.
Enumerated datatypes

These datatypes describe data elements that can take on a finite discrete set of possible values. They use a basic datatype for internal representation, and are defined by an Enumerator, an exhaustive list of names designating the immutable instance prototypes. The specification allows for a given enumeration member to take on multiple values (for example, the HLAboolean value HLAtrue could have been defined as “not zero”). This causes a number of problems, so we’ve chosen to restrict each enumeration member to a single value (although constructors may accept a wider range of input values). Also, the specification does not mention whether enumerated types are ordered or not. We chose here to assume the enumeration sequence is fixed and ordered.

The additional functionality of the enumerated datatypes is captured in the HLAenumerateddatatype interface, which extends HLADatatype by adding a couple of Iterator members.

In order to simplify as much as possible the task of creating new federation-specific enumerated datatypes, we implemented an HLAenumeratedIterator class that automates, using Java object reflection, the task of giving the class an Iterator interface. Although it has not been done here, adding an immutable static List member to each class would facilitate normalization and unnormalization considerably.

Three of the MOM-defined enumerated datatypes map directly to IEEE 1516 classes: HLAresignAction (ResignAction), HLAorderType (OrderType), and HLAServiceGroupName (ServiceGroup). It should be a very simple matter to modify the implementations given here in order to allow constructors to accept the mapped objects as inputs (e.g. it should be possible to write HLAServiceGroupName sgn_ss = new HLAServiceGroupName(ServiceGroup.SUPPORT_SERVICES).

Array datatypes

These datatypes describe indexed homogeneous collections of datatypes. The collection’s size is either fixed or variable, and the elements that make them up are all of the same datatype: simple, enumerated, array or record.

The additional functionality of the array datatypes is captured in the HLAarraydatatype interface, which extends HLADatatype and List. Common behaviour is again taken care of by ancestor abstract classes, in a two-tier structure (HLAArrayType and its descendents HLAVariableArrayType and HLAFixedArrayType). The implementation provides a concrete class, HLAObjectArray, which provides most of the behaviour required of non-trivial variable array types.
**Fixed record datatypes**

These datatypes describe heterogeneous collections of datatypes. The individual elements (considered to appear in a fixed sequence) are potentially each of a different datatype: simple, enumerated, array or record.

The additional functionality of the fixed record datatypes is captured in the **HLAfixedrecorddatatype** interface, which extends **HLAdatatype** and **Map**. Common behaviour is again taken care of by an ancestor abstract class, **HLAfixedRecordType**, which relies on **HLAfixedRecordIterator** to automate the construction of **Iterators**, in a manner analogous to enumerated datatypes.

**Variant record datatypes**

These datatypes describe discriminated unions of datatypes. They are characterised by a discriminant, an enumerated datatype value which is used to switch the interpretation of the record’s contents between a number of alternatives. Each alternative is either null (absent) or some other datatype: simple, enumerated, array or record.

The additional functionality of the variant record datatypes is captured in the **HLAvariantrecorddatatype** interface, which extends **HLAdatatype**. Common behaviour is again taken care of by an ancestor abstract class, **HLAvariantRecordType**.
The **ByteWrapper** class is the workhorse of the package; it implements all of the encoding and decoding behaviours.

```java
// File: ByteWrapper.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

/**
 * Utility class for managing data in byte arrays.
 * It is similar to java.nio.ByteBuffer but considerably simpler. The key components are:
 * A backing store (returned by the array() method), which is either a wrapped pre-existing byte[] or a newly created one.
 * An inclusive lower bound (returned by the lowerBound() method), which is the zero-based index of the first accessible byte.
 * An exclusive upper bound (returned by the upperBound() method), which is the zero-based index of the byte before which the last accessible byte lies.
 * A cursor (returned by the pos() method), which is the zero-based index of the current byte (i.e. the next byte to read or write).
 * The cursor always lies between the lower bound (included) and the upper bound (included).
 * When the ByteWrapper is created, its cursor is always set to the lower bound.
 */

Typical use:
```
```
* //To decode an HLA data type from a byte array<br>* ByteWrapper bw = new ByteWrapper(bytes);<br>* HLAinteger32BE post = new HLAinteger32BE(); //default value of zero<br>* post.decode(bw);<br>* //Alternately, decode directly from the byte array...<br>* HLAinteger32BE post = new HLAinteger32BE(bytes);<br>* //...or the byte wrapper<br>* HLAinteger32BE post = new HLAinteger32BE(bw);<br>* </code></ul>

@version 1.1
*/

public class
ByteWrapper
implements java.lang.Cloneable
{
    private byte[] _buffer;
    private int _lowerBound;
    private int _upperBound;
    private int _pos;

    /**
     * Constructs a new ByteWrapper backed by a new byte array of the
     * specified length.
     * Lower bound will be zero, upper bound will be length.
     * The entire array is read/write enabled.
     * @param length the size (in bytes) of the byte[] to create to
     * back the new ByteWrapper
     */
    public ByteWrapper(int length)
    {
        this(new byte[length]); //exception if length zero?
    }

    /**
     * Constructs a new ByteWrapper backed by the specified pre-
     * existing byte[] buffer.
     * Lower bound will be zero, upper bound will be the buffer.length.
     * (Changes to the Byte Wrapper will write through to buffer)
     * The entire array is read/write enabled.
     * @param buffer the byte[] to use to back the new ByteWrapper
     */
    public ByteWrapper(byte[] buffer)
    {
        this(buffer, 0, buffer.length);
    }
/**
 * Constructs a new ByteWrapper backed by the specified pre-existing byte[] buffer and
 * specifies the buffer's lower bound.
 * Upper bound will be the buffer.length.
 * (Changes to the Byte Wrapper will write through to buffer)
 * The array is read/write enabled from the lower bound (included)
 * to the upper bound (excluded).
 * @param buffer the byte[] to use to back the new ByteWrapper
 * @param lowerBound the first accessible position within the byte[]
 * @throws ArrayIndexOutOfBoundsException if the lowerBound is greater than the buffer.length
 */
 public ByteWrapper(byte[] buffer, int lowerBound)
 {
  this(buffer, lowerBound, buffer.length);
 }

/**
 * Constructs a new ByteWrapper backed by the specified pre-existing byte[] buffer and
 * specifies the buffer's lower and upper bounds.
 * (Changes to the Byte Wrapper will write through to buffer)
 * The array is read/write enabled from the lower bound (included)
 * to the upper bound (excluded).
 * @param buffer the byte[] to use to back the new ByteWrapper
 * @param lowerBound the first accessible position within the byte[]
 * @param upperBound the position within the byte[] before which the last accessible byte lies
 * @throws ArrayIndexOutOfBoundsException if the lowerBound is greater than the upperBound or if the upperBound is greater than the buffer.length
 */
 public ByteWrapper(byte[] buffer, int lowerBound, int upperBound)
 {
  _buffer = buffer;
  _lowerBound = lowerBound;
  _pos = _lowerBound;
  _upperBound = upperBound;
  if (_upperBound > buffer.length) throw new
  ArrayIndexOutOfBoundsException(_upperBound);
  verify(0);
 }
/**
 * Constructs a new ByteWrapper from the specified AttributeHandle Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the AttributeHandle to encode into the new ByteWrapper
 */

public ByteWrapper(hla.rti1516.AttributeHandle o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(0);
}

/**
 * Constructs a new ByteWrapper from the specified DimensionHandle Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the DimensionHandle to encode into the new ByteWrapper
 */

public ByteWrapper(hla.rti1516.DimensionHandle o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(0);
}
/**
 * Constructs a new ByteWrapper from the specified FederateHandle Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength. The entire array is read/write enabled.
 * @param o the FederateHandle to encode into the new ByteWrapper
 */
public ByteWrapper(hla.rti1516.FederateHandle o) {
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(o);
}

/**
 * Constructs a new ByteWrapper from the specified InteractionClassHandle Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength. The entire array is read/write enabled.
 * @param o the InteractionClassHandle to encode into the new ByteWrapper
 */
public ByteWrapper(hla.rti1516.InteractionClassHandle o) {
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(o);
}
/**
 * Constructs a new ByteWrapper from the specified LogicalTime Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the LogicalTime to encode into the new ByteWrapper
 */

public ByteWrapper(hla.rti1516.LogicalTime o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(0);
}

/**
 * Constructs a new ByteWrapper from the specified LogicalTimeInterval Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the LogicalTimeInterval to encode into the new ByteWrapper
 */

public ByteWrapper(hla.rti1516.LogicalTimeInterval o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(0);
}
public ByteWrapper(hla.rti1516.ObjectClassHandle o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}

/**
 * Constructs a new ByteWrapper from the specified
 * ObjectClassHandle Object.
 * This is a convenience constructor, provided to facilitate using
 * this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the ObjectClassHandle to encode into the new
 * ByteWrapper
 */

public ByteWrapper(hla.rti1516.ObjectInstanceHandle o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}
/**
 * Constructs a new ByteWrapper from the specified OrderType Object.
 * This is a convenience constructor, provided to facilitate using
 * this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param the OrderType to encode into the new ByteWrapper
 */
public ByteWrapper(hla.rti1516.OrderType o) {
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}

/**
 * Constructs a new ByteWrapper from the specified ParameterHandle Object.
 * This is a convenience constructor, provided to facilitate using
 * this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param the ParameterHandle to encode into the new ByteWrapper
 */
public ByteWrapper(hla.rti1516.ParameterHandle o) {
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}
/**
 * Constructs a new ByteWrapper from the specified TransportationType Object.
 * This is a convenience constructor, provided to facilitate using this HLA object.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the TransportationType to encode into the new ByteWrapper
 */
public ByteWrapper(hla.rti1516.TransportationType o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}

/**
 * Constructs a new ByteWrapper from the specified HLAEncodable Object.
 * This is a convenience constructor, provided to facilitate using the HLA objects.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * @param o the HLAEncodable Object to encode into the new ByteWrapper
 */
public ByteWrapper(HLAEncodable o)
{
    final byte[] bytes = new byte[o.encodedLength()];
    o.encode(bytes, 0);
    //Now we can pass this to the (byte[]) constructor:
    this(bytes);
    //Which passes it to the (byte[], int, int) constructor:
    this(bytes, 0, bytes.length);
    //Which does:
    _lowerBound = 0;
    _pos = _lowerBound;
    _upperBound = bytes.length;
    verify(O);
}
/**
 * Constructs a new ByteWrapper from the specified Object, which is
 * presumed to have
 * the encode(byte[] buffer, int offset) and encodedLength() methods.
 * This is a convenience constructor, provided to facilitate using
 * the HLA objects.
 * Lower bound will be zero, upper bound will be the encodedLength.
 * The entire array is read/write enabled.
 * <p>
 * Things would be an awful lot simpler if IEEE 1516 specified an
 * Encodable interface
 * consisting of those two methods (see the constructor that
 * precedes this one).
 * @param o the Object to encode into the new ByteWrapper
 * @throws IllegalArgumentException if the Object does not support
 * the encode and encodedLength methods
 */
public ByteWrapper(Object o) {
    try {
        java.lang.reflect.Method eL =
            o.getClass().getMethod("encodedLength", null);
        // throws NoSuchMethodException, SecurityException
        final byte[] bytes = new byte[((Integer) (eL.invoke(o,
            null))).intValue()];
        java.lang.reflect.Method e = o.getClass().getMethod("encode",
            new Class[]{ byte[].class, int.class });
        e.invoke(o, new Object[] { bytes, new Integer(0) });
        // throws IllegalAccessException, IllegalArgumentException,
        InvocationTargetException
        // The rest wouldn't need to be inside the try if it weren't
        // for bytes being local to it
        // Now we can pass this to the (byte[]) constructor:
        // this(bytes);
        // Which passes it to the (byte[], int, int) constructor:
        // this(bytes, 0, bytes.length);
        // Which does:
        _lowerBound = 0;
        _pos = _lowerBound;
        _upperBound = bytes.length;
        verify(0);
    } catch (Exception ex) {
        // For some unfathomable reason, throwing this here and
        // declaring it above does not compile.
        // Since all HLA objects do support HLAencodable, it cannot
        // occur anyway.
        throw new IllegalArgumentException(ex.getLocalizedMessage()).initCause(ex);
    }
}
/**
 * Returns the complete backing array.
 * The returned array runs from zero to length, not just from the
 * lower to the upper bound.
 * @return the complete byte[] backing the ByteWrapper
 */
public final byte[]
array()
{
    return _buffer;
}

/**
 * Returns the lower bound.
 * @return the ByteWrapper's lower bound
 */
public final int
lowerBound()
{
    return _lowerBound;
}

/**
 * Returns the upper bound.
 * @return the ByteWrapper's upper bound
 */
public final int
upperBound()
{
    return _upperBound;
}

/**
 * Returns the current position.
 * @return the current read/write position of the ByteWrapper
 */
public final int
pos()
{
    return _pos;
}

/**
 * Resets the current position to the first accessible byte (the
 * lower bound) of the ByteWrapper.
 */
public void
reset()
{
    _pos = _lowerBound;
}
private final void verify(int length) {
    if ((pos + length) > upperBound) throw new ArrayIndexOutOfBoundsException(_pos + length);
}

private long getBytes(int n, boolean bigendian) throws IllegalArgumentException {
    if ((n < 0) || (n > 8)) throw new IllegalArgumentException(Integer.toString(n));
    verify(n);
    long value = 0;
    if (bigendian) {
        for (int nn = n - 1; nn >= 0; nn--) {
            value += (_buffer[_pos++] & 0x00FFL) << (8*nn);
        }
    } else {
        for (int nn = 0; nn < n; nn++) {
            value += (_buffer[_pos++] & 0x00FFL) << (8*nn);
        }
    }
    return value;
}
/**
 * Gets the next eight bytes from the wrapped byte array as a big-endian 64-bit integer.
 * The ByteWrapper's current position is increased by 8.
 * @return a long decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up a long
 */
public final long
getLong()
{
    return getBytes(8, true);
}

/**
 * Gets the next eight bytes from the wrapped byte array as a little-endian 64-bit integer.
 * The ByteWrapper's current position is increased by 8.
 * @return a long decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up a long
 */
public final long
getLongLE()
{
    return getBytes(8, false);
}

/**
 * Gets the next four bytes from the wrapped byte array as a big-endian 32-bit integer.
 * The ByteWrapper's current position is increased by 4.
 * @return an int decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up an int
 */
public final int
getInt()
{
    return (int)getBytes(4, true);
}

/**
 * Gets the next four bytes from the wrapped byte array as a little-endian 32-bit integer.
 * The ByteWrapper's current position is increased by 4.
 * @return an int decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up an int
 */
public final int
getIntLE()
{
    return (int)getBytes(4, false);
}
/**
 * Gets the next two bytes from the wrapped byte array as a big-endian 16-bit integer.
 * The ByteWrapper's current position is increased by 2.
 * @return a short decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up a short
 */
public final short getShort()
{
    return (short)getBytes(2, true);
}

/**
 * Gets the next two bytes from the wrapped byte array as a little-endian 16-bit integer.
 * The ByteWrapper's current position is increased by 2.
 * @return a short decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up a short
 */
public final short getShortLE()
{
    return (short)getBytes(2, false);
}

/**
 * Gets the next two bytes from the wrapped byte array as an unsigned 16-bit char.
 * The ByteWrapper's current position is increased by 2.
 * @return a char decoded value
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to make up a char
 */
public final char getChar()
{
    return (char)getBytes(2, true);
}

/**
 * Gets the next byte from the wrapped byte array.
 * The ByteWrapper's current position is increased by 1.
 * @return an int decoded value (representing a single byte)
 * @throws ArrayIndexOutOfBoundsException if there isn't at least one byte left in the wrapped byte array
 */
public final byte getByte()
{
    // return (byte)getBytes(1, true);
    // verify(1);
    // return (_buffer[_pos++] & 0xFF);
    return _buffer[_pos++];
}
/**
 * Writes to the specified byte array from the wrapped byte array.
 * The ByteWrapper's current position is increased by dest.length.
 * @param dest the byte[] to fill from the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to fill the dest byte[]
 */
public final ByteWrapper write(byte[] dest)
{
    verify(dest.length);
    System.arraycopy(_buffer, _pos, dest, 0, dest.length);
    _pos += dest.length;
    return this;
}

/**
 * Puts the value into the wrapped byte array; it is n bytes wide (a power of two, normally)
 * and is big-endian if the boolean is true.
 * @param value a long holding the value to put in
 * @param n the number of bytes to put in
 * @param bigendian whether the value should be written big-endian or not
 * @throws IllegalArgumentException if n is not within the 0..8 range
 * @throws ArrayIndexOutOfBoundsException if there are not at least n bytes left in the wrapped byte array
 */
private void putBytes(long value, int n, boolean bigendian)
throws IllegalArgumentException
{
    if ((n < 0) || (n > 8)) throw new IllegalArgumentException(Integer.toString(n));
    verify(n);
    if (bigendian)
    {
        for (int nn = n - 1; nn >= 0; nn--)
            _buffer[_pos++] = (byte) ((value >>> (8*nn)) & 0xFF);
    }
    else
    {
        for (int nn = 0; nn < n; nn++)
            _buffer[_pos++] = (byte) ((value >>> (8*nn)) & 0xFF);
    }
}
/**
 * Puts value into the wrapped byte array as a big-endian 64-bit integer.
 * The ByteWrapper's current position is increased by 8.
 * @param value the long to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write a long into
 */
public final ByteWrapper putLong(long value)
{
    putBytes(value, 8, true);
    return this;
}

/**
 * Puts value into the wrapped byte array as a little-endian 64-bit integer.
 * The ByteWrapper's current position is increased by 8.
 * @param value the long to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write a long into
 */
public final ByteWrapper putLongLE(long value)
{
    putBytes(value, 8, false);
    return this;
}

/**
 * Puts value into the wrapped byte array as a big-endian 32-bit integer.
 * The ByteWrapper's current position is increased by 4.
 * @param value the int to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write an int into
 */
public final ByteWrapper putInt(int value)
{
    putBytes(value, 4, true);
    return this;
}
/**
 * Puts value into the wrapped byte array as a little-endian 32-bit integer.
 * The ByteWrapper's current position is increased by 4.
 * @param value the int to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write an int into
 */
public final ByteWrapper putIntLE(int value)
{
    putBytes(value, 4, false);
    return this;
}

/**
 * Puts value into the wrapped byte array as a big-endian 16-bit integer.
 * The ByteWrapper's current position is increased by 2.
 * @param value the short to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write a short into
 */
public final ByteWrapper putShort(int value)
{
    putBytes(value, 2, true);
    return this;
}

/**
 * Puts value into the wrapped byte array as a little-endian 16-bit integer.
 * The ByteWrapper's current position is increased by 2.
 * @param value the short to write to the wrapped byte array
 * @return this ByteWrapper
 * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to write a short into
 */
public final ByteWrapper putShortLE(short value)
{
    putBytes(value, 2, false);
    return this;
}
/**
* Puts value into the wrapped byte array as an unsigned 16-bit char.
* The ByteWrapper's current position is increased by 2.
* @param value the char to write to the wrapped byte array
* @return this ByteWrapper
* @throws ArrayIndexOutOfBoundsException if there are not enough
* bytes left in the wrapped byte array to write a char into
*/
public final ByteWrapper putChar(char value)
{
    putBytes(value, 2, true);
    return this;
}

/**
* Puts the byte b into the wrapped byte array.
* The ByteWrapper's current position is increased by 1.
* @param b int to put into the wrapped byte array's current byte
* @return this ByteWrapper
* @throws ArrayIndexOutOfBoundsException if there isn't at least
* one byte left in the wrapped byte array to write to
*/
public final ByteWrapper putByte(byte b)
{
    //     putBytes(value, 1, true);
    verify(1);
    _buffer[_pos++] = b;
    return this;
}

/**
* Reads a byte array into the wrapped byte array.
* The ByteWrapper's current position increases by the array's size.
* @param src byte[] to read into the wrapped byte array
* @return this ByteWrapper
* @throws ArrayIndexOutOfBoundsException if there are not enough
* bytes left in the wrapped byte array to read the src byte[] into
*/
public final ByteWrapper read(byte[] src)
{
    verify(src.length);
    System.arraycopy(src, 0, _buffer, _pos, src.length);
    _pos += src.length;
    return this;
}
/**
   * Skips n bytes without reading from or writing to them.
   * @param n an int specifying by how many bytes to advance the current position
   * @return this ByteWrapper
   * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to skip ahead as requested
   */
   public final ByteWrapper skip(int n)
   {
     verify(n);
     _pos += n;
     return this;
   }

   /**
   * Advances the current position until the specified byte alignment is achieved (typically a power of 2).
   * @param alignment an int specifying what the current position should be a whole multiple of
   * @throws ArrayIndexOutOfBoundsException if there are not enough bytes left in the wrapped byte array to align as requested
   */
   public final void align(int alignment)
   {
     while ((_pos % alignment) != 0) skip(1);
   }

   /**
   * Creates a new ByteWrapper backed by the same byte array but using the current position as its lower bound.
   * The upper bound is inherited.
   * @return a new ByteWrapper backed by the same byte[] and whose lower bound is the current position
   */
   public final ByteWrapper slice()
   {
     return new ByteWrapper(_buffer, _pos, _upperBound);
   }
/**
 * Creates a new ByteWrapper backed by the same byte array but using the current position as its lower bound and the specified length to mark the upper bound.
 * @param length an int specifying the new ByteWrapper's upper bound, measured from the current position
 * @return a new ByteWrapper backed by the same byte[], but whose accessible bytes run from the current position (included) to length bytes further
 * @throws ArrayIndexOutOfBoundsException if there are not at least length bytes left in the ByteWrapper
 */
public final ByteWrapper slice(int length) {
  verify(length);
  return new ByteWrapper(_buffer, _pos, _pos + length);
}

//java.lang.Object methods
/**
 * Creates and returns a copy of this object. The cloned Object will be backed by a copy of the original byte[].
 * @return an independent copy of this Object
 * @throws CloneNotSupportedException if the object's class is not Cloneable or if the instance cannot be cloned
 */
public Object clone() throws CloneNotSupportedException {  
  //ByteWrapper theClone = new ByteWrapper(_buffer.length);
  //theClone._lowerBound = this._lowerBound;
  //theClone._pos = this._pos;
  //theClone._upperBound = this._upperBound;
  System.arraycopy(this._buffer, 0, theClone._buffer, 0, theClone._buffer.length);
  //return theClone;
  return (Object)theClone;
}
/**
 * Compares the specified object with this one for equality.
 * Returns <tt>true</tt> if and only if the specified object is also a
 * ByteWrapper (or descendant), and both have the same backing
 * store and bounds.
 * @param obj the Object to be compared for equality with this one
 * @return <tt>true</tt> if the specified Object is equal to this one
 */
public boolean equals(Object obj) {
    //The java.lang.Object implementation is simply:
    // return (this == obj);

    //Trivial case: both this and obj are references to the same object
    if (obj == this) return true;

    //obj must be an instance of this class or a descendant.
    //However, we do NOT want sibling classes to be "equal".
    //If we did this:
    // if (! (obj instanceof ByteWrapper)) return false;
    //Then two sibling classes (that descend from ByteWrapper along diverging paths)
    //would nevertheless be considered equal since their inherited equals()
    //would compare them to their common super-class, ByteWrapper.

    //Doing this:
    // if (! this.getClass().isInstance(obj)) return false;
    //Insures that obj's class descends from this' class, but the method is then not symmetric
    //So we match the classes instead:
    if (! this.getClass().equals(obj.getClass())) return false;

    //So far so good; now compare relevant fields
    return ((_buffer == ((ByteWrapper)obj).array()) &&
            (_lowerBound == ((ByteWrapper)obj).lowerBound()) &&
            (_upperBound == ((ByteWrapper)obj).upperBound()));
    //The cursor (i.e. _pos vs obj.pos()) is not relevant
}

//end ByteWrapper
The HLAencodable interface is supported by just about all HLA objects.

// File: HLAencodable.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

/**
 * Interface which just about all HLA interfaces and object types support.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier})
 * @version 1.1
 */
public interface HLAencodable {

  /**
   * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.
   * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
   */
  public int encodedLength();

  /**
   * Encodes <code>this</code> into the <code>byte[]</code> at the specified <code>offset</code>.
   * @param buffer the <code>byte[]</code> into which to encode <code>this</code>
   * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
   * @return how many bytes were written to the buffer, including any prefix padding bytes
   */
  public int encode(byte[] buffer, int offset);
}

//end HLAencodable
The HLAdatatype interface is the minimal one supported by all HLA datatypes.

// File: HLAdatatype.java
package ca.gc.drdc.rddc.hla.rti1516.omt;

import hla.rti1516.CouldNotDecode;

/**
 * Interface implemented by the various HLA data types.
 * In addition to this interface, the HLA data type classes are expected to supply:
 * <ul>
 * <li> Constructors (default and specified-value)</li>
 * <li> Constructor (byte[] buffer) throws CouldNotDecode</li>
 * <li> Constructor (byte[] buffer, int offset) throws CouldNotDecode</li>
 * <li> Constructor (ByteWrapper byteWrapper) throws CouldNotDecode</li>
 * <li> java.lang.Object methods toString(); equals(Object otherObject) and hashCode() </li>
 * <li> Class-specific extensions to get and set the value as a basic Java data type (int, boolean, etc.)</li>
 * </ul>
 * Note that even though encodedLength (sometimes) and octetBoundary (always) could be static methods, Java forbids an interface method from being static, and also forbids an interface method implementation from being static.
 * This is because static invocation mode isn't like virtual invocation mode in the sense that the method invoked will be the *local* one instead of the run-time class'. For example, suppose XX descends from X, and that X defines methods M and MM.
 * XX overrides MM but not M. When XX.M is invoked, its resolution is passed to X.M. If MM is non-static, then a (virtual) invocation of MM by X.M will invoke XX.MM. If MM is static, on the other hand, then a (static) invocation of MM by X.M will invoke X.MM instead. Thus, to ensure proper polymorphism one must not use static methods.
 * Nevertheless, to allow obtaining a class' octetBoundary and/or encodedLength, when these are defined at the class level, one need only declare (public static final) *fields* of those names.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public interface HLADatatype
  extends HLAencodable
{
/**
 * As a general rule, Variant Records [HLAvariantRecord], Dynamic Arrays [HLAvariableArray], and any
 * Fixed Arrays [HLAfixedArray] or Fixed Records [HLAfixedRecord]
 * containing either of these first two
 * cannot have a constant encodedLength.
 */
/**
 * Returns the octet boundary of <code>this</code>.  
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.  
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.
 * @return the octet boundary of <code>this</code>
 */
int octetBoundary();
/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> into which to
 * encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
ByteWrapper encode(ByteWrapper byteWrapper);
/**
 * Encodes <code>this</code> into a new <code>byte[]</code>.  
 * @return a <code>byte[]</code> encoding <code>this</code>
 */
byte[] toByteArray();
/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the data type  
 * @return how many bytes were read from the <code>byte[]</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
public int decode(byte[] buffer)
    throws CouldNotDecode;
/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> <code>this</code> representation begins
 * @return where in the <code>buffer</code> <code>this</code> representation ends
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public int
decode (byte[] buffer,
        int offset)
        throws CouldNotDecode;

/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper
decode (ByteWrapper byteWrapper)
        throws CouldNotDecode;

//end HLADatatype
The HLAbasicType abstract class regroups behaviour common to the HLA basic datatypes.

```java
// File: HLAbasicType.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract ancestor for the type-safe simple and basic data types.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public abstract class HLAbasicType
    implements java.io.Serializable,
              java.lang.Cloneable,
              HLAdatatype
{
    //HLAdatatype interface implementation

    /**
     * Encodes <code>this</code> into the <code>byte[]</code> at the
     * specified <code>offset</code>.
     * @param buffer the <code>byte[]</code> into which to encode
     * <code>this</code>
     * @param offset the offset into the <code>byte[]</code> at which
     * to encode <code>this</code>
     * @return how many bytes were written to the buffer, including any
     * prefix padding bytes
     */
    public int encode(byte[] buffer, int offset)
    {
        return encode(new ByteWrapper(buffer, offset)).pos() - offset;
    }

    /**
     * Encodes <code>this</code> into a new <code>byte[]</code>.
     * @return a <code>byte[]</code> encoding <code>this</code>
     */
    public byte[] toByteArray()
    {
        return encode(new ByteWrapper(encodedLength())).array();
    }
}
/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @return how many bytes were read from the <code>byte[]</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public int
    decode (byte[] buffer)
    throws CouldNotDecode
    {
    return decode (buffer, 0);
    }

/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> <code>this</code> representation begins
 * @return where in the <code>buffer</code> <code>this</code> representation ends
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public int
    decode (byte[] buffer, 
            int     offset)
    throws CouldNotDecode
    {
    try
    {
    return decode(new ByteWrapper(buffer, offset)).pos();
    }
    catch (Exception e)
    {
    CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
    throw (CouldNotDecode)cnd.initCause(e);
    }
    }
//Cloneable implementation

/**
 * Creates and returns a copy of this object.
 * @return an independent copy of this Object
 * @throws CloneNotSupportedException if the object's class is not
 * Cloneable or if the instance cannot be cloned
 */
public Object clone()
    throws CloneNotSupportedException
{
    // throw new CloneNotSupportedException();
    return super.clone();
}
//end HLAbasicType
The next twenty classes implement the HLA basic datatypes, including our HLAnull addition.

The pattern is constant: each class declares a protected value-holder field, final static fields for encodedLength and octetBoundary, then the constructors (default, class-specific, byte[], byte[] with offset, ByteWrapper), and the java.lang.Object methods toString, equals and hashCode. Next, the HLAdatatype interface, already partially implemented by HLAbasicType, is completed: the encodedLength and octetBoundary methods simply reflect the static fields (something which the more complex datatypes cannot do), and the encode(ByteWrapper) and decode(ByteWrapper) methods (which HLAbasicType delegates to) are supplied. Finally, any class-specific extensions (get/setValue) are added.

Each of the big-endian/little-endian datatype pairs is implemented first through an abstract class that regroups common fields and methods.

// File: HLAnull.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe null basic data type.
 * <p>
 * <code>HLAnull</code> is not part of the HLA spec; it is used only to make the HLAobjectArray class concrete by providing a zero-size implementation of HLAdatatype.
 * @author Daniel U. Thibault (mailto:Daniel.Thibault@DRDC-RDDC.gc.ca)
 * @version 1.1
 */
public class HLAnull extends HLAbasicType
{
    /** Length (in bytes) of the <code>byte[]</code> representation of this class. */
    public final static int encodedLength = 0;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 1;
/**
 * Constructs a <code>HLAnull</code>.
 */
public HLAnull()
{
    //super() not called because super-class is abstract
}

/*
 * Creates a <code>HLAnull</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
representation of the <code>HLAnull</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAnull(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/*
 * Creates a <code>HLAnull</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
representation of the <code>HLAnull</code>
 * @param offset where in the <code>buffer</code> the <code>HLAnull</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAnull(byte[] buffer,
    int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a <code>HLAnull</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAnull</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAnull(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}

//@java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {@link java.lang.String} reflecting <code>this</code> value
 */
public String toString()
{
    return "null";
}

/**
 * Returns true iff <code>this</code> and <code>otherObject</code> represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code> is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (this == otherObject) return true;
    //This form is more inheritable: match classes
    return this.getClass().equals(otherObject.getClass());
}
/**
 * Returns a hash code for <code>this</code>; two objects for which
 * <code>equals()</code> is <code>true</code> should yield the same hash
 * code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
Object.equals(java.lang.Object)
 * @see java.util.Hashtable Hashtable
 */
public int
hashCode()
{
    return 0;
}

//HLA datatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public final int
encodedLength()
{
    return encodedLength;
}

/**
 * Returns the octet boundary of <code>this</code>.
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary()); // Won't move cursor
    return byteWrapper; // No change
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public ByteWrapper decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary()); //Won't move
        return byteWrapper; //No change
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//end HLAnull
package ca.gc.drdc_rddc.hla.rti1516.omti

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 8-bit byte basic data type.
 * 
 * @author Daniel U. Thibault
 * @version 1.1
 */
public class HLAoctet extends HLAbasicType
{
    /** The 8-bit byte value: a Java byte. */
    protected byte _value;

    /** Length (in bytes) of the byte[] representation of this class. */
    public final static int encodedLength = 1;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 1;

    /**
     * Constructs a HLAoctet of default value (zero).
     */
    public HLAoctet()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /**
     * Constructs a HLAoctet of the specified value.
     * @param value a byte specifying this's value
     */
    public HLAoctet(byte value)
    {
        this();
        setValue(value);
    }
/**
   * Creates a <code>HLAoctet</code> from the network representation
   * in the provided <code>byte[]</code>.  
   * @param buffer the network-provided <code>byte[]</code> 
   * representation of the <code>HLAoctet</code>  
   * @throws CouldNotDecode if the <code>buffer</code> could not be 
   * decoded  
   */
   public
   HLAoctet(byte[] buffer)
   throws CouldNotDecode
   {
      this(buffer, 0);
   }

   /**
   * Creates a <code>HLAoctet</code> from the network representation
   * in the provided <code>byte[]</code> at the indicated 
   * <code>offset</code>.  
   * @param buffer the network-provided <code>byte[]</code> 
   * representation of the <code>HLAoctet</code>  
   * @param offset where in the <code>buffer</code> the 
   * <code>HLAoctet</code> representation begins  
   * @throws CouldNotDecode if the <code>buffer</code> could not be 
   * decoded  
   */
   public
   HLAoctet(byte[] buffer, int offset)
   throws CouldNotDecode
   {
      this(new ByteWrapper(buffer, offset));
   }

   /**
   * Creates a <code>HLAoctet</code> from the supplied 
   * <code>ByteWrapper</code>.  
   * @param byteWrapper the <code>ByteWrapper</code> at whose current 
   * <code>pos()</code> the representation of the <code>HLAoctet</code> 
   * begins  
   * @throws CouldNotDecode if the <code>ByteWrapper</code> could not 
   * be decoded  
   */
   public
   HLAoctet(ByteWrapper byteWrapper)
   throws CouldNotDecode
   {
      this();
      decode(byteWrapper);
   }
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.  
 * @return a [link java.lang.String] reflecting <code>this</code>
 * value
 */
public String toString ()
{
    return Byte.toString(getValue());
}

/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
 * represent the same object.  
 * @param otherObject the <code>Object</code> to compare with 
 * @return <code>true</code> iff supplied <code>otherObject</code>
 * is of the same type as <code>this</code> and has the same value
 * @see Object.equals
 * @see java.util.Hashtable
 */
public boolean equals (Object otherObject)
{
    if (this == otherObject) return true;
    // if (!(otherObject instanceof HLAoctet)) return false;
    // final HLAoctet other = (HLAoctet)otherObject;
    // return (getValue() == other.getValue());
    // //This form is more inheritable: first, match the classes 
    // if (! this.getClass().equals(otherObject.getClass())) return false;
    return (getValue() == ((HLAoctet)otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which
 * <code>equals() </code> is <code>true</code> should yield the same hash
 * code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 * @see java.lang.Object
 */
public int hashCode()
{
    return new Byte(getValue()).hashCode();
}
//HLAdatatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code>
representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code>
representation of <code>this</code>
*/
public final int
encodedLength()
{
    return encodedLength;
}

/**
 * Returns the octet boundary of <code>this</code>.  
 * The octet boundary value is defined as the smallest power of 2
which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary
value of all components within it.
 * @return the octet boundary of <code>this</code>
*/
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to
encode <code>this</code>
 * @return the <code>ByteWrapper</code>
*/
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return byteWrapper.putByte(getValue());
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins  
 * @return the <code>ByteWrapper</code>  
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public ByteWrapper
    decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        setValue(byteWrapper.getByte());
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//Class-specific extensions

/**
 * Returns the byte value of <code>this</code>.  
 * @return the byte value of <code>this</code>  
 * @see #setValue
 */

public byte
    getValue()
{
    return _value;
}

/**
 * Sets the byte value of <code>this</code>.  
 * @param value the byte new value for <code>this</code>
 * @see #getValue
 */

public void
    setValue(byte value)
{
    _value = value;
}

//end HLAoctet
// File: HLAoctetPair.java
package ca.gc.drdc.rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract type-safe 16-bit octet-pair basic data type.
 * It uses the <code>char</code> Java type (an unsigned 16-bit integer) to store the value
 * and is the ancestor of the HLAoctetPairBE and HLAoctetPairLE concrete classes.
 * <p>
 * Note that HLAoctetPair is the only data type to actually store the unmodified byte sequence;
 * all other types instead store the value represented by the byte sequence.
 * <p>
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public abstract class HLAoctetPair
    extends HLAbasicType
{
    /**
     * The 16-bit octet-pair value: a Java char.
     * As an unsigned 16-bit, the bytes will be in appropriate endian order.
     */
    protected char _value;

    /** Length (in bytes) of the <code>byte[]</code> representation of this class. */
    public final static int encodedLength = 2;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 2;
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a <code>String</code> reflecting <code>this</code>
 * value
 */
public String toString()
{
    String s = "0x";
    if (getFirstByte() < 0x0010) s = s + "0";
    s = s + Integer.toHexString(getFirstByte());
    s = s + " 0x";
    if (getSecondByte() < 0x0010) s = s + "0";
    s = s + Integer.toHexString(getSecondByte());
    return s;
}

/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
 * represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code>
 * is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (this == otherObject) return true;
    if (!this.getClass().equals(otherObject.getClass())) return false;
    // if (this.isBigEndian() !=
    // ((HLAoctetPair)otherObject).isBigEndian()) return false;
    return (getValue() == ((HLAoctetPair)otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which
 * <code>equals()</code> is <code>true</code> should yield the same hash
 * code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 * Object.equals(java.lang.Object)
 * @see java.util.Hashtable Hashtable
 */
public int hashCode()
{
    return new Character(getValue()).hashCode();
}
//HLAdatatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>. 
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public final int
encodedLength()
{
    return encodedLength;
}

/**
 * Returns the octet boundary of <code>this</code>. 
 * The octet boundary value is defined as the smallest power of 2 which is greater than or equal to the size of the datatype in bytes. 
 * For a constructed datatype, it is the maximum octet boundary value of all components within it. 
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code> 
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return byteWrapper.putChar(getValue());
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins 
 * @return the <code>ByteWrapper</code> 
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded 
 */
public ByteWrapper decode (ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        setValue(byteWrapper.getChar());
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//Class-specific extensions

/**
 * Returns the char value of <code>this</code>.  
 * The char's bytes will be in the appropriate endian order; in other words, returns the unmodified byte sequence.  
 * @return the char value of <code>this</code>  
 * @see #setValue 
 */
protected char getValue()
{
    return _value;
}

/**
 * Returns the first byte value of <code>this</code>.  
 * @return the first byte value of <code>this</code>  
 * @see #setFirstByte 
 */
public byte getFirstByte()
{
    return (byte)(_value >>> 8); // >>> is right-shift with zero-extension 
}
/**
 * Returns the second byte value of <code>this</code>.
 * @return the second byte value of <code>this</code>
 * @see #setSecondByte
 */
public byte getSecondByte()
{
    return (byte)(_value & 0xFF);
}

/**
 * Returns the high byte value of <code>this</code>.
 * @return the high byte value of <code>this</code>
 * @see #setHighByte
 */
public byte getHighByte()
{
    return (isBigEndian() ? getFirstByte() : getSecondByte());
}

/**
 * Returns the low byte value of <code>this</code>.
 * @return the low byte value of <code>this</code>
 * @see #setLowByte
 */
public byte getLowByte()
{
    return (isBigEndian() ? getSecondByte() : getFirstByte());
}

/**
 * Sets the char value of <code>this</code>.
 * The char's bytes will be stored in unmodified sequence;
 * in other words, the char should be the appropriate-endian
 * representation of the octet pair.
 * @param value the char new value for <code>this</code>
 * @see #getValue
 */
protected void setValue(char value)
{
    _value = value;
}
/**
 * Sets the first byte value of <code>this</code>.  
 * @param firstByte the new byte value for <code>this</code>' first byte
 * @see #getFirstByte
 */
public void setFirstByte(byte firstByte) {
    _value = (char)((_value & 0xFF00) | ((int)firstByte) << 8));
}

/**
 * Sets the second byte value of <code>this</code>.  
 * @param secondByte the new byte value for <code>this</code>' second byte  
 * @see #getSecondByte
 */
public void setSecondByte(byte secondByte) {
    _value = (char)((_value & 0xFF00) | secondByte);
}

/**
 * Sets the high byte value of <code>this</code>.  
 * @param highByte the new byte value for <code>this</code>' high byte  
 * @see #getHighByte
 */
public void setHighByte(byte highByte) {
    if(isBigEndian()) { setFirstByte(highByte); } else {
    setSecondByte(highByte); }
}

/**
 * Sets the low byte value of <code>this</code>.  
 * @param lowByte the new byte value for <code>this</code>' low byte  
 * @see #getLowByte
 */
public void setLowByte(byte lowByte) {
    if(isBigEndian()) { setSecondByte(lowByte); } else {
    setFirstByte(lowByte); }
}
/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public abstract boolean isBigEndian();

// If we declare isBigEndian() static, it won't work: static invocation mode isn't like virtual invocation mode in the sense
// that the method invoked will be the *local* one instead of the run-time class'. In other words, when a descendant of
// HL AoctetPair invokes one of the inherited methods defined here, if isBigEndian is static then HL AoctetPair's isBigEndian
// will be invoked, not the descendant's...

// Likewise, we cannot rely on a field, because fields are never virtual in their invocation mode.
// Finally, declaring a "blank final" (non-static) field set by the constructors works for this class but fails for the
// descendants (which cannot assign to the inherited final).
}
//end HL AoctetPair
/**
 * Type-safe 16-bit big-endian octet-pair basic data type.
 * Note that HLAoctetPair is the only data type to actually store the
 * unmodified byte sequence;
 * all other types instead store the value represented by the byte
 * sequence.
 *<p>
 *<code>HLAoctetPairBE</code>s are normally never obtained directly,
 *as the RTI uses simple data types instead.
 *</p>
 * They could also be obtained by using an eventual
 *<code>HLAoctetPairBEfactory</code>'s decode method on a
 *<code>byte[]</code>
 * received as part of an attribute update or interaction. However, no
 * such factory is currently supplied by the RTIambassador.
 *
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} {{link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier}}
 * @version 1.1
 */
public class
HLAoctetPairBE
extends HLAoctetPair
{
    /**
     * Constructs a <code>HLAoctetPairBE</code> of default value (zero
     * or Unicode null).
     */
    public
HLAoctetPairBE()
    {
        //super() not called because super-class is abstract
        //setValue('\u0000');
    }

    /**
     * Constructs a <code>HLAoctetPairBE</code> of the specified char
     * value.
     * @param value a char specifying <code>this</code>' value
     */
    public
HLAoctetPairBE(char value)
    {
        this();
        setValue(value);
    }
/**
 * Constructs a <code>HLAoctetPairBE</code> of the specified short value.
 * The short's bytes are stored in unmodified sequence (thus first -> high; second -> low).
 * @param value a short specifying <code>this</code>' value
 * /\n * public
 * HLAoctetPairBE(short value)
 * {\n * this((char)value);
 * }
 */

/**
 * Constructs a <code>HLAoctetPairBE</code> of the specified value, using high and low bytes.
 * @param highByte a byte specifying the high byte of <code>this</code>' value
 * @param lowByte a byte specifying the low byte of <code>this</code>' value
 * /\n * public
 * HLAoctetPairBE(byte highByte,
 * byte lowByte)
 * {
 * //Note the byte order reversal when compared with HLAoctetPairLE
 * this( (char)(( ((int)highByte) « 8 ) | lowByte) );
 * }
 */

/**
 * Creates a <code>HLAoctetPairBE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAoctetPairBE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 * /\n * public
 * HLAoctetPairBE(byte[] buffer)
 * throws CouldNotDecode
 * {
 * this(buffer, 0);
 * }
/**
 * Creates a <code>HLAoctetPairBE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAoctetPairBE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAoctetPairBE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAoctetPairBE(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAoctetPairBE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAoctetPairBE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAoctetPairBE(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

//Class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian() {
    return true;
}
} //end HLAoctetPairBE
package ca.gc.drdc_rddc.hla.rti1516.omti
import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 16-bit little-endian octet-pair basic data type.
 *<p>
 * <code>HLAoctetPairLE</code>s are normally never obtained directly,
as the RTI uses simple data types instead.
 *<p>
 * They could also be obtained by using an eventual
<code>HLAoctetPairLEfactory</code>’s decode method on a
<code>byte[]</code>
 * received as part of an attribute update or interaction. However, no
such factory is currently supplied by the RTI ambassador.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
Valcartier])
 * @version 1.1
 */
public class HLAoctetPairLE
    extends HLAoctetPair
{
    /**
     * Constructs a <code>HLAoctetPairLE</code> of default value (zero
or Unicode null).
     */
    public HLAoctetPairLE()
    {
        //super() not called because super-class is abstract
        //setValue(' ');
    }

    /**
     * Constructs a <code>HLAoctetPairLE</code> of the specified char
value.
     * @param value a char specifying <code>this</code>’s value
     */
    public HLAoctetPairLE(char value)
    {
        this();
        setValue(value);
    }
/**
 * Constructs a <code>HLAoctetPairLE</code> of the specified short value.
 * The short's bytes are stored in unmodified sequence (thus first -> low; second -> high).
 * @param value a short specifying <code>this</code>' value
 */
public HLAoctetPairLE(short value)
{
    this ((char)value);
}

/**
 * Constructs a <code>HLAoctetPairLE</code> of the specified short value, using high and low bytes.
 * @param highByte a byte specifying the high byte of <code>this</code>' value
 * @param lowByte a byte specifying the low byte of <code>this</code>' value
 */
public HLAoctetPairLE(byte highByte, byte lowByte)
{
    //Note the byte order reversal when compared with HLAoctetPairBE
    this( (char)( ((int)lowByte) << 8 ) | highByte );
}

/**
 * Creates a <code>HLAoctetPairLE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAoctetPairLE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAoctetPairLE(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}
/**
 * Creates a <code>HLAoctetPairLE</code> from the network representation in the provided <code>byte[]</code> at the indicated offset.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAoctetPairLE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAoctetPairLE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAoctetPairLE(byte[] buffer, int offset)
 throws CouldNotDecode
 { 
 this(new ByteWrapper(buffer, offset));
 }

 /**
 * Creates a <code>HLAoctetPairLE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAoctetPairLE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
 public HLAoctetPairLE(ByteWrapper byteWrapper)
 throws CouldNotDecode
 { 
 this();
 decode(byteWrapper);
 }

 //Class-specific extensions

 /**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
 public boolean isBigEndian()
 { 
 return false;
 }

 } //end HLAoctetPairLE
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract type-safe 16-bit integer basic data type.
 * It uses the <code>short</code> Java type to store the value
 * and is the ancestor of the HLAinteger16BE and HLAinteger16LE
 * concrete classes.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} (http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier)
 * @version 1.1
 */
public abstract class HLAinteger16
    extends HLAbasicType
{
    /** The 16-bit integer value: a Java short. */
    protected short _value;

    /** Length (in bytes) of the <code>byte[]</code> representation of
    this class. */
    public final static int encodedLength = 2;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 2;

    //java.lang.Object methods

    /**
     * Returns a <code>String</code> representation of
     * <code>this</code>.
     * @return a {link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String toString()
    {
        return Short.toString(getValue());
    }
}
/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
 * represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code>
 * is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (this == otherObject) return true;
    if (!this.getClass().equals(otherObject.getClass())) return false;
    // if (this.isBigEndian() != ((HLAinteger16)otherObject).isBigEndian()) return false;
    return (getValue() == ((HLAinteger16) otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which
 * <code>equals()</code> is <code>true</code> should yield the same hash
 * code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 * @see java.util.Hashtable Hashtable
 */
public int hashCode()
{
    return new Short(getValue()).hashCode();
}

//HLAdatatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]{}</code>
 * representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]{}</code>
 * representation of <code>this</code>
 */
public final int encodedLength()
{
    return encodedLength;
}
/**
 * Returns the octet boundary of <code>this</code>.
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to
 * encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return (isBigEndian() ? byteWrapper.putShort(getValue()) : byteWrapper.putShortLE(getValue()));
}

/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        if (isBigEndian())
        {
            setValue(byteWrapper.getShort());
        }
        else
        {
            setValue(byteWrapper.getShortLE());
        }
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
// class-specific extensions

/**
 * Returns the short value of <code>this</code>.  
 * @return the short value of <code>this</code>
 * @see #setValue
 */
public short getValue()
{
    return _value;
}

/**
 * Sets the short value of <code>this</code>.  
 * @param value the short new value for <code>this</code>  
 * @see #getValue
 */
public void setValue(short value)
{
    _value = value;
}

/**
 * Returns true if the class is big-endian.  
 * @return a boolean which is true if the class is big-endian
 */
public abstract boolean isBigEndian();

// end HLAinteger16
// File: HLAinteger16BE.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 16-bit big-endian integer basic data type.
 * It uses the <code>short</code> Java type to store the value.
 * <p>
 * <code>HLAinteger16BE</code>s are normally never obtained directly,
 * as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual
 * <code>HLAinteger16BEfactory</code>'s decode method on a
 * <code>byte[]</code> received as part of an attribute update or interaction. However, no
 * such factory is currently supplied by the RTIambassador.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault}
 * @version 1.1
 */
public class HLAinteger16BE
    extends HLAinteger16
{
    /**
     * Constructs a <code>HLAinteger16BB</code> of default value
     * (zero).
     */
    public HLAinteger16BE()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /**
     * Constructs a <code>HLAinteger16BE</code> of the specified value.
     * @param value a short specifying <code>this</code>'s value
     */
    public HLAinteger16BE(short value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAinteger16BE</code> from the network representation in the provided <code>byte[]</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger16BE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger16BE(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAinteger16BE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger16BE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAinteger16BE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger16BE(byte[] buffer, int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAinteger16BE</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAinteger16BE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAinteger16BE(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
//Class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean
isBigEndian()
{
    return true;
}

//end HLAinteger16BE
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 16-bit little-endian integer basic data type.
 * It uses the <code>short</code> Java type to store the value.
 * <p>
 * <code>HLAinteger16LE</code>s are normally never obtained directly,
 * as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual
 * <code>HLAinteger16LEfactory</code>’s decode method on a
 * <code>byte[]</code>
 * received as part of an attribute update or interaction. However, no
 * such factory is currently supplied by the RTI ambassador.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */

public class HLAinteger16LE
    extends HLAinteger16
{
    /**
     * Constructs a <code>HLAinteger16LE</code> of default value
     * (zero).
     */
    public HLAinteger16LE()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /**
     * Constructs a <code>HLAinteger16LE</code> of the specified value.
     * @param value a short specifying <code>this</code>’ value
     */
    public HLAinteger16LE(short value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAinteger16LE</code> from the network representation in the provided <code>byte[]</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger16LE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger16LE(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAinteger16LE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger16LE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAinteger16LE</code> representation begins 
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger16LE(byte[] buffer,
                      int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAinteger16LE</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAinteger16LE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAinteger16LE(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian()
{
    return false;
}
//end HLAinteger16LE
// File: HLAinteger32.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract type-safe 32-bit integer basic data type.
 * It uses the <code>int</code> Java type to store the value (not the byte sequence).
 * and is the ancestor of the HLAinteger32BE and HLAinteger32LE concrete classes.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}
 * @version 1.1
 */
public abstract class HLAinteger32 extends HLAbasicType
{
    /** The 32-bit integer value: a Java int. */
    protected int _value;

    /** Length (in bytes) of the <code>byte[]</code> representation of this class. */
    public final static int encodedLength = 4;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 4;

    //java.lang.Object methods

    /**
     * Returns a <code>String</code> representation of <code>this</code>.
     * @return a {java.lang.String} reflecting <code>this</code>
     */
    public String toString()
    {
        return Integer.toString(getValue());
    }
}
/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
 * represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code>
 * is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals (Object otherObject) {
    if (this == otherObject) return true;
    if (!this.getClass().equals(otherObject.getClass())) return false;
    // if (this.isBigEndian() != ((HLAInteger32)otherObject).isBigEndian()) return false;
    return (getValue() == ((HLAInteger32)otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which
 * <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 */
public int hashCode () {
    return new Integer(getValue()).hashCode();
}

//HLA datatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code>
 * representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code>
 * representation of <code>this</code>
 */
public final int encodedLength() {
    return encodedLength;
}
/**
 * Returns the octet boundary of <code>this</code>.  
 * The octet boundary value is defined as the smallest power of 2 
 * which is greater than or equal to the size of the datatype in bytes.  
 * For a constructed datatype, it is the maximum octet boundary 
 * value of all components within it.  
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> into which to 
 * encode <code>this</code>  
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return (isBigEndian() ? byteWrapper.putInt(getValue()) : byteWrapper.putIntLE(getValue()));
}

/**
 * Sets <code>this</code> value from the supplied  
 * <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current 
 * <code>pos()</code> the representation of the data type begins  
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not 
 * be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        if (isBigEndian())
        {
            setValue(byteWrapper.getInt());
        }
        else
        {
            setValue(byteWrapper.getIntLE());
        }
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//class-specific extensions

/**
 * Returns the int value of <code>this</code>.
 * @return the int value of <code>this</code>
 * @see #setValue
 */
public int getValue()
{
    return _value;
}

/**
 * Sets the int value of <code>this</code>.
 * @param value the int new value for <code>this</code>
 * @see #getValue
 */
public void setValue(int value)
{
    _value = value;
}

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public abstract boolean isBigEndian();
package ca.gc.drdd_rddc.hla.rti1516.omti;
import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit big-endian integer basic data type.
 * It uses the <code>int</code> Java type to store the value (not the byte sequence).
 * <p>
 * <code>HLAinteger32BE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual <code>HLAinteger32BEfactory</code>'s decode method on a <code>byte[]</code>
 * received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTIambassador.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public class HLAinteger32BE
    extends HLAinteger32
{
    /**
     * Constructs a <code>HLAinteger32BE</code> of default value (zero).
     */
    public HLAinteger32BE()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /**
     * Constructs a <code>HLAinteger32BE</code> of the specified value.
     * @param value an int specifying <code>this</code>' value
     */
    public HLAinteger32BE(int value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAinteger32BE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger32BE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAinteger32BE(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAinteger32BE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger32BE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAinteger32BE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAinteger32BE(byte[] buffer, int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAinteger32BE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAinteger32BE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
 public HLAinteger32BE(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian() {
    return true;
}
//end HLAInteger32BE
// File: HLAinteger32LE.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit little-endian integer basic data type.
 * It uses the <code>int</code> Java type to store the value (not the byte sequence).
 * <p>
 * <code>HLAinteger32LE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual <code>HLAinteger32LEfactory</code>’s decode method on a <code>byte[]</code>
 * received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTIambassador.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public class HLAinteger32LE
extends HLAinteger32
{
    /**
     * Constructs a <code>HLAinteger32LE</code> of default value (zero).
     */
    public HLAinteger32LE()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /**
     * Constructs a <code>HLAinteger32LE</code> of the specified value.
     * @param value an int specifying <code>this</code>’s value
     */
    public HLAinteger32LE(int value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAinteger32LE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger32LE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger32LE(byte[] buffer)
  throws CouldNotDecode
{
  this(buffer, 0);
}

/**
 * Creates a <code>HLAinteger32LE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAinteger32LE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAinteger32LE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger32LE(byte[] buffer, int offset)
  throws CouldNotDecode
{
  this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAinteger32LE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAinteger32LE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAinteger32LE(ByteWrapper byteWrapper)
  throws CouldNotDecode
{
  this();
  decode(byteWrapper);
}
/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian()
{
    return false;
}
//end HLAinteger32LE
// File: HLAinteger64.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract type-safe 64-bit integer basic data type.
 * It uses the <code>long</code> Java type to store the value (not the
 * byte sequence)
 * and is the ancestor of the HLAinteger64BE and HLAinteger64LE
 * concrete classes.
 *<p>
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} ( DRDC Valcartier)
 * @version 1.1
 */
public abstract class HLAinteger64
    extends HLAbasicType
{
    /** The 64-bit integer value: a Java long. */
    protected long _value;

    /** Length (in bytes) of the <code>byte[]</code> representation of
     * this class. */
    public final static int encodedLength = 8;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 8;

    //java.lang.Object methods

    /**
     * Returns a <code>String</code> representation of
     * <code>this</code>.
     * @return a {@link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String toString()
    { return Long.toString(getValue()); }
/**
 * Returns true iff <code>this</code> and <code>otherObject</code> represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return true iff supplied <code>otherObject</code> is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (this == otherObject) return true;
    if (!this.getClass().equals(otherObject.getClass())) return false;
    // if (this.isBigEndian() != ((HLAInteger64)otherObject).isBigEndian()) return false;
    return (getValue() == ((HLAInteger64)otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 */
public int hashCode()
{
    return new Long(getValue()).hashCode();
}

//HLAdatatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public final int encodedLength()
{
    return encodedLength;
}
/**
 * Returns the octet boundary of <code>this</code>. The octet boundary value is defined as
 * the smallest power of 2 which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary value of all components within it.
 * @return the octet boundary of <code>this</code>
 */
public final int octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return (isBigEndian() ? byteWrapper.putLong(getValue()) : byteWrapper.putLongLE(getValue()));
}

/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type
 * begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        if (isBigEndian())
        {
            setValue(byteWrapper.getLong());
        }
        else
        {
            setValue(byteWrapper.getLongLE());
        }
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//class-specific extensions

/**
 * Returns the long value of <code>this</code>. 
 * @return the long value of <code>this</code>
 * @see #setValue
 */
public long
getValue()
{
    return _value;
}

/**
 * Sets the long value of <code>this</code>. 
 * @param value the long new value for <code>this</code>
 * @see #getValue
 */
public void
setValue(long value)
{
    _value = value;
}

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public abstract boolean
isBigEndian();

//end HLAinteger64
Type-safe 64-bit big-endian integer basic data type. It uses the long Java type to store the value (not the byte sequence).

Typically, HLAinteger64BE's are normally never obtained directly, as the RTI uses simple data types instead.

They could also be obtained by using an eventual HLAinteger64BEFactory's decode method on a byte[] received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTI Ambassador.

* @author Daniel U. Thibault
  * @version 1.1

public class HLAinteger64BE extends HLAinteger64
{

    /*
     * Constructs a HLAinteger64BE of default value (zero).
     */
    public HLAinteger64BE()
    {
        //super() not called because super-class is abstract
        //Default _value is zero
    }

    /*
     * Constructs a HLAinteger64BE of the specified value.
     * @param value a long specifying this's value
     */
    public HLAinteger64BE(long value)
    {
        this();
        setValue(value);
    }
public HLAinteger64BE(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}

public HLAinteger64BE(byte[] buffer,
int offset)
throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

public HLAinteger64BE(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
// Class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian()
{
    return true;
}

// end HLAinteger64BE
package ca.gc.drdc_rddc.hla.rti516.omti;

import hla.rti516.CouldNotDecode;

/**
 * Type-safe 64-bit little-endian integer basic data type.
 * It uses the <code>long</code> Java type to store the value (not the byte sequence).
 * <p>
 * <code>HLAinteger64LE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual <code>HLAinteger64LEfactory</code>'s decode method on a <code>byte[]</code> received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTI ambassador.
 * @author [Daniel.U.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */

public class HLAinteger64LE
    extends HLAinteger64
{
    /**
     * Constructs a <code>HLAinteger64LE</code> of default value (zero).
     */
    public HLAinteger64LE()
    {
      //super() not called because super-class is abstract
      //Default _value is zero
    }

    /**
     * Constructs a <code>HLAinteger64LE</code> of the specified value.
     * @param value a long specifying <code>this</code>' value
     */
    public HLAinteger64LE(long value)
    {
      this();
      setValue(value);
    }
/**
 * Creates a <code>HLAinteger64LE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAinteger64LE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger64LE(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAinteger64LE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAinteger64LE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAinteger64LE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAinteger64LE(byte[] buffer,
int offset)
throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAinteger64LE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAinteger64LE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAinteger64LE(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
//class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian()
{
    return false;
}
//end HLAinteger64LE
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Abstract type-safe 32-bit IEEE 754-1985 floating point basic data type.
 * It uses the <code>float</code> Java type to store the value (not the byte sequence)
 * and is the ancestor of the HLAfloat32BE and HLAfloat32LE concrete classes.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.1
 */
public abstract class HLAfloat32
  extends HLAbasicType
{
  /** The stored 32-bit IEEE 754-1985 floating point value: a Java float. */
  protected float _value;

  /** Length (in bytes) of the <code>byte[]</code> representation of this class. */
  public final static int encodedLength = 4;

  /** Octet boundary of this class. */
  public final static int octetBoundary = 4;

  //java.lang.Object methods

  /**
   * Returns a <code>String</code> representation of <code>this</code>.
   * @return a {@link java.lang.String} reflecting <code>this</code>
   */
  public String toString()
  {
    return Float.toString(getValue());
  }
}
/**
   * Returns true iff <code>this</code> and <code>otherObject</code>
   * represent the same object.
   * @param otherObject the <code>Object</code> to compare with
   * @return <code>true</code> iff supplied <code>otherObject</code>
   * is of the same type as <code>this</code> and has the same value
   * @see Object#hashCode Object.hashCode()
   * @see java.util.Hashtable Hashtable
   */
   public boolean 
   equals(Object otherObject)
   {
     if (this == otherObject) return true;
     if (! this.getClass().equals(otherObject.getClass())) return false;
     // if (this.isBigEndian() != ((HLAfloat32)otherObject).isBigEndian()) return false;
     return (getValue() == ((HLAfloat32)otherObject).getValue());
   }

   /**
   * Returns a hash code for <code>this</code>; two objects for which
   * <code>equals() and true</code> should yield the same hash code.
   * @return an <code>int</code> hash code
   * @see Object#equals(java.lang.Object)
   * @see java.util.Hashtable Hashtable
   */
   public int 
   hashCode()
   {
     return new Float(getValue()).hashCode();
   }

   // HLA datatype interface implementation

   /**
   * Returns the length (in bytes) of the <code>byte[]</code>
   * representation of <code>this</code>.
   * @return the length (in bytes) of the <code>byte[]</code>
   * representation of <code>this</code>
   */
   public final int 
   encodedLength()
   {
     return encodedLength;
   }
/**
 * Returns the octet boundary of <code>this</code>.  
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.  
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.  
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> into which to
 * encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return (isBigEndian() ?
        byteWrapper.putInt(Float.floatToRawIntBits(getValue())) :
        byteWrapper.putIntLE(Float.floatToRawIntBits(getValue())) );
}

/**
 * Sets <code>this</code> value from the supplied
 * <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current
 * <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not
 * be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    try
    {
        byteWrapper.align(octetBoundary());
        if (isBigEndian())
        {
            setValue(Float.intBitsToFloat(byteWrapper.getInt()));
        }
        else
        {
            setValue(Float.intBitsToFloat(byteWrapper.getIntLE()));
        }
        return byteWrapper;
    }
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//class-specific extensions

/**
 * Returns the float value of <code>this</code>.  
 * @return the float value of <code>this</code>  
 * @see #setValue
 */
public float
getValue()
{
    return _value;
}

/**
 * Sets the float value of <code>this</code>.  
 * @param value the float new value for <code>this</code>  
 * @see #getValue
 */
public void
setValue(float value)
{
    _value = value;
}

/**
 * Returns true if the class is big-endian.  
 * @return a boolean which is true if the class is big-endian  
 */
public abstract boolean
isBigEndian();

//end HLAfloat32
// File: HLAfloat32BE.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit big-endian IEEE 754-1985 floating point basic data type. 
 * It uses the <code>float</code> Java type to store the value (not the byte sequence).
 * <p>
 * <code>HLAfloat32BE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual <code>HLAfloat32BEfactory</code>’s decode method on a <code>byte[]</code> received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTI ambassador.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public class HLAfloat32BE
    extends HLAfloat32
{
    /**
     * Constructs a <code>HLAfloat32BE</code> of default value (NaN).
     */
    public HLAfloat32BE()
    {
        //super() not called because super-class is abstract
        setValue(Float.NaN); //Java’s default is "positive zero"
    }

    /**
     * Constructs a <code>HLAfloat32BE</code> of the specified value.
     * @param value a float specifying <code>this</code>’s value
     */
    public HLAfloat32BE(float value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAfloat32BE</code> from the network representation in the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat32BE</code>  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
 */
public HLAfloat32BE(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAfloat32BE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat32BE</code>  
 * @param offset where in the <code>buffer</code> the <code>HLAfloat32BE</code> representation begins  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
 */
public HLAfloat32BE(byte[] buffer, int offset)
throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAfloat32BE</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfloat32BE</code> begins  
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
 */
public HLAfloat32BE(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
// Class-specific extensions

/**
   * Returns true if the class is big-endian.
   * @return a boolean which is true if the class is big-endian
   */
public boolean isBigEndian()
{
   return true;
}

// end HLAfloat32BE
// File: HLAfloat32LE.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
  * Type-safe 32-bit little-endian IEEE 754-1985 floating point basic
data type.
  * It uses the <code>float</code> Java type to store the value (not
  * the byte sequence).
  * <p>
  * <code>HLAfloat32LE</code>s are normally never obtained directly, as
  * the RTI uses simple data types instead.
  * <p>
  * They could also be obtained by using an eventual
  * <code>HLAfloat32LEfactory</code>'s decode method on a
  * <code>byte[]</code>
  * received as part of an attribute update or interaction. However, no
  * such factory is currently supplied by the RTI ambassador.
  * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
  * Thibault] [http://www.valcartier.drdc-rddc.gc.ca DRDC
  * Valcartier]
  * @version 1.1
  */
public class
HLAfloat32LE
extends HLAfloat32
{
/**
  * Constructs a <code>HLAfloat32LE</code> of default value (NaN).
  */
public
HLAfloat32LE()
{
  //super() not called because super-class is abstract
  setValue(Float.NaN); //Java's default is "positive zero"
}

/**
  * Constructs a <code>HLAfloat32LE</code> of the specified value.
  * @param value a float specifying <code>this</code>' value
  */
public
HLAfloat32LE(float value)
{
  this();
  setValue(value);
}
/**
 * Creates a `<code>HLAfloat32LE</code>` from the network representation in the provided `<code>byte[]</code>`.
 * @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAfloat32LE</code>`
 * @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
 */
public HLAfloat32LE(byte[] buffer)
  throws CouldNotDecode
{
  this(buffer, 0);
}

/**
 * Creates a `<code>HLAfloat32LE</code>` from the network representation in the provided `<code>byte[]</code>` at the indicated `<code>offset</code>`.
 * @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAfloat32LE</code>`
 * @param offset where in the `<code>buffer</code>` the `<code>HLAfloat32LE</code>` representation begins
 * @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
 */
public HLAfloat32LE(byte[] buffer, int offset)
  throws CouldNotDecode
{
  this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a `<code>HLAfloat32LE</code>` from the supplied `<code>ByteWrapper</code>`.
 * @param byteWrapper the `<code>ByteWrapper</code>` at whose current `<code>pos()</code>` the representation of the `<code>HLAfloat32LE</code>` begins
 * @throws CouldNotDecode if the `<code>ByteWrapper</code>` could not be decoded
 */
public HLAfloat32LE(ByteWrapper byteWrapper)
  throws CouldNotDecode
{
  this();
  decode(byteWrapper);
}
//class-specific extensions

/**
   * Returns true if the class is big-endian.
   * @return a boolean which is true if the class is big-endian
   */
public boolean isBigEndian()
{
    return false;
}

//end HLAfloat32LE
/**
 * Abstract type-safe 64-bit IEEE 754-1985 floating point basic data type.
 * It uses the <code>double</code> Java type to store the value (not the byte sequence)
 * and is the ancestor of the HLAfloat64BE and HLAfloat64LE concrete classes.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public abstract class HLAfloat64
    extends HLAbasicType
{
    /** The 64-bit IEEE 754-1985 floating point value: a Java double. */
    protected double _value;

    /** Length (in bytes) of the <code>byte[]</code> representation of this class. */
    public final static int encodedLength = 8;

    /** Octet boundary of this class. */
    public final static int octetBoundary = 8;

    //java.lang.Object methods

    /**
     * Returns a <code>String</code> representation of <code>this</code>.
     * @return a {link java.lang.String} reflecting <code>this</code>
     */
    public String toString()
    {
        return Double.toString(getValue());
    }
/**
 * Returns true iff <code>this</code> and <code>otherObject</code> represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code> is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals (Object otherObject)
{
    if (this == otherObject) return true;
    if (!this.getClass().equals(otherObject.getClass())) return false;
    // if (this.isBigEndian() != ((HLAfloat64)otherObject).isBigEndian()) return false;
    return (getValue() == ((HLAfloat64)otherObject).getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 */
public int hashCode()
{
    return new Double(getValue()).hashCode();
}

//HLAdatatype interface implementation
/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public final int encodedLength()
{
    return encodedLength;
}
/**
 * Returns the octet boundary of <code>this</code>. 
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    return octetBoundary;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> into which to
 * encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    return ( isBigEndian() ?
    byteWrapper.putLong(Double.doubleToRawLongBits(getValue())) :
    byteWrapper.putLongLE(Double.doubleToRawLongBits(getValue())) );
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper decode(ByteWrapper byteWrapper) throws CouldNotDecode {
    try {
        byteWrapper.align(octetBoundary());
        if (isBigEndian()) {
            setValue(Double.longBitsToDouble(byteWrapper.getLong()));
        } else {
            setValue(Double.longBitsToDouble(byteWrapper.getLongLE()));
        }
        return byteWrapper;
    }
    catch (Exception e) {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//Class-specific extensions

/**
 * Returns the double value of <code>this</code>.
 * @return the double value of <code>this</code>
 * @see #setValue
 */
public double getValue() {
    return _value;
}

/**
 * Sets the double value of <code>this</code>.
 * @param value the double new value for <code>this</code>
 * @see #getValue
 */
public void setValue(double value) {
    _value = value;
}
/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public abstract boolean isBigEndian();
}
//end HLAfloat64
// File: HLAfloat64BE.java
package ca.gc.drdc_rddc.hla.rti1516.omti;
import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 64-bit big-endian IEEE 754-1985 floating point basic data type.
 * It uses the <code>double</code> Java type to store the value (not the byte sequence).
 * <p>
 * <code>HLAfloat64BE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p>
 * They could also be obtained by using an eventual <code>HLAfloat64BEfactory</code>'s decode method on a <code>byte[]</code>
 * received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTI ambassador.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public class HLAfloat64BE
    extends HLAfloat64
{
    /**
     * Constructs a <code>HLAfloat64BE</code> of default value (NaN).
     */
    public HLAfloat64BE()
    {
        //super() not called because super-class is abstract
        setValue(Double.NaN); //Java’s default value is "positive zero"
    }

    /**
     * Constructs a <code>HLAfloat64BE</code> of the specified value.
     * @param value a double specifying <code>this</code>'s value
     */
    public HLAfloat64BE(double value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAfloat64BE</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat64BE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAfloat64BE(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAfloat64BE</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat64BE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAfloat64BE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAfloat64BE(byte[] buffer,
int offset)
throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAfloat64BE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfloat64BE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public
HLAfloat64BE(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
// class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian() {
    return true;
}

// end HLAfloat64BE
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 64-bit little-endian IEEE 754-1985 floating point basic data type.
 * It uses the <code>double</code> Java type to store the value (not the byte sequence).
 * <p *
 * <code>HLAfloat64LE</code>s are normally never obtained directly, as the RTI uses simple data types instead.
 * <p *
 * They could also be obtained by using an eventual <code>HLAfloat64LEfactory</code>'s decode method on a <code>byte[]</code>
 * received as part of an attribute update or interaction. However, no such factory is currently supplied by the RTIambassador.
 * @author {mailto:Daniels.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */

class HLAfloat64LE
    extends HLAfloat64
{

    /**
     * Constructs a <code>HLAfloat64LE</code> of default value (NaN).
     */
    public HLAfloat64LE()
    {
        //super() not called because super-class is abstract
        setValue(Double.NaN); //Java's default value is "positive zero"
    }

    /**
     * Constructs a <code>HLAfloat64LE</code> of the specified value.
     * @param value a double specifying <code>this</code>' value
     */
    public HLAfloat64LE(double value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAfloat64LE</code> from the network representation in the provided <code>byte[]</code>
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat64LE</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAfloat64LE(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAfloat64LE</code> from the network representation in the provided <code>byte[]</code> at the indicated offset.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfloat64LE</code>
 * @param offset where in the <code>buffer</code> the <code>HLAfloat64LE</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAfloat64LE(byte[] buffer, int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAfloat64LE</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfloat64LE</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAfloat64LE(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
//class-specific extensions

/**
 * Returns true if the class is big-endian.
 * @return a boolean which is true if the class is big-endian
 */
public boolean isBigEndian()
{
    return false;
}

//end HLAfloat64LE
The next three classes implement the HLA simple datatypes. In most cases the underlying basic datatype is simply extended and the constructors redeclared, but the HLAASCIIChar and HLAUnicodeChar classes need a little extra because of the way Java treats characters.

// File: HLAbyte.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 8-bit byte simple data type.
 * It uses the <code>HLAoctet</code> basic data type.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public class HLAbyte
extends HLAoctet
{
  /**
   * Constructs a <code>HLAbyte</code> of default value (zero).
   *
   */
  public HLAbyte()
  {
    super();
  }

  /**
   * Constructs a <code>HLAbyte</code> of the specified value.
   * @param value a byte specifying <code>this</code>' value
   *
   */
  public HLAbyte(byte value)
  {
    super(value);
  }

  /**
   * Creates a <code>HLAbyte</code> from the network representation
   * in the provided <code>byte[]</code>
   * @param buffer the network-provided <code>byte[]</code>
   * representation of the <code>HLAbyte</code>
   * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
   *
   */
  public HLAbyte(byte[] buffer)
  throws CouldNotDecode
  {
    super(buffer);
  }
}
/**
 * Creates a <code>HLAbyte</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAbyte</code>
 * @param offset where in the <code>buffer</code> the <code>HLAbyte</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public HLAbyte(byte[] buffer,
        int offset) throws CouldNotDecode
 {
     super(buffer, offset);
 }

/**
 * Creates a <code>HLAbyte</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAbyte</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public HLAbyte(ByteWrapper byteWrapper)
 throws CouldNotDecode
 {
     super(byteWrapper);
 }

//end HLAbyte
```java
// File: HLAASCIIchar.java
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 8-bit ASCII character simple data type.
 * It uses the <code>HLAoctet</code> basic data type.
 * The first 128 characters of the Unicode character encoding are the
 * ASCII characters (cf. ANSI/INCITS 4-1986(R1997)).
 * The Java <code>char</code> primitive type is a 16-bit unsigned
 * integer representing Unicode characters.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */
public class HLAASCIIchar
    extends HLAoctet
{
    /**
     * Constructs a <code>HLAASCIIchar</code> of default value (Unicode
     * Null).
     */
    public HLAASCIIchar()
    {
        super();
    }

    /**
     * Constructs a <code>HLAASCIIchar</code> of the specified value.
     * An exception occurs if the specified char is not ASCII.
     * @param value a char specifying <code>this</code>' value
     * @throws IllegalArgumentException if the value isn’t ASCII
     */
    public HLAASCIIchar(char value)
        throws IllegalArgumentException
    {
        super(verify(value));
    }
}
```
/**
 * Constructs a <code>HLAASCIIchar</code> from the specified String.
 * @param s a String whose charAt(0) specifies <code>this</code>' value
 * @throws IllegalArgumentException if the String is empty or its first character isn't ASCII
 */
public HLAASCIIchar(String s)
    throws IllegalArgumentException
{
    super(verify(s)); //This includes the zero-length string exception
}

/**
 * Constructs a <code>HLAASCIIchar</code> from the specified Object.
 * @param o an Object whose toString().charAt(0) specifies <code>this</code>' value
 * @throws IllegalArgumentException if Object.toString() is empty or its first character isn't ASCII
 */
public HLAASCIIchar(Object o)
    throws IllegalArgumentException
{
    this(o.toString());
}

/**
 * Creates a <code>HLAASCIIchar</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAASCIIchar</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAASCIIchar(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}
/**
 * Creates a <code>HLAASCIIchar</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAASCIIchar</code>  
 * @param offset where in the <code>buffer</code> the <code>HLAASCIIchar</code> representation begins  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
 */
public
HLAASCIIchar(byte[] buffer,
            int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAASCIIchar</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAASCIIchar</code> begins  
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
 */
public
HLAASCIIchar(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
/**
 * Verifies if the char value is ASCII and returns its byte value.
 * @param value the char to verify
 * @return the byte representation of the ASCII character
 * @throws IllegalArgumentException if value is not in the 00..7F range
 */
protected final static byte verify(char value)
throws IllegalArgumentException
{
    if (value > 0x0FFF)
        throw new IllegalArgumentException("0x" +
            Integer.toHexString((int)value));
    else if (value > 0x00FF)
        throw new IllegalArgumentException("0x0" +
            Integer.toHexString((int)value));
    else if (value > 0x007F)
        throw new IllegalArgumentException("0x00" +
            Integer.toHexString((int)value));
    else
    {
        //Verification succeeded
        return (byte)value;
    }
}

/**
 * Verifies if the specified String is ASCII and returns the byte value of its first character.
 * @param s the String to verify
 * @return the byte representation of the String's first character (ASCII)
 * @throws IllegalArgumentException if value is not in the 00..7F range
 */
protected final static byte verify(String s)
throws IllegalArgumentException
{
    if (s.length() <= 0) throw new IllegalArgumentException();
    return verify(s.charAt(0));
}
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {@link java.lang.String} reflecting <code>this</code> value
 */
public String toString()
{
    return Character.toString(getChar());
}

//Class-specific extensions

/**
 * Returns the char value of <code>this</code>. 
 * @return the char value of <code>this</code> 
 */
public char getChar()
{
    return (char)getValue();
}

/**
 * Returns <code>this</code> value as a Character.
 * @return the Character representation of <code>this</code> value
 */
protected Character getCharacter()
{
    return new Character(getChar());
}

//Override to force verify
/**
 * Sets the byte value of <code>this</code>. 
 * @param value the byte new value for <code>this</code>
 * @throws IllegalArgumentException if the value isn't ASCII 
 */
public void setValue(byte value)
{
    super.setValue(verify((char)value));
}
// overload
/**
 * Sets the char value of <code>this</code>.
 * @param value the char new value for <code>this</code>
 * @throws IllegalArgumentException if the value isn’t ASCII
 */
public void setValue(char value)
{
    // We could also write `setValue(verify(value));` but this would
    // end up calling verify twice
    super.setValue(verify(value));
}

// end HLAASCIIchar
// File: HLAunicodeChar.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 16-bit Unicode character simple data type.
 * It uses the <code>HLAoctetPairBE</code> basic data type.
 * The Java <code>char</code> primitive type is a 16-bit unsigned
 * integer representing Unicode characters.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */
public class HLAunicodeChar
    extends HLAoctetPairBE
{
    /**
     * Constructs a <code>HLAunicodeChar</code> of default value
     * (Unicode Null).
     */
    public HLAunicodeChar()
    {
        super();
    }

    /**
     * Constructs a <code>HLAunicodeChar</code> of the specified value.
     * @param value a char specifying <code>this</code>' value
     */
    public HLAunicodeChar(char value)
    {
        super(value);
    }

    /**
     * Constructs a <code>HLAunicodeChar</code> from the specified
     * String.
     * @param s a String whose charAt(0) specifies <code>this</code>
     * @throws IllegalArgumentException if s is empty.
     */
    public HLAunicodeChar(String s)
        throws IllegalArgumentException
    {
        this();
        if (s.length() == 0) throw new IllegalArgumentException();
        setValue(s.charAt(0));
    }
public HLAunicodeChar(Object o) {
    this(o.toString());
}

public HLAunicodeChar(byte[] buffer) throws CouldNotDecode {
    this(buffer, 0);
}

public HLAunicodeChar(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a `<code>HLAunicodeChar</code>` from the supplied
 * `<code>ByteWrapper</code>`.  
 * @param byteWrapper the `<code>ByteWrapper</code>` at whose current
 * `<code>pos()</code>` the representation of the
 * `<code>HLAunicodeChar</code>` begins
 * @throws CouldNotDecode if the `<code>ByteWrapper</code>` could not
 * be decoded
 */
 public HLAunicodeChar(ByteWrapper byteWrapper)
 throws CouldNotDecode
 {
   this();
   decode(byteWrapper);
 }

 //java.lang.Object methods

 /**
 * Returns a `<code>String</code>` representation of
 * `<code>this</code>`.  
 * @return a `{@link java.lang.String}` reflecting `<code>this</code>
 * value
 */
 public String toString()
 {
   return Character.toString(getValue());
 }

 //Class-specific extensions

 // getValue already returns a char.
 /**
 * Returns `<code>this</code>` value as a Character.  
 * @return the Character representation of `<code>this</code>` value
 */
 protected Character getCharacter()
 {
   return new Character(getValue());
 }

 //Overload
 /**
 * Sets the short value of `<code>this</code>`.  
 * @param value the short new value for `<code>this</code>`
 */
 public void setValue(short value)
 {
   setValue((char)value);
 }

 //end HLAunicodeChar
The next four classes implement the HLA MOM simple datatypes.

// File: HLAcount.java
package ca.gc.drdc_rddc.hla.rti1516.omt;

import hla.rti1516.CouldNotDecode;

/**
* Type-safe 32-bit big-endian integer basic data type used by the
* Management Object Model (MOM).
* It is an HLAinteger32BE.
* @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
* @version 1.1
*/
public class HLAcount
extends HLAinteger32BE
{
    /**
     * Constructs an <code>HLAcount</code> of default value (zero).
     */
    public HLAcount()
    {
        super(); //Default _value is zero
    }

    /**
     * Constructs a <code>HLAcount</code> of the specified value.
     * @param value an int specifying <code>this</code>' value
     */
    public HLAcount(int value)
    {
        this();
        setValue(value);
    }

    /**
     * Creates a <code>HLAcount</code> from the network representation
     * in the provided <code>byte[]</code>.
     * @param buffer the network-provided <code>byte[]</code>
     * representation of the <code>HLAcount</code>
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
     */
    public HLAcount(byte[] buffer)
    throws CouldNotDecode
    {
        this(buffer, 0);
    }
}
/**
 * Creates a <code>HLAcount</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAcount</code>
 * @param offset where in the <code>buffer</code> the <code>HLAcount</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public
 HLAcount(byte[] buffer,
           int offset)
   throws CouldNotDecode
 {
   this(new ByteWrapper(buffer, offset));
 }

/**
 * Creates a <code>HLAcount</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAcount</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
 public
 HLAcount(ByteWrapper byteWrapper)
   throws CouldNotDecode
 {
   this();
   decode(byteWrapper);
 }

//end HLAcount
// File: HLAfederateHandle.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit big-endian integer basic data type used by the Management Object Model (MOM).
 * It is the type of the argument to the normalizeFederateHandle service, that is to say, FederateHandle.
 * Stored as an HLAInteger32BE, it is however a pointer to an RTI defined programming language object, NOT a true HLAInteger32BE.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.1
 */
public class HLAfederateHandle extends HLAinteger32BE
{
    /**
     * Constructs an <code>HLAfederateHandle</code> of default value (zero).
     */
    public HLAfederateHandle()
    {
        super(); //Default _value is zero
    }

    /**
     * Constructs a <code>HLAfederateHandle</code> of the specified value.
     * @param value an int specifying <code>this</code>' value
     */
    public HLAfederateHandle(int value)
    {
        this();
        setValue(value);
    }
/**
 * Creates a <code>HLAfederateHandle</code> from the network representation in the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfederateHandle</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public 
 HLAfederateHandle(byte[] buffer)
     throws CouldNotDecode
 {
     this(buffer, 0);
 }

/**
 * Creates a <code>HLAfederateHandle</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAfederateHandle</code>
 * @param offset where in the <code>buffer</code> the <code>HLAfederateHandle</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public 
 HLAfederateHandle(byte[] buffer, 
     int offset)
     throws CouldNotDecode
 {
     this(new ByteWrapper(buffer, offset));
 }

/**
 * Creates a <code>HLAfederateHandle</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfederateHandle</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
 public 
 HLAfederateHandle(ByteWrapper byteWrapper)
     throws CouldNotDecode
 {
     this();
     decode(byteWrapper);
 }
//end HLAfederateHandle
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit big-endian integer basic data type used by the
 * Management Object Model (MOM).
 * It is an HLAinteger32BE.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault}({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.1
 */
public class HLAmsec
    extends HLAinteger32BE
{
    /**
     * Constructs an &lt;code&gt;HLAmsec&lt;/code&gt; of default value (zero).
     */
    public HLAmsec()
    {
        super(); //Default _value is zero
    }

    /**
     * Constructs a &lt;code&gt;HLAmsec&lt;/code&gt; of the specified value.
     * @param value an int specifying &lt;code&gt;this&lt;/code&gt;' value
     */
    public HLAmsec(int value)
    {
        this();
        setValue(value);
    }

    /**
     * Creates a &lt;code&gt;HLAmsec&lt;/code&gt; from the network representation
     * in the provided &lt;code&gt;byte[]&lt;/code&gt;.
     * @param buffer the network-provided &lt;code&gt;byte[]&lt;/code&gt;
     * representation of the &lt;code&gt;HLAmsec&lt;/code&gt;
     * @throws CouldNotDecode if the &lt;code&gt;buffer&lt;/code&gt; could not be decoded
     */
    public HLAmsec(byte[] buffer)
    throws CouldNotDecode
    {
        this(buffer, 0);
    }
}
/**
 * Creates a <code>HLAmsec</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAmsec</code>  
 * @param offset where in the <code>buffer</code> the <code>HLAmsec</code> representation begins  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
 */
public HLAmsec(byte[] buffer, int offset) throws CouldNotDecode  
    this (new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAmsec</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAmsec</code> begins  
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
 */
public HLAmsec(ByteWrapper byteWrapper) throws CouldNotDecode  
    this();  
    decode(byteWrapper);
}
// File: HLAseconds.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe 32-bit big-endian integer basic data type used by the Management Object Model (MOM).
 * It is an HLAinteger32BE.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public class HLAseconds extends HLAinteger32BE {
    /**
     * Constructs an <code>HLAseconds</code> of default value (zero).
     */
    public HLAseconds()
    {
        super(); //Default _value is zero
    }

    /**
     * Constructs a <code>HLAseconds</code> of the specified value.
     * @param value an int specifying <code>this</code>' value
     */
    public HLAseconds(int value)
    {
        this();
        setValue(value);
    }

    /**
     * Creates a <code>HLAseconds</code> from the network representation in the provided <code>byte[]</code>.
     * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAseconds</code>
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
     */
    public HLAseconds(byte[] buffer)
    throws CouldNotDecode
    {
        this(buffer, 0);
    }
}
/**
 * Creates a <code>HLAseconds</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAseconds</code>
 * @param offset where in the <code>buffer</code> the <code>HLAseconds</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public HLAseconds(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAseconds</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAseconds</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public HLAseconds(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

//end HLAseconds
In constructing enumerated datatypes, it was assumed the pattern established by such classes as ResignAction would be followed.

// File: HLAEnumeratedDatatype.java
package ca.gc.drdc_rddc.hla.rti1516.omt;

import java.util.Iterator;

/**
 * Interface implemented by the HLA enumerated data types.
 * <p>
 * An enumerated data type extends a basic data type (like a simple data type does) and
 * specifies an immutable finite set of prototype values (not necessarily contiguous nor ordered).
 * <p>
 * Design guidelines:
 * <ul>
 * <li>A private constructor should be used to generate the static values (the enumerators).
 * <li>The java.lang.Object methods <code>toString()</code>, <code>equals()</code> and <code>hashCode()</code> may or may not need to be overridden.
 * <li>Although the inherited <code>getValue()</code> does not need to be overridden, this may be useful if only to change the Javadoc.
 * <li>The inherited <code>setValue</code>, on the other hand, will definitely need to be overridden so that it throws an exception when a value other than an acceptable one is passed in, or an <code>UnsupportedOperationException</code> when applied to one of the static values.
 * <li>Specialised <code>get/set</code> methods can be supplied if needed.
 * <li>A complete set of prototype immutable (public static final) instances must be generated by the class initializer.
 * <li>There must be a <code>public static Iterator iterator()</code> method that returns an Iterator over the enumeration's elements (prototype immutable instances).
 * This Iterator must throw an <code>UnsupportedOperationException</code> in response to its <code>remove()</code> method.
 * <li>There must also be a <code>public static Iterator nameIterator()</code> method which returns an Iterator over the names of the enumeration's elements, in the same sequence as <code>iterator()</code>.
 * </ul>
 * In order to facilitate the implementation, the class HLAEnumeratedIterator is supplied;
 * it defines constructors which should fulfill the contracts without modification.
 * See the comments below and the HLAboolean class for examples.
 * @author {mailto:Daniell.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public interface HLAenumerateddatatype extends HLADatatype
{
    /**
     * Returns an <code>Iterator</code> over the enumeration's elements.
     * Note that the <code>Iterator</code> returned by this method should throw an
     * <code>UnsupportedOperationException</code> in response to its
     * <code>remove()</code> method.
     * @return an Iterator over the elements in this enumeration
     * @throws ClassCastException if the Iterator construction fails
     */
    public Iterator iterator() throws ClassCastException;

    // { //You can use the self-defining Iterator:
    // return new HLAenumeratedIterator(HLAbollean.class, false);
    // } //Or you can specify the enumerators manually:
    // return new HLAenumeratedIterator(new Object[] { HLAbollean.HLAbfalse, HLAbollean.HLAttrue }, false);
    // }

    /**
     * Returns an <code>Iterator</code> over the enumeration's element names.
     * Note that the <code>Iterator</code> returned by this method throws an
     * <code>UnsupportedOperationException</code> in response to its
     * <code>remove()</code> method.
     * @return an Iterator over the names of the elements in this enumeration
     * @throws ClassCastException if the Iterator construction fails
     */
    public Iterator nameIterator() throws ClassCastException;

    // { //You can use the self-defining Iterator:
    // return new HLAenumeratedIterator(HLAbollean.class, true);
    // } //Or you can specify the enumerators manually:
    // return new HLAenumeratedIterator(new Object[] { HLAbollean.HLAbfalse, HLAbollean.HLAttrue }, true);
    // }

>Returns the underlying Class of the enumeration's elements (the elements' representation Class).

@returns the underlying Class of the enumeration's elements

Class
getElementClass();

///
//   return this.getClass().getSuperclass();
// }

/**
 * Returns true if the instance is immutable.
 * This is useful to distinguish between the class-supplied prototype instances and
 * the user-created ones (it is not possible to segregate them completely, since a user-defined
 * reference may end up pointing to a class-supplied immutable instance).
 * @returns a boolean which is true if the instance is immutable
 */
boolean
isImmutable();

//end HLAenumerateddatatype
// File: HLAenumeratedIterator.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.ArrayList;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

/**
 * Iterator implementation for HLA enumerated types.
 * Since the backing object (an HLA enumerated data type class) is
 * supposed to be immutable,
 * the Iterator is considerably simplified.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.2
 */
public class HLAenumeratedIterator
    implements Iterator {
    /**
     * The enumerators over which to iterate.
     */
    private Object enumerators[];

    /**
     * Index of enumerator to be returned by subsequent call to next().
     */
    private int cursor;

    /**
     * Constructs the <code>Iterator</code> from a <code>Class</code>
     * reference.
     * @param theClass the Class whose public static final instances
     * are to be enumerated
     * @param asStrings a boolean which is false if the enumeration
     * should supply Objects, true if it should supply the field names (as
     * Strings) instead
     * @throws ClassCastException if theInstance doesn't implement the
     * HLAenumerateddatatype interface (or an IllegalAccessException
     * occurred internally)
     */

public
HLAEnumeratedIterator(Class theClass, boolean asStrings)
throws ClassCastException
{
    if (!HLAEnumeratedDatatype.class.isAssignableFrom(theClass))
        throw new ClassCastException(theClass.toString());
    int mods;
    cursor = 0;
    ArrayList al = new ArrayList();
    // Obtain the class' fields
    Field[] theFields = theClass.getFields();
    for (int i = 0; i < theFields.length; i++)
    {
        // We're looking for fields which are instances of the class itself
        if (theFields[i].getType().equals(theClass))
            if (Modifier.isFinal(mods) && Modifier.isPublic(mods) &&
                Modifier.isStatic(mods))
                if (asStrings)
                    al.add(theFields[i].getName());
                else
                    try
                        { al.add(theFields[i].get(null));
                        catch (IllegalAccessException e)
                            { ClassCastException cce = new
ClassCastException(e.getMessage());
                            throw (ClassCastException)cce.initCause(e);
                        }
                    }
        al.trimToSize();
        enumerators = al.toArray();
    }
/**
 * Constructs the enumeration using the specified enumerators.
 * This can be used to change the enumeration order, repeat or omit enumerators.
 * If the <code>boolean asStrings</code> is false, the <code>Iterator</code> will enumerate the <code>Object</code>s;
 * otherwise it will enumerate the prototype field names as <code>String</code>s.
 * <p>
 * A prototype field is a field of a class or instance whose class is the same as the class or instance's class.
 * For example, the prototype fields of the <code>HLAboolean</code> class are <code>HLAfalse</code> and <code>HLAtrue</code>.
 * The names of <code>theEnumerators</code> that are not prototype fields of their classes will appear <code>null</code>.
 * Note that the constructor expects <code>theEnumerators</code> to all be <code>Object</code>s of the same <code>Class</code>,
 * which must implement the HLAenumerateddatatype interface.
 * However, it does not demand that the fields be <code>public static final</code>.
 * @param theEnumerators an Object[] specifying the enumerators and their order
 * @param asStrings a boolean which is false if the enumeration should supply theEnumerators, true if it should supply their field names (as Strings) instead
 * @throws ClassCastException if theInstance doesn't implement the HLAenumerateddatatype interface (or an IllegalAccessException occurred internally)
 */
public
HLAEnumeratedIterator(Object[] theEnumerators, boolean asStrings)
throws ClassCastException
{
    // Validate theEnumerators
    for (int i = 0; i < theEnumerators.length; i++)
    {
        if (!HLAEnumeratedDatatype.class.isAssignableFrom(theEnumerators[i].getClass()))
            throw new ClassCastException(theEnumerators[i].getClass().toString());
        if (i > 0)
        {
            // Make sure they're both the same class (if each one is castable to the other, then they must be of precisely the same class)
            if (!
                theEnumerators[i].getClass().isInstance(theEnumerators[i-1]) &&
                theEnumerators[i-1].getClass().isInstance(theEnumerators[i]) )
                throw new ClassCastException(theEnumerators[i].getClass().toString());
        }
    }
    cursor = 0;
    if (!asStrings)
    {
        enumerators = theEnumerators;
        return;
    }
// Set up a String enumeration
enumerators = new Object[theEnumerators.length];
for (int i = 0; i < enumerators.length; i++)
{
    // Obtain the class fields
    java.lang.reflect.Field[] theFields =
        theEnumerators[i].getClass().getFields();
    for (int j = 0; j < theFields.length; j++)
    {
        // To simplify the constructor, the IllegalAccessException
        // that this may throw is wrapped in a ClassCastException
        try
        {
            if (theFields[j].getType().equals(theEnumerators[i].getClass()) &&
                theFields[j].get(null) == theEnumerators[i])
            {
                enumerators[i] =
                    theEnumerators[i].getClass().getName() + "." + theFields[i].getName();
                enumerators[i] =
                    theFields[j].getName();
                break;
            }
        }
        catch (IllegalAccessException e) {
            ClassCastException cce = new
                ClassCastException(e.getMessage());
            throw (ClassCastException)cce.initCause(e);
        }
    }
}

// Iterator implementation

/**
 * Returns <code>true</code> if the iteration has more elements.
 * In other words, returns <code>true</code> if <code>next()</code> would return an element rather than throwing an exception.
 * @return a boolean which is <code>true</code> if the iterator has more elements
 */
public boolean hasNext()
{
    return cursor < enumerators.length;
}
/**
 * Returns the next element in the iteration.
 * @return the next element in the iteration
 * @throws NoSuchElementException if the iteration has no more elements
 */
 public Object next()
 {
  if (hasNext()) return enumerators[cursor++];
  throw new NoSuchElementException();
 }

/**
 * Removes from the underlying collection the last element returned by the iterator (optional operation).
 * This method can be called only once per call to <code>next()</code>.
 * The behaviour of an <code>Iterator</code> is unspecified if the underlying collection is modified
 * while the iteration is in progress in any way other than by calling this method.
 * @throws UnsupportedOperationException if this operation is not supported by this Iterator
 * @throws IllegalStateException if <code>next()</code> has not yet been called, or <code>remove()</code> has already been called since the last call to <code>next()</code>.
 */
 public void remove()
 {
  throw new UnsupportedOperationException();
 }

//end HLAEnumeratedIterator
There is only one OMT-specified enumerated datatype: HLAboolean. The MOM, on the other hand, adds eight more.

```java
// File: HLAboolean.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe boolean enumerated data type.
 * Uses the <code>HLAinteger32BE</code> basic data type with 0 for HLAfalse and 1 for HLAtrue.
 * Note that a cloned immutable instance (a cloned enumerator) will be immutable also.
 * <p>
 * It can be used as a template for other HLA enumerated data types.
 * <p>
 * Design guidelines:
 * <ul>
 * <li>A private constructor should be used to generate the static values (the enumerators).</li>
 * <li>The java.lang.Object methods toString(), equals() and hashCode() may or may not need to be overridden.</li>
 * <li>Although the inherited getValue() does not need to be overridden, this may be useful if only to change the Javadoc.</li>
 * <li>The inherited setValue, on the other hand, will definitely need to be overridden so that it throws an exception when a value other than an acceptable one is passed in, or an UnsupportedOperationException when applied to one of the static values.</li>
 * <li>Specialised get/set methods can be supplied if needed.</li>
 * <li>The enumerators should appear as a series of static public final instances.</li>
 * </ul>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.01 //Deleted commented-out inherited setValue
 */

public class HLAboolean
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(0);
    //The enumeration begins at the lowest value.
    private static HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());
```
/Whether this instance is immutable
private final boolean
_simplemutable;

/**
 * Constructs a <code>HLAboolean</code> of default (false) value.
 */
public
HLAboolean()
{
    super();
    _simplemutable = false;
}

/**
 * Constructs a <code>HLAboolean</code> of the specified boolean
 * value.
 * @param value a boolean specifying <code>this</code>' value
 */
public
HLAboolean(boolean value)
{
    this();
    setBoolean(value);
}

/**
 * Constructs a <code>HLAboolean</code> from another one.
 * @param otherHLAboolean must be a defined static value or another
 * instance
 */
public
HLAboolean(HLAboolean otherHLAboolean)
{
    this(otherHLAboolean.getBoolean());
}

/**
 * Creates a <code>HLAboolean</code> from the network representation in
 * the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of
 * the <code>HLAboolean</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAboolean(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}
/**
   * Creates a <code>HLAboolean</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
   * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAboolean</code>
   * @param offset where in the <code>buffer</code> the <code>HLAboolean</code> representation begins
   * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
   */
   public HLAboolean(byte[] buffer, int offset) throws CouldNotDecode {
      this(new ByteWrapper(buffer, offset));
   }

   /**
   * Creates a <code>HLAboolean</code> from the supplied <code>ByteWrapper</code>.
   * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAboolean</code> begins
   * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
   */
   public HLAboolean(ByteWrapper byteWrapper) throws CouldNotDecode {
      this();
      decode(byteWrapper);
   }

   /**
   * Class and subclass constructor; it is used to generate the static values.
   */
   private HLAboolean(HLAinteger32BE nextToAssign) {
      super();
      _immutable = true;
      super.setValue(nextToAssign.getValue());
      nextToAssign.setValue(nextToAssign.getValue() + 1);
   }
public Iterator iterator() throws ClassCastException
    { return new HLAenumeratedIterator(HLAboolean.class, false); }

public Iterator nameIterator() throws ClassCastException
    { return new HLAenumeratedIterator(HLAboolean.class, true); }

public Class getElementClass()
    { return HLAbigInteger32BE.class; }

// return this.getClass().getSuperclass(); //non-static only

/**
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */
public boolean
isImmutable()
{
    return _immutable;
}

//java.lang.Object methods
/**
 * Returns a <code>String</code> representation of
 * <code>this</code>.
 * This implementation should work with all HLAenumerateddatatypes;
 * HLAboolean is peculiar.
 * @return a {@link java.lang.String} reflecting <code>this</code>
 * value
 */
public String
toString()
{
    // Cannot happen, but the compiler doesn't know this
    // return "";

    /*
     * Returns a <code>String</code> representation of
     * <code>this</code>.
     * @return a {@link java.lang.String} reflecting <code>this</code>
     * value
     */
    public String
    toString()
    {
        return Boolean.toString(getBoolean());
    }
/**
 * Returns a hash code for <code>this</code>; two objects for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object)
 * @see java.util.Hashtable Hashtable
 */
public int hashCode() {
    return Boolean.valueOf(getBoolean()).hashCode();
}

//Class-specific extensions

/**
 * The inherited setValue sets <code>this</code> value from an int. 
 * A value of zero means false, any other value means true (but will be stored as 1). 
 * @param value the int new value for <code>this</code> 
 * @throws UnsupportedOperationException if used on an enumerator instance
 */
public void setValue(int value) throws UnsupportedOperationException {
    setBoolean(value != 0);
}

/**
 * Returns the boolean value of <code>this</code>. 
 * @return the boolean value of <code>this</code> 
 * @see #setBoolean
 */
public boolean getBoolean() {
    return (getValue() != 0);
}
/**
 * Sets the boolean value of <code>this</code>.  
 * @param value the boolean new value for <code>this</code>
 * @throws UnsupportedOperationException if applied to an  
 * enumerator instance
 * @see #getBoolean
 */
 public void
 setBoolean(boolean value)
 throws UnsupportedOperationException
 {
  if (_immutable) throw new UnsupportedOperationException();
  //"Cannot setBoolean of immutable instance"
  if (value)
   {
   //Must call super.setValue explicitly because setValue calls
   setBoolean:
   super.setValue(1); //HLAtrue
   }
  else
   {
   super.setValue(0); //HLAfalse
   }
 }

 //The prototype immutable instances (the enumerators)

 /**
 * False.
 */
 static public final HLAboolean
 HLAfalse = new HLAboolean(_nextToAssign);

 /**
 * True.
 */
 static public final HLAboolean
 HLAtrue = new HLAboolean(_nextToAssign);
}
//end HLAboolean
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe federateState enumerated data type, used by the Management Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 1 for ActiveFederate,
 * 3 for FederateSaveInProgress and 5 for FederateRestoreInProgress.
 * Note that a cloned immutable instance (a cloned enumerator) will be immutable also.
 */
public class HLAfederateState
extends HLAinteger32BE
implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(1);

    //Enumeration step
    private static final HLAinteger32BE _step = new HLAinteger32BE(2);

    //The enumeration begins at the lowest value.
    private static final HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean _immutable;

    /**
     * Constructs a <code>HLAfederateState</code> of default (ActiveFederate) value.
     */
    public HLAfederateState()
    {
        super(1);
        _immutable = false;
    }
}
/**
 * Constructs a <code>HLAfederateState</code> from another one.
 * @param otherHLAfederateState must be a defined static value or
 * another instance
 */
 public
 HLAfederateState(HLAfederateState otherHLAfederateState)
 {
     this();
     //Calling super.setValue directly skips the validation
     super.setValue(otherHLAfederateState.getValue());
 }

 /**
 * Creates a <code>HLAfederateState</code> from the network
 * representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAfederateState</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
 HLAfederateState(byte[] buffer)
 throws CouldNotDecode
 {
     this(buffer, 0);
 }

 /**
 * Creates a <code>HLAfederateState</code> from the network
 * representation in the provided <code>byte[]</code> at the indicated
 * <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAfederateState</code>
 * @param offset where in the <code>buffer</code> the
 * <code>HLAfederateState</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
 HLAfederateState(byte[] buffer,
                 int offset)
 throws CouldNotDecode
 {
     this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAfederateState</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfederateState</code> begins 
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded 
 */
public HLAfederateState(ByteWrapper byteWrapper) 
throws CouldNotDecode 
{ 
this(); 
decode(byteWrapper); 
}

/**
 * Class and subclass constructor; it is used to generate the static values. 
 */
private HLAfederateState(HLAinteger32BE nextToAssign) 
{ 
super(); 
_immutable = true; 
//@Calling super.setValue directly skips the validation super.setValue(nextToAssign.getValue()); 
nextToAssign.setValue(nextToAssign.getValue() + _step.getValue()); 
}

//HLA enumerated datatype implementation

/**
 * Returns an Iterator over the enumeration's elements. 
 * Note that the iterator returned by this method throws an 
 * <code>UnsupportedOperationException</code> in response to its 
 * <code>remove()</code> method. 
 * @return an Iterator over the elements in this enumeration 
 * @throws ClassCastException if the Iterator construction fails 
 */
public Iterator iterator() 
throws ClassCastException 
{ 
return new HLAenumeratedIterator(HLAfederateState.class, false); 
}
/**
 * Returns an Iterator over the enumeration’s element names.
 * Note that the iterator returned by this method throws an
 * `<code>UnsupportedOperationException</code>` in response to its
 * `<code>remove()</code>` method.
 * <p>
 * This `nameIterator()` is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain `<code>iterator()</code>` and
 * `<code>nameIterator()</code>` and then
 * go `<code>next()</code>` in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
 public Iterator
 nameIterator()
 throws ClassCastException
 {
 return new HLAenumeratedIterator(HLAfederateState.class, true);
 }

/**
 * Returns the underlying Class of the enumeration’s elements (the
 * elements' representation Class).
 * @return the underlying Class of the enumeration’s elements
 */
 public Class
 getElementClass()
 {
 return HLAinteger32BE.class;
 // return this.getClass().getSuperclass(); //non-static only
 }

/**
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */
 public boolean
 isImmutable()
 {
 return _immutable;
 }
// Java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {@link java.lang.String} reflecting <code>this</code>
 */
public String toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    // Cannot happen, but the compiler doesn't know this
    return "";
}
//class-specific extensions

/**
 * The inherited setValue sets this value from an int.
 * @param value the int new value for this
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 */
public void
setValue(int value)
throws UnsupportedOperationException,
        IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        if (((HLAresignAction)i.next()).getValue() == value)
            super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gv =
            o.getClass().getMethod("getValue", (Class[])null); //throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gv.invoke(o,
            (Object[])null))).intValue())
            throw new IllegalAccessException();
            super.setValue(value);
            return;
        }
        catch (Exception ignored) {}} } 
throw new IllegalArgumentException();}
// The prototype immutable instances (the enumerators)

/**
 * ActiveFederate.
 */
static public final HLAfederateState
ActiveFederate = new HLAfederateState(_nextToAssign);

/**
 * FederateSaveInProgress.
 */
static public final HLAfederateState
FederateSaveInProgress = new HLAfederateState(_nextToAssign);

/**
 * FederateRestoreInProgress.
 */
static public final HLAfederateState
FederateRestoreInProgress = new HLAfederateState(_nextToAssign);

} // end HLAfederateState
package ca.gc.drdc.rddc.hla.rti1516.omti;
import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe orderType enumerated data type, used by the Management
 * Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 0 for
 * Receive
 * and 1 for TimeStamp.
 * Note that a cloned immutable instance (a cloned enumerator) will be
 * immutable also.
 * See {link HLAboolean} and {link HLAenumerateddatatype} for the
design guidelines.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault]
 * (http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.2
 */
public class HLAorderType
extends HLAinteger32BE
implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE
        _lowestValue = new HLAinteger32BE(0);
    //Enumeration step
    private static final HLAinteger32BE
        _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static HLAinteger32BE
        _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean
    _immutable;

    /**
     * Constructs a <code>HLAorderType</code> of default
     * (ActiveFederate) value.
     */
    public HLAorderType()
    {
        super();
        _immutable = false;
    }
/**
 * Constructs a <code>HLAorderType</code> from another one.
 * @param otherHLAorderType must be a defined static value or another instance
 */
 public HLAorderType(HLAorderType otherHLAorderType)
 { 
   this();
   //Calling super.setValue directly skips the validation
   super.setValue(otherHLAorderType.getValue());
 }

/**
 * Creates a <code>HLAorderType</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAorderType</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAorderType(byte[] buffer)
 throws CouldNotDecode
 { 
   this(buffer, 0);
 }

/**
 * Creates a <code>HLAorderType</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAorderType</code>
 * @param offset where in the <code>buffer</code> the <code>HLAorderType</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAorderType(byte[] buffer, 
 int offset)
 throws CouldNotDecode
 { 
   this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAorderType</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAorderType</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAorderType(ByteWrapper byteWrapper)
throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

/**
 * Class and subclass constructor; it is used to generate the static values.
 */
private HLAorderType(HLAinteger32BE nextToAssign) {
    super();
    _immutable = true;
    //Calling super.setValue directly skips the validation
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() + _step.getValue());
}

//HLAenumerateddatatype implementation

/**
 * Returns an Iterator over the enumeration’s elements.
 * Note that the iterator returned by this method throws an 
 * <code>UnsupportedOperationException</code> in response to its 
 * <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator iterator() throws ClassCastException {
    return new HLAenumeratedIterator(HLAorderType.class, false);
}
/**
 * Returns an Iterator over the enumeration's element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator
nameIterator()
throws ClassCastException
{
    return new HLAenumeratedIterator(HLAorderType.class, true);
}

/***
 * Returns the underlying Class of the enumeration's elements (the
 * elements' representation Class).
 * @return the underlying Class of the enumeration's elements
 */
public Class
getElementClass()
{   return HLAinteger32BE.class;
//   return this.getClass().getSuperclass(); //non-static only
}

/***
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */
public boolean
isImmutable()
{   return _immutable;
}
//java.lang.Object methods

/**
   * Returns a <code>String</code> representation of <code>this</code>. 
   * @return a {@link java.lang.String} reflecting <code>this</code> value
   */
   public String toString()
   {
      Iterator i = iterator();
      Iterator n = nameIterator();
      String s;
      while (i.hasNext())
      {
         s = (String)n.next();
         if (i.next().equals(this)) return s;
      }
      //Cannot happen, but the compiler doesn't know this
      return "";
   }
//class-specific extensions

/**
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 *
 */
public void setValue(int value)
  throws UnsupportedOperationException, IllegalArgumentException
{
  if (isImmutable()) throw new UnsupportedOperationException();
  Iterator i = iterator();
  Object o;
  while (i.hasNext())
  {
    // if (((HLAorderType)i.next()).getValue() == value)
    super.setValue(value);
    o = i.next();
    try {
      java.lang.reflect.Method gV =
      o.getClass().getMethod("getValue", (Class[])null); //throws
      NoSuchMethodException, SecurityException
      if (value == ((Integer) (gV.invoke(o, (Object[])null))).intValue()) //throws IllegalAccessPermission, InvocationTargetException,
      {
        super.setValue(value);
        return;
      }
    } catch (Exception ignored) {}}
  throw new IllegalArgumentException();
}

//The prototype immutable instances (the enumerators)

/**
 * Receive.
 */
static public final HLAorderType Receive = new HLAorderType(_nextToAssign);

/**
 * TimeStamp.
 */
static public final HLAorderType TimeStamp = new HLAorderType(_nextToAssign);

//end HLAorderType
import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe ownership enumerated data type, used by the Management Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 0 for Unowned
 * and 1 for Owned.
 * Note that a cloned immutable instance (a cloned enumerator) will be immutable also.
 * See {@link HLAboolean} and {@link HLAenumerateddatatype} for the design guidelines.
 * @author Daniel U. Thibault [{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}]
 * @version 1.2
 */
public class HLAownership
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(0);

    //Enumeration step
    private static final HLAinteger32BE _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static final HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean _immutable;

    /**
     * Constructs a <code>HLAownership</code> of default (ActiveFederate) value.
     */
    public HLAownership()
    {
        super();
        _immutable = false;
    }
/**
 * Constructs a <code>HLAownership</code> from another one.
 * @param otherHLAownership must be a defined static value or another instance
 */
 public HLAownership(HLAownership otherHLAownership)
 {
   this();
   //Calling super.setValue directly skips the validation
   super.setValue(otherHLAownership.getValue());
 }

/**
 * Creates a <code>HLAownership</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAownership</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAownership(byte[] buffer)
 throws CouldNotDecode
 {
   this(buffer, 0);  
 }

/**
 * Creates a <code>HLAownership</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAownership</code>
 * @param offset where in the <code>buffer</code> the <code>HLAownership</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAownership(byte[] buffer, int offset)
 throws CouldNotDecode
 {
   this(new ByteWrapper(buffer, offset));  
 }
/**
 * Creates a <code>HLAownership</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAownership</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAownership(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

/**<n
 * Class and subclass constructor; it is used to generate the static values.
 */
private HLAownership(HLAnInteger32BE nextToAssign) {
    super();
    _immutable = true;
    //Calling super.setValue directly skips the validation
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() +
    _step.getValue());
}

//HLAenumerateddatatype implementation

/**<n
 * Returns an Iterator over the enumeration's elements.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator iterator() throws ClassCastException {
    return new HLAenumeratedIterator(HLAownership.class, false);
}
/**
* Returns an Iterator over the enumeration's element names.
* Note that the iterator returned by this method throws an
* <code>UnsupportedOperationException</code> in response to its
* <code>remove()</code> method.
* <p>
* This nameIterator is guaranteed to match the sequence of the
* other one.
* This means you can obtain <code>iterator()</code> and
* <code>nameIterator()</code> and then
* go <code>next()</code> in lock-step to enumerate both the
* immutable instances and their names.
* @return an Iterator over the names of the elements in this
* enumeration
* @throws ClassCastException if the Iterator construction fails
*/
public Iterator
nameIterator()
throws ClassCastException
{
    return new HLAenumeratedIterator(HLAownership.class, true);
}

/***
* Returns the underlying Class of the enumeration's elements (the
* elements' representation Class).
* @return the underlying Class of the enumeration's elements
*/
public Class
getElementClass()
{
    return HLAinteger32BE.class;
    // return this.getClass().getSuperclass(); //non-static only
}

/***
* Returns true if the instance is immutable.
* @return a boolean which is true if the instance is immutable
*/
public boolean
isImmutable()
{
    return _immutable;
}
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>.  
 * @return a <code>String</code> reflecting <code>this</code> value
 */
public String toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    //Cannot happen, but the compiler doesn't know this
    return "";
}
// class-specific extensions

/**
 * The inherited setValue sets this value from an int.
 * @param value the int new value for this
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        // if (((HLAresignAction)i.next()).getValue() == value)
        super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV =
                o.getClass().getMethod("getValue", (Class[])null); // throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gV.invoke(o, (Object[])null))).intValue()) // throws IllegalAccessError, InvocationTargetException,
            {
                super.setValue(value);
                return;
            }
        } catch (Exception ignored) {} } try
         throw new IllegalArgumentException();
    }

// the prototype immutable instances (the enumerators)

/**
 * Unowned.
 */
static public final HLAownership Unowned = new HLAownership(_nextToAssign);

/**
 * Owned.
 */
static public final HLAownership Owned = new HLAownership(_nextToAssign);
import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe resignAction enumerated data type, used by the Management
 * Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 1 for
 * DivestOwnership
 * through 6 for NoAction.
 * Note that a cloned immutable instance (a cloned enumerator) will be
 * immutable also.
 * <p>
 * See <a href="http://www.valcartier.drdc-rddc.gc.ca">HLAboolean</a> and <a href="http://www.valcartier.drdc-rddc.gc.ca">HLAenumerateddatatype</a> for the
design guidelines.
 * @author Dan Thibault
 * @version 1.2
 */
public class HLAresignAction
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE
        _lowestValue = new HLAinteger32BE(1);

    //Enumeration step
    private static final HLAinteger32BE
        _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static HLAinteger32BE
        _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean
        _immutable;

    /**
     * Constructs a <code>HLAresignAction</code> of default
     * (ActiveFederate) value.
     */
    public HLAresignAction()
    {
        super();
        _immutable = false;
    }
/**
 * Constructs a <code>HLAresignAction</code> from another one.
 * @param otherHLAresignAction must be a defined static value or
 * an other instance
 */
 public HLAresignAction(HLAresignAction otherHLAresignAction)
 {
     this();
     //Calling super.setValue directly skips the validation
     super.setValue(otherHLAresignAction.getValue());
 }

 /**
 * Creates a <code>HLAresignAction</code> from the network
 * representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAresignAction</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAresignAction(byte[] buffer)
 throws CouldNotDecode
 {
     this(buffer, 0);
 }

 /**
 * Creates a <code>HLAresignAction</code> from the network
 * representation in the provided <code>byte[]</code> at the indicated
 * <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAresignAction</code>
 * @param offset where in the <code>buffer</code> the
 * <code>HLAresignAction</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAresignAction(byte[] buffer,
    int offset)
 throws CouldNotDecode
 {
     this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAresignAction</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAresignAction</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAresignAction(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

/**
 * Class and subclass constructor; it is used to generate the static values.
 */
private HLAresignAction(HLAinteger32BE nextToAssign) {
    super();
    _immutable = true;
    //Calling super.setValue directly skips the validation
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() + _step.getValue());
}

//HLAenumerateddatatype implementation

/**
 * Returns an Iterator over the enumeration’s elements.
 * Note that the iterator returned by this method throws an <code>UnsupportedOperationException</code> in response to its <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator iterator() throws ClassCastException {
    return new HLAenumeratedIterator(HLAresignAction.class, false);
}
/**
 * Returns an Iterator over the enumeration's element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 *<p>
 * Returns the underlying Class of the enumeration's elements (the
 * elements' representation Class).
 * @return the underlying Class of the enumeration's elements
 *<p>
 * @return a boolean which is true if the instance is immutable
 *<p>
 * @param this
 * @return _immutable;
 */

public Iterator nameIterator ()
    throws ClassCastException
{
    return new HLAenumeratedIterator(HLAresignAction.class, true);
}

public Class getElementClass ()
{
    return HLAinteger32BE.class;
    //    return this.getClass().getSuperclass(); //non-static only
}

public boolean isImmutable ()
{
    return _immutable;
}
//java.lang.Object methods

/**
   * Returns a <code>String</code> representation of <code>this</code>.  
   * @return a {@link java.lang.String} reflecting <code>this</code> value
   */
   public String toString()
   {
      Iterator i = iterator();
      Iterator n = nameIterator();
      String s; 
      while (i.hasNext())
      {
         s = (String)n.next();
         if (i.next().equals(this)) return s;
      }
      //Cannot happen, but the compiler doesn't know this
      return "";
   }
//class-specific extensions

/**
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn’t one of the enumerators’
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        // if (((HLAresignAction)i.next()).getValue() == value)
        super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV =
            o.getClass().getMethod("getValue", (Class[])null); //throws
            NoSuchMethodException, SecurityException
            if (value == (Integer) (gV.invoke(o,
            (Object[])null)).intValue()) //throws IllegalArgumentException,
            IllegalArgumentExcepti
            on, InvocationTargetException
            {
                super.setValue(value);
                return;
            }
        } catch (Exception ignored) {}}
    throw new IllegalArgumentException();
}
//The prototype immutable instances (the enumerators)

/**
 * DivestOwnership.
 */
static public final HLAresignAction DivestOwnership = new HLAresignAction(_nextToAssign);

/**
 * DeleteObjectInstances.
 */
static public final HLAresignAction DeleteObjectInstances = new HLAresignAction(_nextToAssign);

/**
 * CancelPendingAcquisitions.
 */
static public final HLAresignAction CancelPendingAcquisitions = new HLAresignAction(_nextToAssign);

/**
 * DeleteObjectInstancesThenDivestOwnership.
 */
static public final HLAresignAction DeleteObjectInstancesThenDivestOwnership = new HLAresignAction(_nextToAssign);

/**
 * CancelPendingAcquisitionsThenDeleteObjectInstancesThenDivestOwnership.
 */
static public final HLAresignAction CancelPendingAcquisitionsThenDeleteObjectInstancesThenDivestOwnership = new HLAresignAction(_nextToAssign);

/**
 * NoAction.
 */
static public final HLAresignAction NoAction = new HLAresignAction(_nextToAssign);
// File: HLAserviceGroupName.java
package ca.gc.drdc.rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe serviceGroupName enumerated data type, used by the Management Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 0 for FederationManagement
 * through 6 for SupportServices.
 * Note that a cloned immutable instance (a cloned enumerator) will be immutable also.
 * @see {@link HLAboolean} and {@link HLAenumerateddatatype} for the design guidelines.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.2
 */
public class HLAserviceGroupName
extends HLAinteger32BE
implements HLAenumerateddatatype
{

    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(0);

    //Enumeration step
    private static final HLAinteger32BE _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean _immutable;

    /**
     * Constructs a <code>HLAserviceGroupName</code> of default (FederationManagement) value.
     */
    public HLAserviceGroupName()
    {
        super();
        _immutable = false;
    }
/**
 * Constructs a <code>HLAserviceGroupName</code> from another one.
 * @param otherHLAserviceGroupName must be a defined static value
 * or another instance
 */
 public HLAserviceGroupName(HLAserviceGroupName otherHLAserviceGroupName)
 {
   this();
   //Calling super.setValue directly skips the validation
   super.setValue(otherHLAserviceGroupName.getValue());
 }

 /**
 * Creates a <code>HLAserviceGroupName</code> from the network
 * representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAserviceGroupName</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAserviceGroupName(byte[] buffer)
 throws CouldNotDecode
 {
   this(buffer, 0);
 }

 /**
 * Creates a <code>HLAserviceGroupName</code> from the network
 * representation in the provided <code>byte[]</code> at the indicated
 * <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAserviceGroupName</code>
 * @param offset where in the <code>buffer</code> the
 * <code>HLAserviceGroupName</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAserviceGroupName(byte[] buffer,
   int offset)
 throws CouldNotDecode
 {
   this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAserviceGroupName</code> from the supplied
 * <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current
 * <code>pos()</code> the representation of the
 * <code>HLAserviceGroupName</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not
 * be decoded
 */

public HLAserviceGroupName(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}

/**
 * Class and subclass constructor; it is used to generate the
 * static values.
 */

private HLAserviceGroupName(HLAinteger32BE nextToAssign)
{
    super();
    _immutable = true;
    //Calling super.setValue directly skips the validation
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() +
        _step.getValue());
}

//HLAenumerateddatatype implementation

/**
 * Returns an Iterator over the enumeration's elements.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */

public Iterator
    throws ClassCastException
{
    return new HLAenumeratedIterator(HLAserviceGroupName.class,
        false);
}
/**
 * Returns an Iterator over the enumeration’s element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
 public Iterator
 nameIterator()
 throws ClassCastException
 {
 return new HLAEnumeratedIterator(HLAServiceGroupName.class,
 true);
 }

 /**
 * Returns the underlying Class of the enumeration’s elements (the
 * elements’ representation Class).
 * @return the underlying Class of the enumeration’s elements
 */
 public Class
 getElementClass()
 { return HLAInteger32BE.class;
 // return this.getClass().getSuperclass(); //non-static only
 }

 /**
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */
 public boolean
 isImmutable()
 { return _immutable;
 }
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of 
 * <code>this</code>.  
 * @return a {@link java.lang.String} reflecting <code>this</code> value
 */
public String
toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    //Cannot happen, but the compiler doesn't know this
    return "";
}
/**
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        if (((HLAserviceGroupName)i.next()).getValue() == value)
        super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV = o.getClass().getMethod("getValue", (Class[])null); //throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gV.invoke(o, (Object[][])null))).intValue) //throws IlleagalAccessException, IllegalArgumentException, InvocationTargetException
            {
                super.setValue(value);
                return;
            }
        } catch (Exception ignored) {}
    }
    throw new IllegalArgumentException();
}
The prototype immutable instances (the enumerators)

/**
 * FederationManagement.
 */
static public final HLAserviceGroupName
FederationManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * DeclarationManagement.
 */
static public final HLAserviceGroupName
DeclarationManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * ObjectManagement.
 */
static public final HLAserviceGroupName
ObjectManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * OwnershipManagement.
 */
static public final HLAserviceGroupName
OwnershipManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * TimeManagement.
 */
static public final HLAserviceGroupName
TimeManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * DataDistributionManagement.
 */
static public final HLAserviceGroupName
DataDistributionManagement = new HLAserviceGroupName(_nextToAssign);

/**
 * SupportServices.
 */
static public final HLAserviceGroupName
SupportServices = new HLAserviceGroupName(_nextToAssign);
}
public class HLAswitch
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE
        _lowestValue = new HLAinteger32BE(0);

    //Enumeration step
    private static final HLAinteger32BE
        _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static final HLAinteger32BE
        _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean
        _immutable;

    /**
     * Constructs a <code>HLAswitch</code> of default (ActiveFederate) value.
     */
    public HLAswitch()
    {
        super();
        _immutable = false;
    }
}
/**
 * Constructs a <code>HLAswitch</code> from another one.
 * @param otherHLAswitch must be a defined static value or another
 * instance
 */
 public HLAswitch(HLAswitch otherHLAswitch)
 {
   this();
   //Calling super.setValue directly skips the validation
   super.setValue(otherHLAswitch.getValue());
 }

/**
 * Creates a <code>HLAswitch</code> from the network representation
 * in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAswitch</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAswitch(byte[] buffer)
 throws CouldNotDecode
 {
   this(buffer, 0);
 }

/**
 * Creates a <code>HLAswitch</code> from the network representation
 * in the provided <code>byte[]</code> at the indicated
 * <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAswitch</code>
 * @param offset where in the <code>buffer</code> the
 * <code>HLAswitch</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAswitch(byte[] buffer,
   int offset)
 throws CouldNotDecode
 {
   this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAswitch</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAswitch</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAswitch(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

/**
 * Class and subclass constructor; it is used to generate the static values.
 */
private HLAswitch(HLAinteger32BE nextToAssign) {
    super();
    _immutable = true;
    //Calling super.setValue directly skips the validation
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() +
    _step.getValue());
}

//HLAenumerateddatatype implementation

/**
 * Returns an Iterator over the enumeration’s elements.
 * Note that the iterator returned by this method throws an 
 * <code>UnsupportedOperationException</code> in response to its 
 * <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator iterator() throws ClassCastException {
    return new HLAenumeratedIterator(HLAswitch.class, false);
}
/**
 * Returns an Iterator over the enumeration's element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one. This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
public Iterator
nameIterator()
throws ClassCastException
{
    return new HLAenumeratedIterator(HLAswitch.class, true);
}

/**
 * Returns the underlying Class of the enumeration's elements (the
 elements' representation Class).
 * @return the underlying Class of the enumeration's elements
 */
public Class
getElementClass()
{
    return HLAinteger32BE.class;
    // return this.getClass().getSuperclass(); //non-static only
}

/**
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */
public boolean
isImmutable()
{
    return _immutable;
}
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>. 
 * @return a {@link java.lang.String} reflecting <code>this</code> value
 */
public String toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    //Cannot happen, but the compiler doesn't know this return "";
}
// class-specific extensions

/**
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        // if (((HLAswitch)i.next()).getValue() == value)
        super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV =
            o.getClass().getMethod("getValue", (Class[])(null)); // throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gV.invoke(o,
            (Object[])(null))).intValue())) // throws IllegalArgumentException,
            IllegalAccessError, InvocationTargetException
            {
                super.setValue(value);
                return;
            }
        }
        catch (Exception ignored) {}
    }
    throw new IllegalArgumentException();
}

// The prototype immutable instances (the enumerators)

/**
 * Disabled.
 */
static public final HLAswitch Disabled = new HLAswitch(_nextToAssign);

/**
 * Enabled.
 */
static public final HLAswitch Enabled = new HLAswitch(_nextToAssign);

// end HLAswitch
public class HLAsyncPointStatus
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(0);

    //Enumeration step
    private static final HLAinteger32BE _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean _immutable;

    /**
     * Constructs a <code>HLAsyncPointStatus</code> of default
     * <code>ActiveFederate</code> value.
     */
    public HLAsyncPointStatus()
    {
        super();
        _immutable = false;
    }
** Constructs a `<code>HLAsyncPointStatus</code>` from another one.
* @param otherHLAsyncPointStatus must be a defined static value or another instance
*/
public
HLAsyncPointStatus(HLAsyncPointStatus otherHLAsyncPointStatus)
{
    this();
    //Calling super.setValue directly skips the validation
    super.setValue(otherHLAsyncPointStatus.getValue());
}

/**
 * Creates a `<code>HLAsyncPointStatus</code>` from the network representation in the provided `<code>byte[]</code>`.
* @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAsyncPointStatus</code>`
* @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
*/
public
HLAsyncPointStatus(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a `<code>HLAsyncPointStatus</code>` from the network representation in the provided `<code>byte[]</code>` at the indicated `<code>offset</code>`.
* @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAsyncPointStatus</code>`
* @param offset where in the `<code>buffer</code>` the `<code>HLAsyncPointStatus</code>` representation begins
* @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
*/
public
HLAsyncPointStatus(byte[] buffer, int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a <code>HLAsyncPointStatus</code> from the supplied
 * <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current
 * <code>pos()</code> the representation of the
 * <code>HLAsyncPointStatus</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not
 * be decoded
 */
 public HLAsyncPointStatus(ByteWrapper byteWrapper)
 throws CouldNotDecode
 {
   this();
   decode(byteWrapper);
 }

 /**
 * Class and subclass constructor; it is used to generate the
 * static values.
 */
 private HLAsyncPointStatus(HLAinteger32BE nextToAssign)
 {
   super();
   _immutable = true;
   //Calling super.setValue directly skips the validation
   super.setValue(nextToAssign.getValue());
   nextToAssign.setValue(nextToAssign.getValue() +
   _step.getValue());
 }

 //HLAenumerateddatatype implementation

 /**
 * Returns an Iterator over the enumeration’s elements.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 * @throws ClassCastException if the Iterator construction fails
 */
 public Iterator iterator()
 throws ClassCastException
 {
   return new HLAenumeratedIterator(HLAsyncPointStatus.class,
   false);
 }
/**
 * Returns an Iterator over the enumeration's element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 * /
 *
 * public Iterator
 * nameIterator()
 * throws ClassCastException
 * {
 *   return new HLAenumeratedIterator(HLAasyncPointStatus.class,
 *   true);
 * }
 *
 /*
 * Returns the underlying Class of the enumeration's elements (the
 * elements' representation Class).
 * @return the underlying Class of the enumeration's elements
 * *
 * public Class
 * getElementClass()
 * {
 *   return HLAinteger32BE.class;
 * //   return this.getClass().getSuperclass(); //non-static only
 * }
 *
 /*
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 * *
 * public boolean
 * isImmutable()
 * {
 *   return _immutable;
 * }
//java.lang.Object methods
/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {@link java.lang.String} reflecting <code>this</code> value
 */
public String toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    //Cannot happen, but the compiler doesn't know this
    return "";
}
//class-specific extensions

/**
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator instance
 * @throws IllegalArgumentExeption if the value isn't one of the enumerators
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        // if (((HLAsyncPointStatus)i.next()).getValue() == value)
        super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV =
                o.getClass().getMethod("getValue", (Class[])null); //throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gV.invoke(o,
                (Object[])null))).intValue()) //throws IllegalArgumentException, InvocationTargetException
            {
                super.setValue(value);
                return;
            }
        } catch (Exception ignored) {} 
    }
    throw new IllegalArgumentException();
}
// The prototype immutable instances (the enumerators)

/**
 * NoActivity.
 */
static public final HLAsyncPointStatus
NoActivity = new HLAsyncPointStatus(_nextToAssign);

/**
 * AttemptingToRegisterSyncPoint.
 */
static public final HLAsyncPointStatus
AttemptingToRegisterSyncPoint = new
HLAsyncPointStatus(_nextToAssign);

/**
 * MovingToSyncPoint.
 */
static public final HLAsyncPointStatus
MovingToSyncPoint = new HLAsyncPointStatus(_nextToAssign);

/**
 * WaitingForRestOfFederation.
 */
static public final HLAsyncPointStatus
WaitingForRestOfFederation = new HLAsyncPointStatus(_nextToAssign);

} //end HLAsyncPointStatus
package ca.gc.drdc.rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Type-safe timeState enumerated data type, used by the Management Object Model (MOM).
 * Uses the <code>HLAinteger32BE</code> basic data type with 0 for TimeGranted
 * and 1 for TimeAdvancing.
 * Note that a cloned immutable instance (a cloned enumerator) will be immutable also.
 * See {@link HLAinteger32BE} for the design guidelines.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel Thibault}
 * @version 1.2
 */
public class HLAtimeState
    extends HLAinteger32BE
    implements HLAenumerateddatatype
{
    //Initial value for enumeration
    private static final HLAinteger32BE _lowestValue = new HLAinteger32BE(0);

    //Enumeration step
    private static final HLAinteger32BE _step = new HLAinteger32BE(1);

    //The enumeration begins at the lowest value.
    private static HLAinteger32BE _nextToAssign = new HLAinteger32BE(_lowestValue.getValue());

    //Whether this instance is immutable
    private final boolean _immutable;

    /**
     * Constructs a <code>HLAtimeState</code> of default (ActiveFederate) value.
     */
    public HLAtimeState()
    {
        super();
        _immutable = false;
    }
/**
 * Constructs a <code>HLAtimeState</code> from another one.
 * @param otherHLAtimeState must be a defined static value or another instance
 */
 public HLAtimeState(HLAtimeState otherHLAtimeState)
 { this();
   //Calling super.setValue directly skips the validation
   super.setValue(otherHLAtimeState.getValue());
 }

/**
 * Creates a <code>HLAtimeState</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAtimeState</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAtimeState(byte[] buffer)
 { throws CouldNotDecode
   this(buffer, 0);
 }

/**
 * Creates a <code>HLAtimeState</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAtimeState</code>
 * @param offset where in the <code>buffer</code> the <code>HLAtimeState</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAtimeState(byte[] buffer,
                     int offset)
 { throws CouldNotDecode
   this(new ByteWrapper(buffer, offset));
 }
/**
 * Creates a <code>HLAtimeState</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAtimeState</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAtimeState(ByteWrapper byteWrapper)
    throws CouldNotDecode
    this();
    decode(byteWrapper);
}

/**
 * Class and subclass constructor; it is used to generate the static values.
 */
private HLAtimeState(HLAinteger32BE nextToAssign)
    super();
    _immutable = true;
    super.setValue(nextToAssign.getValue());
    nextToAssign.setValue(nextToAssign.getValue() + _step.getValue());
}

//HLAenumerateddatatype implementation

/**
 * Returns an Iterator over the enumeration's elements.
 * Note that the iterator returned by this method throws an <code>UnsupportedOperationException</code> in response to its <code>remove()</code> method.
 * @return an Iterator over the elements in this enumeration
 */
public Iterator iterator()
    throws ClassCastException
    return new HLAenumeratedIterator(HLAtimeState.class, false);
/**
 * Returns an Iterator over the enumeration's element names.
 * Note that the iterator returned by this method throws an
 * <code>UnsupportedOperationException</code> in response to its
 * <code>remove()</code> method.
 * <p>
 * This nameIterator is guaranteed to match the sequence of the
 * other one.
 * This means you can obtain <code>iterator()</code> and
 * <code>nameIterator()</code> and then
 * go <code>next()</code> in lock-step to enumerate both the
 * immutable instances and their names.
 * @return an Iterator over the names of the elements in this
 * enumeration
 * @throws ClassCastException if the Iterator construction fails
 */

public Iterator
nameIterator()
throws ClassCastException
{
return new HLAEnumeratedIterator(HLAtimeState.class, true);
}

/**
 * Returns the underlying Class of the enumeration's elements (the
 * elements' representation Class).
 * @return the underlying Class of the enumeration's elements
 */

public Class
getElementClass()
{
return HLAinteger32BE.class;
// return this.getClass().getSuperclass(); // non-static only
}

/**
 * Returns true if the instance is immutable.
 * @return a boolean which is true if the instance is immutable
 */

public boolean
isImmutable()
{
return _immutable;
}
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>. 
 * @return a {@link java.lang.String} reflecting <code>this</code> value 
 */
public String toString()
{
    Iterator i = iterator();
    Iterator n = nameIterator();
    String s;
    while (i.hasNext())
    {
        s = (String)n.next();
        if (i.next().equals(this)) return s;
    }
    //Cannot happen, but the compiler doesn't know this
    return "";
}
//class-specific extensions
/*
 * The inherited setValue sets <code>this</code> value from an int.
 * @param value the int new value for <code>this</code>
 * @throws UnsupportedOperationException if used on an enumerator
 * instance
 * @throws IllegalArgumentException if the value isn't one of the enumerators'
 */
public void setValue(int value)
    throws UnsupportedOperationException, IllegalArgumentException
{
    if (isImmutable()) throw new UnsupportedOperationException();
    Iterator i = iterator();
    Object o;
    while (i.hasNext())
    {
        if (((HLAresignAction)i.next()).getValue() == value)
            super.setValue(value);
        o = i.next();
        try {
            java.lang.reflect.Method gV = o.getClass().getMethod("getValue", (Class[])null); //throws
            NoSuchMethodException, SecurityException
            if (value == ((Integer)(gV.invoke(o, (Object[])null))).intValue())
                //throws IllegalArgumentException, InvocationTargetException
                try {
                    super.setValue(value);
                    return;
                }
            catch (Exception ignored) {}
        }
        throw new IllegalArgumentException();
    }
}

//The prototype immutable instances (the enumerators)
/*
 * ActiveFederate.
 */
static public final HLAtimeState TimeGranted = new HLAtimeState(_nextToAssign);

/*
 * TimeAdvancing.
 */
static public final HLAtimeState TimeAdvancing = new HLAtimeState(_nextToAssign);
} //end HLAtimeState
The HLAArraydatatype interface combines HLAdatatype with Java's List. The HLAArrayType abstract class extends Java's AbstractList and provides the foundation for the two (abstract) subclasses HLAVariableArrayType and HLAFixedArrayType.

```java
// File: HLAarraydatatype.java
package ca.gc.drdc_rddc.hla.rtl1516.omti

/**
 * Interface implemented by the various HLA array data types.
 * In addition to this interface, the HLA array data type classes are
 * expected to supply:
 * <ul>
 * <li> Constructors (default and specified-value)</li>
 * <li> Constructor (byte[] buffer) throws CouldNotDecode</li>
 * <li> Constructor (byte[] buffer, int offset) throws
 *     CouldNotDecode</li>
 * <li> Constructor (ByteWrapper byteWrapper) throws
 *     CouldNotDecode</li>
 * <li> java.lang.Object methods toString(); equals(Object
 *     otherObject) and hashCode()</li>
 * <li> Class-specific extensions to get and set the value as a basic
 *     java data type (int, boolean, etc.)</li>
 * </ul>
 *
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 *     Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 *     Valcartier}}
 * @version 1.1
 */
public interface HLAarraydatatype
  extends HLAdatatype,
          java.util.List
{  
  /**
   * Returns the Class of the array's elements.
   * @return the Class of the array's elements
   */
  Class getElementType();

  /**
   * Returns a boolean which is <code>true</code> if the array is
dynamic.
   * @return a boolean which is true if the array is dynamic
   */
  boolean isDynamic();
  }

//end HLAarraydatatype
```
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Abstract ancestor for the type-safe array data types.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */
public abstract class HLAarrayType
    extends java.util.AbstractList // thus implements java.util.List
    implements java.io.Serializable,
             java.lang.Cloneable,
             java.util.RandomAccess,
             HLAarraydatatype
{
    //If at all possible, it is strongly recommended that the concrete
    class declare this field:

    // public static final int
    // octetBoundary = <whatever>;

    //HLAdatatype interface implementation

    /**
     * Encodes <code>this</code> into the <code>byte[]</code> at the
     * specified <code>offset</code>.
     * @param buffer the <code>byte[]</code> into which to encode
     *               <code>this</code>
     * @param offset the offset into the <code>byte[]</code> at which
     *               to encode <code>this</code>
     * @return how many bytes were written to the buffer, including any
     *         prefix padding bytes
     */
    public int encode(byte[] buffer,
                      int offset)
    {
        return encode(new ByteWrapper(buffer, offset)).pos() - offset;
    }
/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper encode(ByteWrapper byteWrapper) {
    byteWrapper.align(octetBoundary());
    // Variable arrays prefix the encode with the number of elements;
    // static arrays do not
    if (isDynamic()) (new HLAIinteger32BE(size())).encode(byteWrapper);
    // And then one just encodes the individual elements
    Iterator i = iterator();
    while (i.hasNext()) ((HLA datatype)(i.next(1))).encode(byteWrapper);
    return byteWrapper;
}

/**
 * Encodes <code>this</code> into a new <code>byte[]</code>.
 * @return a <code>byte[]</code> encoding <code>this</code>
 */
public byte[] toByteArray() {
    return encode(new ByteWrapper(encodedLength())).array();
}

/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the data type
 * @return how many bytes were read from the <code>byte[]</code>
 * @throws CouldNotDecode if the <code>byte[]</code> could not be decoded
 */
public int decode(byte[] buffer) throws CouldNotDecode {
    return decode(buffer, 0);
}
/**
 * Sets <code>this</code> value from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> <code>this</code> representation begins
 * @return where in the <code>buffer</code> <code>this</code> representation ends
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public int
decode(byte[] buffer, int offset)
throws CouldNotDecode
{
    try
    {
        return decode(new ByteWrapper(buffer, offset)).pos();
    }
    catch (Exception e)
    {
        CouldNotDecode cn = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cn.initCause(e);
    }
}

/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    try
    {
        int count;
        // java.lang.reflect.Constructor cons =
        getElementType().getConstructor(new Class[] { ByteWrapper.class });
        java.lang.reflect.Constructor cons =
        java.lang.reflect.Constructor cons =
        getElementType().getConstructor(new Class[] { ByteWrapper.class });
        
        return decode(new ByteWrapper(buffer, offset)).pos();
    }
    catch (Exception e)
    {
        CouldNotDecode cn = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cn.initCause(e);
    }
}
byteWrapper.align(octetBoundary());
if (isDynamic())
{
    //Dynamic arrays
    clear();
    //First we read the number of elements to decode
    count = byteWrapper.getInt();
    //Then we decode the individual elements
    for (int i = 0; i < count; i++) add(cons.newInstance(new
        Object[] { byteWrapper }));
}
else
{
    //Static arrays
    count = size();
    //Decode the individual elements
    for (int i = 0; i < count; i++) set(i,
        cons.newInstance(new Object[] { byteWrapper }));
}
return byteWrapper;
}
catch (Exception e)
{
    CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
    throw (CouldNotDecode)cnd.initCause(e);
}

//java.lang.Object methods
/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
 * represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return true iff supplied <code>otherObject</code>
 * is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object .hashCode ()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (!super.equals(otherObject)) return false;
    //At this point, we know both this and otherObject implement the
    List interface
    //and have the same iteration of elements (using equals at their
    level)
    //All we need do here is make sure otherObject is of this' class
    return this.getClass().equals(otherObject.getClass());
}
//Cloneable implementation

/**
 * Creates and returns a copy of this object.
 * @return an independent copy of this Object
 * @throws CloneNotSupportedException if the object's class is not
 * Cloneable or if the instance cannot be cloned
 */
public Object clone()
    throws CloneNotSupportedException
{
    // throw new CloneNotSupportedException();
    return super.clone();
}
//end HLAarrayType
// File: HLAfixedArrayType.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

/**
 * Abstract ancestor for the type-safe fixed array data types.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}]
 * @version 1.1
 */
public abstract class HLAfixedArrayType
    extends HLAarrayType
{
    //HLAarraydatatype interface implementation
    //Being based on java.util.AbstractList, concrete fixed array
classes will need to
    //provide implementations for:
    //get(int index)
    //size()
    //set(int index, Object element)
    //You should provide two constructors (void and collection
arguments).
    //You do not have to provide an Iterator implementation.
    //List methods may be overridden if a more efficient implementation
is possible.
    /**
     * Returns a boolean which is <code>true</code> if the array is
dynamic.
     * @return a boolean which is true if the array is dynamic
     */
    public final boolean isDynamic()
    {
        return false;
    }
} //end HLAfixedArrayType
package ca.gc.drdc_rddc.hla.rti1516.omti;

/**
 * Abstract ancestor for the type-safe dynamic array data types.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public abstract class HLAvariableArrayType extends HLAarrayType {
    //Being based on java.util.AbstractList, concrete variable array classes will need to
    //provide implementations for:
    //get(int index)
    //size()
    //set(int index, Object element)
    //add(int index, Object element)
    //remove(int index)
    //You should provide two constructors (void and collection arguments).
    //You do not have to provide an Iterator implementation.
    //List methods may be overridden if a more efficient implementation is possible.

    /**
     * Returns a boolean which is true if the array is dynamic.
     * @return a boolean which is true if the array is dynamic
     */
    public final boolean isDynamic() {
        return true;
    }
} //end HLAVariableArrayType
We implement the OMT's HLAopaqueData, HLAASCII string and HLAunicodeString, grouping the last two under the abstract HLAstring class. Our concrete HLAobjectArray will serve as ancestor for most remaining array datatypes.

```java
// File: HLAopaqueData.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;
import java.util.Iterator;

/**
 * Type-safe variable byte array data type.
 * It uses <code>HLAbyte</code> elements and a Java <code>byte[]</code>
 * internally.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (DRDC Valcartier)
 * @version 1.1
 */
public class HLAopaqueData
extends HLAvariableArrayType
{
    /** Octet boundary of this class. */
    public static final int octetBoundary = HLAinteger32BE.octetBoundary;

    /** The array of bytes. */
    protected transient byte[] _values;

    /** Constructs an empty <code>HLAopaqueData</code>.
     */
    public HLAopaqueData()
    {
        //super() not called because super-class is abstract
        _values = new byte[0];
    }
```
public HLAopaqueData(Collection c) {
    // We could start with an empty _values (by invoking this())
    // and then add() using the c.iterator(), but that is relatively
    // inefficient.
    _values = new byte[c.size()];
    Iterator it = c.iterator();
    for (int i = 0; it.hasNext(); i++)
    {
        set(i, it.next());
    }
}

/**
 * Constructs an <code>HLAopaqueData</code> containing the specified byte value.
 * @param value a byte specifying <code>this</code>'s value
 */
public HLAopaqueData(byte value)
{
    this();
    add(new Byte(value));
}

/**
 * Creates an <code>HLAopaqueData</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAopaqueData</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAopaqueData(byte[] buffer)
    throws CouldNotDecode
{
    this(buffer, 0);
}
/**
 * Creates an <code>HLAopaqueData</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAopaqueData</code>
 * @param offset where in the <code>buffer</code> the <code>HLAopaqueData</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAopaqueData(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates an <code>HLAopaqueData</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAopaqueData</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAopaqueData(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
}

/**
 * Sets the size of the array to the specified value.
 * Reducing the size will result in the loss of the "chopped off" bytes.
 * @param newSize an int specifying the array's desired new size
 */
protected void setSize(int newSize) {
    modCount++;
    int bytesToCopy = _values.length;
    if (newSize < bytesToCopy) bytesToCopy = newSize;
    byte oldValues[] = _values;
    _values = new byte[newSize];
    System.arraycopy(oldValues, 0, _values, 0, bytesToCopy);
}
//List interface implementation
/**
 * Inserts the specified element at the specified position in this
 * array.
 * Elements must be compatible with HLAByte; this is why this
 * implementation accepts only Byte and HLAoctet (and their descendants).
 * @param index an int indicating the position before which to
 * insert the element
 * @param element an Object to insert before index
 * @throws UnsupportedOperationException if this method is not
 * supported by this list
 * @throws ClassCastException if the class of the specified element
 * prevents it from being added to this list
 * @throws NullPointerException if the specified element is null
 * and this list does not support null elements
 * @throws IllegalArgumentException if some aspect of this element
 * prevents it from being added to this list
 * @throws IndexOutOfBoundsException if <code>index</code> is
 * negative or larger than <code>size()</code>
 */
public void add(int index, Object element)
{
if ((index > size()) || (index < 0)) throw new
IndexOutOfBoundsException(Integer.toString(index));
if (element == null) throw new NullPointerException();
byte value;
if (element instanceof Byte)
{
value = ((Byte)element).byteValue();
}
else if (element instanceof HLAoctet)
{
value = ((HLAoctet)element).getValue();
}
else
{
throw new ClassCastException(element.getClass().toString());
}
setSize(size() + 1); //increments modCount
System.arraycopy(_values, index, _values, index + 1, size() -
(index + 1));
_values[index] = value;
/**
 * Inserts all of the elements in the specified collection into
 * this array at the specified position.
 * @param index an int indicating before which element to insert
 * the collection
 * @param c a Collection of elements to insert
 * @return a boolean which is true if this array changed as a
 * result of this call
 * @throws IllegalArgumentException if any character to append is
 * not in the 00..7F range
 */
public boolean addAll(int index, Collection c)
{
    boolean modified = false;
    //Presume abnormal termination
    boolean abnormal = true;
    //Preserve the old array and modCount
    byte oldValues[] = new byte[size()];
    System.arraycopy(_values, 0, oldValues, 0, size());
    int m = modCount;
    try
    {
        modified = super.addAll(index, c);
        abnormal = false;
    }
    finally
    {
        if (abnormal)
        {
            //If any of the add() failed, restore the old String and
            //Preserve fail-fast iterators (and list iterators)
            modCount = m;
            modified = false;
        }
    }
    return modified; //If one has a return within a finally, one
    gets a warning
}

/**
 * Removes all of the elements from this array.
 */
public void clear()
{
    //This implementation is more efficient than AbstractList.clear()
    _values = new byte[0];
    //Provide fail-fast iterators (and list iterators)
    modCount++;
/**
 * Returns the element (a Byte) at the specified position in this array.
 * @param index an int specifying the element to return
 * @return the requested Object (a Byte)
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object get(int index) {
    return new HLAByte(_values[index]);
}

/**
 * Removes the element at the specified position in this array and returns it.
 * @param index an int indicating the position from which to remove the element
 * @return the element that was removed from the list
 * @throws UnsupportedOperationException if this method is not supported by this list
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object remove(int index) {
    if ((index > size()) || (index < 0)) throw new IndexOutOfBoundsException(Integer.toString(index));
    Object o = get(index);
    System.arraycopy(_values, index + 1, _values, index, size() - (index + 1));
    setSize(size() - 1); // increments modCount
    return o;
}
/**
 * Replaces the element at the specified position in this array
 * with the specified element.
 * The element must be compatible with HLAByte; this is why this
 * implementation accepts only Byte and HLAoctet (and their descendants).
 * @param index an int specifying the index of the element to
 * replace
 * @param element the Object to use in replacing the element
 * @return the element previously at the specified position
 * @throws UnsupportedOperationException if this method is not
 * supported by this list
 * @throws ClassCastException if the class of the specified element
 * prevents it from being added to this list
 * @throws NullPointerException if the specified element is null
 * and this list does not support null elements
 * @throws IllegalArgumentException if some aspect of the specified
 * element prevents it from being added to this list
 * @throws IndexOutOfBoundsException if the index is outside the
 * [0..size()[ range
 */
public Object
set(int index,
   Object element)
{
   if ((index > size()) || (index < 0)) throw new
   IndexOutOfBoundsException(Integer.toString(index));
   if (element == null) throw new NullPointerException();
   byte value;
   if (element instanceof Byte)
   {
      value = ((Byte)element).byteValue();
   }
   else if (element instanceof HLAoctet)
   {
      value = ((HLAoctet)element).getValue();
   }
   else
   {
      throw new ClassCastException(element.getClass().toString());
   }
   Object o = get(index);
   _values[index] = value;
   //Note that set() does not modify the array structurally
   return o;
}

/**
 * Returns the number of elements in this array.
 * @return an int specifying the number of elements in this array
 */
public int
size()
{
   return _values.length;
}
//java.lang.Object methods

/**
 * Returns a <code>String</code> representation of <code>this</code>. Here we choose to use the form "[ xx xx ]" where xx is the hexadecimal representation of each byte.
 * @return a <code>String</code> reflecting <code>this</code>
 */
public String toString ()
{
    String s = " ";
    Iterator i = iterator();
    while (i.hasNext()) s = s + Integer.toHexString((int) ( (HLAoctet) i .next ()).getValue()) + " ";
    return "[" + s + "]";
}

//HLAdatatype interface implementation

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>. This implementation is more efficient than the ancestor's.
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper encode (ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    (new HLAinteger32BE(size())).encode(byteWrapper);
    byteWrapper.read(_values);
    return byteWrapper;
}

/**
 * Encodes <code>this</code> into a new <code>byte[]</code>. This implementation is more efficient than the ancestor's.
 * @return a <code>byte[]</code> encoding <code>this</code>
 */
public byte[] toByteArray()
{
    return _values;
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>
 * This implementation is more efficient than the ancestor's.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper)
throws CouldNotDecode
{
try
{
byteWrapper.align(octetBoundary());
//First we read the number of elements to decode
int count = byteWrapper.getInt();
setSize(count); //increments modCount
//Then we decode the individual elements
byteWrapper.write(_values);
return byteWrapper;
}
catch (Exception e)
{
CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
throw (CouldNotDecode)cnd.initCause(e);
}
}

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.  
 * As a general rule, Variant Records [HLAvariantRecord], Dynamic Arrays [HLAvariableArray], and any 
 * Fixed Arrays [HLAfixedArray] or Fixed Records [HLAfixedRecord] containing either of these first two 
 * cannot have a constant encodedLength.
 * @return the length (in bytes) of the <code>byte[]</code>
 */
public final int
encodedLength()
{
return HLAinteger32BE.encodedLength + size();
}
/** *
 * Returns the octet boundary of <code>this</code>. *
 * The octet boundary value is defined as the smallest power of 2 which is greater than or equal to the size of the datatype in bytes. *
 * For a constructed datatype, it is the maximum octet boundary value of all components within it. *
 * For dynamic arrays, octet boundary is thus the largest of the count (an HLAinteger32BE) and the element. *
 * @return the octet boundary of <code>this</code> *
 */
public final int octetBoundary()
{
    return octetBoundary;
}

//HLAArraydatatype interface implementation

/** *
 * Returns the Class of the array's elements. *
 * @return the Class of the array's elements *
 */
public Class getElementClass()
{
    return HLAMemory.class;
}

//end HLAMemory
// File: HLAstring.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecodem;
//import java.util.Collection;

/**
 * Type-safe abstract String variable array data type.
 * It uses <code>char</code> elements and a Java <code>String</code>
 * internally;
 * it is the ancestor of the HLAASCIIString and HLAunicodeString
 * concrete classes.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} { {@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier})
 * @version 1.2
 */
public abstract class
HLAString
extends HLAvariableArrayType
{
/** Octet boundary of this class. */
private final static int
octetBoundary = HLAinteger32BE.octetBoundary;

/** The String variable array: a Java String. */
protected String _value;
//List interface implementation

/**
 * Inserts the specified element at the specified position in this array.
 * Elements must be compatible with HLAUnicodeChar; this is true as long as the element's toString() method
 * returns a non-zero length String. Zero-length Strings are not acceptable.
 * @param index an int indicating the position before which to insert the element
 * @param element an Object to insert before index
 * @throws UnsupportedOperationException if this method is not supported by this list
 * @throws ClassCastException if the class of the specified element prevents it from being added to this list
 * @throws NullPointerException if the specified element is null and this list does not support null elements
 * @throws IllegalArgumentException if some aspect of this element prevents it from being added to this list
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public void add(int index, Object element)
{
    try
    {
        _value = _value.substring(0, index) + verify(element.toString().substring(0, 1)) + _value.substring(index);
    }
    catch (IndexOutOfBoundsException e)
    {
        //element.toString() must have returned a zero-length String
        IllegalArgumentException iae = new IllegalArgumentException(e.getMessage());
        throw (IllegalArgumentException)iae.initCause(e);
    }
    //Provide fail-fast iterators (and list iterators)
    modCount++;
}
/**
 * Inserts all of the elements in the specified collection into this array at the specified position.
 * @param index an int indicating before which element to insert the collection
 * @param c a Collection of elements to append
 * @return a boolean which is true if this array changed as a result of this call
 * @throws IllegalArgumentException if any character to append is not in the 00..7F range
 */
public boolean addAll(int index, Collection c)
{
    boolean modified = false;
    //Presume abnormal termination
    boolean abnormal = true;
    //Preserve the old String and modCount
    String s = _value;
    //Strings are immutable, so although s and _value now reference the same object,
    //changing _value later will change the _value reference away from s
    try
    {
        modified = super.addAll(index, c);
        abnormal = false;
    }
    finally
    {
        if (abnormal)
        {
            //If any of the add() failed, restore the old String and modCount
            _value = s;
            modCount = m;
            modified = false;
        }
    }
    return modified; //If one has a return within a finally, one gets a warning
}
/**
 * Removes all of the elements from this array.
 */
public void clear()
//This implementation is more efficient than AbstractList.clear()
{
    _value = "";
    //Provide fail-fast iterators (and list iterators)
    modCount++;
}

/**
 * Returns true if this array contains the specified element.
 * The method looks for Object.toString().substring(0, 1) in the array.
 * A zero-length String element will return false.
 * @param 0 the Object sought in the array
 * @return a boolean which is true if the array contains
 * o.toString().charAt(0)
 */
public boolean contains(Object o)
//This implementation is more efficient than
AbstractCollection.contains(Object)
//I'm a little surprised this isn't in AbstractList already...
{
    return (indexOf(o) > -1);
}

/**
 * Returns the element (a char) at the specified position in this array.
 * @param index an int specifying the element to return
 * @return the requested Object (a char)
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object get(int index)
{
    return new Character(_value.charAt(index));
}
/**
 * Returns the index in this array of the first occurrence of the specified element,
 * or -1 if this array does not contain this element (zero-length Strings will return -1).
 * @param o the Object whose toString() is sought
 * @return the index of the first occurrence of the specified element (-1 if not found)
 */
public int indexOf(Object o)
{
    if (o.toString().length() <= 0) return -1;
    return _value.indexOf(o.toString().substring(0, 1));
}

/**
 * Returns the index in this array of the last occurrence of the specified element,
 * or -1 if this array does not contain this element (zero-length Strings will return -1).
 * @param o the Object sought
 * @return an int specifying the last occurrence of the specified element (-1 if not found)
 */
public int lastIndexOf(Object o)
{
    if (o.toString().length() <= 0) return -1;
    return _value.lastIndexOf(o.toString().substring(0, 1));
}

/**
 * Removes the element at the specified position in this array and returns it.
 * @param index an int indicating the position from which to remove the element
 * @return the element that was removed from the list
 * @throws UnsupportedOperationException if this method is not supported by this list
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object remove(int index)
{
    Object o = get(index);
    _value = _value.substring(0, index) + _value.substring(index + 1);
    //Provide fail-fast iterators (and list iterators)
    modCount++;
    return o;
}
/**
 * Retains only the elements in this array that are contained in
 * the specified collection.
 * The AbstractCollection implementation clarifies the semantics as
 * follows:
 * One should iterate over the list and, for each element not
 * contained in the collection, remove it.
 * This means all elements that appear at least once in the
 * collection are retained,
 * and that the list retains its original sort order.
 * @param c the Collection of elements to retain from this array
 * @return a boolean which is true if this has changed
 * @throws UnsupportedOperationException if this method is not
 * supported
 * @throws ClassCastException if the elements of the array and
 * collection are incompatible
 * @throws NullPointerException if the specified collection is null
 */
// public boolean
// retainAll(Collection c)
//Provided by AbstractCollection

/**
 * Replaces the element at the specified position in this array
 * with the specified element.
 * @param index an int specifying the index of the element to
 * replace
 * @param element the Object to use in replacing the element
 * @return the element previously at the specified position
 * @throws UnsupportedOperationException if this method is not
 * supported by this list
 * @throws ClassCastException if the class of the specified element
 * prevents it from being added to this list
 * @throws NullPointerException if the specified element is null
 * and this list does not support null elements
 * @throws IndexOutOfBoundsException if some aspect of the specified
 * element prevents it from being added to this list
 * @throws IllegalArgumentException if some aspect of the specified
 * element prevents it from being added to this list
 * @throws IndexOutOfBoundsException if the index is outside the
 * [0..size()]
 */
public Object
set(int index,
Object element)
{
    if (element.toString().length() <= 0) throw new
IllegalArgumentException();
    Object o = get(index);
    _value = _value.substring(0, index) +
verify(element.toString().substring(0, 1)) + _value.substring(index +
1);
    //Note that set() does not modify the array <i>structurally</i>
    return o;
}
/**
 * Returns the number of elements in this array.
 * @return an int specifying the number of elements in this array
 */
public int size()
{
    return _value.length();
}

//java.lang.Object methods
/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {@link java.lang.String} reflecting <code>this</code>
 */
public String toString()
{
    return _value;
}

//HLAdatatype interface implementation
/**
 * Returns the octet boundary of <code>this</code>.
 * The octet boundary value is defined as the smallest power of 2 which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary value of all components within it.
 * For dynamic arrays, octet boundary is thus the largest of the count (an HLAinteger32BE) and the element.
 * @return the octet boundary of <code>this</code>
 */
public final int octetBoundary()
{
    return octetBoundary;
}

//Class-specific extensions
/**
 * Verifies if the String value is acceptable.
 * @param value the String to verify
 * @return the String value
 * @throws IllegalArgumentException if any character in the value is unacceptable
 */
protected abstract String verify(String value)
    throws IllegalArgumentException;
/**
 * Returns the String value of <code>this</code>.
 * @return the String value of <code>this</code>
 * @see #setValue
 */
public String
getValue() {
    return _value;
}

/**
 * Sets the String value of <code>this</code>.
 * @param value the String new value for <code>this</code>
 * @throws IllegalArgumentException if the value isn't Unicode
 * @see #getValue
 */
public void
setValue(String value) {
    _value = verify(value);
    //Provide fail-fast iterators (and list iterators)
    modCount++;
}

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class
getElementClass() {
    return char.class;
}

//end HLAString
// File: HLAASCIIstring.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe ASCII string variable array data type.
 * It uses <code>HLAASCIIchar</code> elements.
 * A Java <code>String</code> is used internally.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.1
 */
public class HLAASCIIstring extends HLAstring
{
    /**
     * Constructs a <code>HLAASCIIstring</code> of default value (empty ASCII string).
     */
    public HLAASCIIstring()
    {
        //super() not called because HLAString is abstract
        setValue("");
    }

    /**
     * Constructs a <code>HLAASCIIstring</code> from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of ASCII characters.
     * @param c a Collection specifying <code>this</code>' value
     * @throws IllegalArgumentException if value isn't ASCII (all characters in numeric range 0..127)
     */
    public HLAASCIIstring(Collection c)
        throws IllegalArgumentException
    {
        this();
        addAll(c);
    }
}
**

* Constructs a `<code>HLAASCIIstring</code>` from the specified String value.
* An exception occurs if any part of value String is not ASCII.
* @param value a String specifying `<code>this</code>`'s value
* @throws IllegalArgumentException if value isn't ASCII (all characters in numeric range 0..127)
*/

public
HLAASCIIstring(String value)
throws IllegalArgumentException
{
    this();
    setValue(value);
}

/**

* Constructs a `<code>HLAASCIIstring</code>` from the specified Object.
* @param o an Object whose toString() specifies `<code>this</code>`
*/

public
HLAASCIIstring(Object o)
{
    this();
    setValue(o.toString());
}

/**

* Creates a `<code>HLAASCIIstring</code>` from the network representation in the provided `<code>byte[]</code>.
* @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAASCIIstring</code>`
* @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
*/

public
HLAASCIIstring(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}
/**
 * Creates a <code>HLAASCIIstring</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAASCIIstring</code>
 * @param offset where in the <code>buffer</code> the <code>HLAASCIIstring</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAASCIIstring(byte[] buffer, int offset) throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAASCIIstring</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAASCIIstring</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAASCIIstring(ByteWrapper byteWrapper) throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}
/**
 * Verifies if the String value is ASCII.
 * @param value the String to verify
 * @return the value String
 * @throws IllegalArgumentException if any character in the value is not in the 00..7F range
 */
protected final String
verify(String value)
throws IllegalArgumentException
{
    for (int i = 0; i < value.length(); i++)
HLAASCIIchar.verify(value.charAt(i));
    //
    // if (value.charAt(i) < 0x0080) continue;
    // if (value.charAt(i) > 0x00FF)
    // {
    //     throw new IllegalArgumentException("[" + i + "] = 0x" +
Integer.toHexString((int)value.charAt(i)));
    // }
    // else if (value.charAt(i) > 0x00FF)
    // {
    //     throw new IllegalArgumentException("[" + i + "] = 0x0" +
Integer.toHexString((int)value.charAt(i)));
    // }
    // else // (value.charAt(i) > 0x007F)
    // {
    //     throw new IllegalArgumentException("[" + i + "] = 0x00" +
Integer.toHexString((int)value.charAt(i)));
    // }
    //
    return value;
}

//List interface implementation

/**
 * Returns the element (an HLAASCIIchar) at the specified position in this array.
 * @param index an int specifying the element to return
 * @return the requested Object (an HLAASCIIchar)
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object
get(int index)
{
    return new HLAASCIIchar(super.get(index));
}
//HLA datatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>. 
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength()
{
    // return HLAinteger32BE.encodedLength + size();
    // return HLAinteger32BE.encodedLength +
    HLAASCIIchar.encodedLength*size();
}

public Class getElementClass()
{
    return HLAASCIIchar.class;
}

//end HLAASCIIstring
package ca.gc.drdc_rddc.hla.rti516.omti;

import hla.rti516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe Unicode string variable array data type.
 * It uses <code>HLAunicodeChar</code> elements.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */

public class HLAunicodeString
    extends HLAstring
{
    /**
     * Constructs a <code>HLAunicodeString</code> of default value
     * (empty Unicode string).
     */
    public HLAunicodeString()
    {
        //super() not called because HLAstring is abstract
        setValue("");
    }

    /**
     * Constructs a <code>HLAunicodeString</code> from the specified
     * Collection.
     * An exception occurs if the Collection doesn't return a series of
     * Unicode characters.
     * @param c a Collection specifying <code>this</code>' value
     * @throws IllegalArgumentException if value isn't Unicode
     */
    public HLAunicodeString(Collection c)
        throws IllegalArgumentException
    {
        //The only reason this isn't in AbstractList already is because of
        the constructor signature
        this();
        addAll(c);
    }
}
/**
 * Constructs a <code>HLAunicodeString</code> from the specified String value.
 * An exception occurs if any part of value String is not Unicode.
 * @param value a String specifying <code>this</code>' value
 * @throws IllegalArgumentException if value isn't Unicode
 */
 public HLAunicodeString(String value)
 throws IllegalArgumentException
 {
 this();
 setValue(value);
 }

 /**
 * Constructs a <code>HLAunicodeString</code> from the specified Object.
 * @param o an Object whose toString() specifies <code>this</code>
 */
 public HLAunicodeString(Object o)
 {
 this();
 setValue(o.toString());
 }

 /**
 * Creates a <code>HLAunicodeString</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAunicodeString</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAunicodeString(byte[] buffer)
 throws CouldNotDecode
 {
 this(buffer, 0);
 }
/**
 * Creates a <code>HLAunicodeString</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAunicodeString</code>
 * @param offset where in the <code>buffer</code> the <code>HLAunicodeString</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public HLAunicodeString(byte[] buffer, int offset)
    throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>HLAunicodeString</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAunicodeString</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public HLAunicodeString(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}

/**
 * Verifies if the String value is acceptable.
 * @param value the String to verify
 * @return the String value
 * @throws IllegalArgumentException if any character in the value is unacceptable
 */

protected String verify(String value)
    throws IllegalArgumentException
{
    //All Java Strings are acceptable because they are Unicode
    return value;
}
// List interface implementation

/**
 * Returns the element (an HLAunicodeChar) at the specified position in this array.
 * @param index an int specifying the element to return
 * @return the requested Object (an HLAunicodeChar)
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object get(int index) {
    return new HLAunicodeChar(super.get(index));
}

// HLADatatype interface implementation

/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength() {
    return HLAinteger32BE.encodedLength() + 2*size();
}

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class getElementClass() {
    return HLAunicodeChar.class;
}

// end HLAunicodeString
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;
import java.util.Iterator;

/**
 * Type-safe variable HLAnull array data type.
 * It uses <code>HLAnull</code> elements and a Java <code>Object[]</code> internally.
 * It is designed as a generic concrete ancestor to various non-basic data type HLA variable arrays.
 * Instead of using the HLAdatatype interface as element class, we used HLAnull,
 * simply in order to make this decodable.
 * The HLAnonunicodeString and HLAnopaqueData classes use String and byte[] respectively,
 * to store HLAnonunicodeChar and HLAnbyte elements, respectively, so their implementations
 * are different in order to be more efficient.
 * Descendant array types need only supply constructors that invoke super,
 * and override List methods when better implementations are possible.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */

public class HLAobjectArray extends HLAvariableArrayType
{
    /**
     * Octet boundary of this class. */
    //The element count is HLAninteger32BE.octetBoundary, but the elements
    //themselves may be wider (e.g. an array of HLAninteger64BE),
    //in which case this field should be redeclared
    public static final int octetBoundary = HLAninteger32BE.octetBoundary;

    /** The array of Objects. */
    protected transient Object[] _values;

    /**
     * Constructs an empty <code>HLAobjectArray</code>.
     */
    public HLAobjectArray()
    {
        //super() not called because super-class is abstract
        _values = new Object[0];
    }
}
/**
 * Constructs an <code>HLAobjectArray</code> from a Collection of Objects.
 * @param c a Collection of Objects
 */
public
HLAobjectArray(Collection c)
{
//super() not called because super-class is abstract
//We could start with an empty _values (by invoking this())
//and then add() using the c.iterator(), but that is relatively
inefficient.
_values = new Object[c.size()];
Iterator it = c.iterator();
for (int i = 0; it.hasNext(); i++)
{
    set(i, it.next());
}
}

/**
 * Constructs an <code>HLAobjectArray</code> containing the specified Object reference.
 * The Object must be of the element class or a descendant.
 * @param o an Object specifying <code>this</code> value
 */
public
HLAobjectArray(Object o)
{
this();
add(o);
}

/**
 * Creates an <code>HLAobjectArray</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAobjectArray</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAobjectArray(byte[] buffer)
    throws CouldNotDecode
{
this(buffer, 0);
}
/**
 * Creates an <code>HLAobjectArray</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAobjectArray</code>
 * @param offset where in the <code>buffer</code> the <code>HLAobjectArray</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAobjectArray(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
 }

 /**
 * Creates an <code>HLAobjectArray</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAobjectArray</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
 public HLAobjectArray(ByteWrapper byteWrapper) throws CouldNotDecode {
    this();
    decode(byteWrapper);
 }

 /**
 * Sets the size of the array to the specified value.
 * Reducing the size will result in the loss of the "chopped off" bytes.
 * @param newSize an int specifying the array’s desired new size
 */
 protected void setSize(int newSize) {
    modCount++;
    int objectsToCopy = _values.length;
    if (newSize < objectsToCopy) objectsToCopy = newSize;
    Object oldValues[] = _values;
    _values = new Object[newSize];
    System.arraycopy(oldValues, 0, _values, 0, objectsToCopy);
}
// List interface implementation

/**
 * Inserts the specified element at the specified position in this array.
 * Elements must be compatible with the array's element data type.
 * @param index an int indicating the position before which to
 * insert the element
 * @param element an Object to insert before index
 * @throws UnsupportedOperationException if this method is not
 * supported by this list
 * @throws ClassCastException if the class of the specified element
 * prevents it from being added to this list
 * @throws NullPointerException if the specified element is null
 * and this list does not support null elements
 * @throws IllegalArgumentException if some aspect of this element
 * prevents it from being added to this list
 * @throws IndexOutOfBoundsException if <code>index</code> is
 * negative or larger than <code>size()</code>
 */
public void add(int index, Object element) {
    if ((index > size()) || (index < 0)) throw new IndexOutOfBoundsException(Integer.toString(index));
    if (element == null) throw new NullPointerException();
    if (!getElementClass().isAssignableFrom(element.getClass())) { //same as:
        if (!getElementClass().isInstance(element)) { //true if element is
            if (!getElementClass().equals(element.getClass())) { //true only
                if classes match exactly
                { //true only
                    throw new ClassCastException(element.getClass().toString());
                }
            }
        }
    }
    setSize(size() + 1); // increments modCount
    System.arraycopy(_values, index, _values, index + 1, size() - (index + 1));
    _values[index] = element;
}


/**
 * Inserts all of the elements in the specified collection into this array at the specified position.
 * @param index an int indicating before which element to insert the collection
 * @param c a Collection of elements to insert
 * @return a boolean which is true if this array changed as a result of this call
 * @throws IllegalArgumentException if some aspect of this element prevents it from being added to this list
 */
public boolean addAll(int index,
Collection c)
{ //This implementation preserves the array if the method ends abnormally
   boolean modified = false;
   //Presume abnormal termination
   boolean abnormal = true;
   //Preserve the old array and modCount
   Object oldValues[] = new Object[size()];
   System.arraycopy(_values, 0, oldValues, 0, size());
   int m = modCount;
   try
   { modified = super.addAll(index, c);
      abnormal = false;
   }
   finally
   { if (abnormal)
   { /*If any of the add() failed, restore the old String and*/
      _values = oldValues;
      modCount = m;
      modified = false;
   }
   return modified; //If one has a return within a finally, one
gets a warning
   }

/**
 * Removes all of the elements from this array.
 */
public void clear()
{ //This implementation is more efficient than AbstractList.clear()
   _values = new Object[0];
   //Provide fail-fast iterators (and list iterators)
   modcount++;
}
/**
 * Returns the element (as an Object) at the specified position in this array.
 * @param index an int specifying the element to return
 * @return the requested Object
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object
get(int index)
{
    return _values[index];
}

/**
 * Removes the element at the specified position in this array and returns it.
 * @param index an int indicating the position from which to remove the element
 * @return the element that was removed from the list
 * @throws UnsupportedOperationException if this method is not supported by this list
 * @throws IndexOutOfBoundsException if <code>index</code> is negative or larger than <code>size()</code>
 */
public Object
remove(int index)
{
    if ((index > size()) || (index < 0)) throw new IndexOutOfBoundsException(Integer.toString(index));
    Object o = get(index);
    System.arraycopy(_values, index + 1, _values, index, size() - (index + 1));
    setSize(size() - 1); // increments modCount
    return o;
}
/**
 * Replaces the element at the specified position in this array with the specified element.
 * The element must be compatible with the element class.
 * @param index an int specifying the index of the element to replace
 * @param element the Object to use in replacing the element
 * @return the element previously at the specified position
 * @throws UnsupportedOperationException if this method is not supported by this list
 * @throws ClassCastException if the class of the specified element prevents it from being added to this list
 * @throws NullPointerException if the specified element is null and this list does not support null elements
 * @throws IllegalArgumentException if some aspect of the specified element prevents it from being added to this list
 * @throws IndexOutOfBoundsException if the index is outside the [0..size()]] range
 */
public Object set(int index, Object element)
{
  if ((index > size()) || (index < 0))
    throw new IndexOutOfBoundsException(Integer.toString(index));
  if (element == null)
    throw new NullPointerException();
  if (!((element instanceof <element class>))
    if (!getElementClass().isAssignableFrom(element.getClass()))
      throw new ClassCastException(element.getClass().toString());
  Object o = get(index);
  _values[index] = element;
  //Note that set() does not modify the array structurally
  return o;
}

/**
 * Returns the number of elements in this array.
 * @return an int specifying the number of elements in this array
 */
public int size()
{
  return _values.length;
}
/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>. 
 *<p>
 * As a general rule, Variant Records [HLAvariantRecord], Dynamic Arrays [HLAvariableArray], and any 
 * Fixed Arrays [HLAfixedArray] or Fixed Records [HLAfixedRecord] containing either of these first two 
 * cannot have a constant encodedLength. 
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public int encodedLength()
{
    //If the element has a fixed size, we can predict the encoded size;
    //such arrays should override this implementation
    HLAdatatype element;
    //Encoding starts with the element count
    int theLength = HLAinteger32BE.encodedLength();
    for (int i = 0; i < size(); i++)
    {
        //Each element is encoded after being aligned to its octetBoundary
        element = (HLAdatatype)get(i);
        while ((theLength % element.octetBoundary()) != 0)
            theLength++;
        theLength += element.encodedLength();
    } //for
    //Note that the last element won't be padded
    return theLength;
}
/**
 * Returns the octet boundary of <code>this</code>.
 * The octet boundary value is defined as the smallest power of 2
 * which is greater than or equal to the size of the datatype in bytes.
 * For a constructed datatype, it is the maximum octet boundary
 * value of all components within it.
 * For dynamic arrays, octet boundary is thus the largest of the
 * count (an HLAinteger32BE) and the element.
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    //If "this" has a static octetBoundary field, use it
    Object o = getThisClassField(this.getClass(), "octetBoundary");
    if (o != null) return ((Integer)o).intValue();
    int cw = HLAinteger32BE.octetBoundary; //element counter's
    octetBoundary
    int ew = 0;
    //Dynamically compare with the element octetBoundary
    o = getThisClassField(getElementClass(), "octetBoundary");
    if (o != null)
    {
        ew = ((Integer)o).intValue();
    } else {
        //Element class did not have a static octetBoundary
        //Must needs invoke an element instance's octetBoundary
        method instead
        if (size() > 0)
        {
            //Use the first element
            o = get(O);
        } else {
            //Construct a throwaway default element
            try
            {
                o = getElementClass().newInstance();
            } catch (Exception e) { //InstantiationException,
                IllegalAccessException
                //Give up!
                return cw;
            }
        }
        ew = ((HLAdatatype)o).octetBoundary();
    }
    return (ew > cw) ? ew : cw;
}
// HLAArrayDatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class getElementClass()
{
    // return HLA.datatype.class; // HLA.datatype is an interface, so it has no constructor -- hence decode fails
    return HLA.null.class;
}

// Supporting classes and methods

/**
 * Returns the specified (static) field's value of the specified Class as an Object.
 * @param aClass the Class whose static field is looked up
 * @param fieldName a String specifying the field's name
 * @return the requested field value, as an Object (null in case of failure)
 */
private Object getClassField(Class aClass, String fieldName)
{
    try {
        return aClass.getField(fieldName).get(null);
    } catch (Exception e) { // NoSuchFieldException, IllegalArgumentException
        return null;
    }
}

// end HLA.objectArray
The MOM array datatypes descend from HLAunicodeString (HLAtransportationName), HLAopaqueData (HLAhandle, HLAlogicalTime, and HLATimeInterval), or HLAobjectArray (the remaining seven). The first four simply declare constructors and do nothing else. The last seven additionally override the getElementClass method (HLAobjectArray does the rest).

```java
// File: HLAtransportationName.java
package ca.gc.drdc.rddc.hla.rtl1516.omti
import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe transportation name (Unicode string) variable array data type.
 * It is essentially identical to <code>HLAunicodeString</code> elements,
 * with an added convenience constructor.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public class HLAtransportationName
extends HLAunicodeString
{
    /**
     * Constructs a <code>HLAtransportationName</code> of default value (empty Unicode string).
     */
    public HLAtransportationName()
    {
        super();
    }

    /**
     * Constructs a <code>HLAtransportationName</code> from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of Unicode characters.
     * @param c a Collection specifying <code>this</code>' value
     * @throws IllegalArgumentException if value isn't Unicode
     */
    public HLAtransportationName(Collection c)
        throws IllegalArgumentException
    {
        super(c);
    }
```
/**
 * Constructs a <code>HLAtransportationName</code> from the
 * specified String value.
 * An exception occurs if any part of value String is not Unicode.
 * @param value a String specifying <code>this</code>' value
 * @throws IllegalArgumentException if value isn't Unicode
 */
 public
 HLAtransportationName(String value)
 throws IllegalArgumentException
 {
     super(value);
 }

 /**
 * Constructs a <code>HLAtransportationName</code> from the
 * specified Object.
 * @param 0 an Object whose toString() specifies <code>this</code>
 * @throws IllegalArgumentException if value isn't Unicode
 */
 public
 HLAtransportationName(Object 0)
 {
     super(0);
 }

 /**
 * Creates a <code>HLAtransportationName</code> from the network
 * representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>HLAtransportationName</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
 HLAtransportationName(byte[] buffer)
 throws CouldNotDecode
 {
     super(buffer);
 }
/**
 * Creates a <code>HLAtransportationName</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAtransportationName</code>
 * @param offset where in the <code>buffer</code> the <code>HLAunicodeString</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAtransportationName(byte[] buffer,
    int offset)
throws CouldNotDecode {
    super(buffer, offset);
}

/**
 * Creates a <code>HLAtransportationName</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAtransportationName</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public
HLAtransportationName(ByteWrapper byteWrapper)
throws CouldNotDecode {
    super(byteWrapper);
}

//end HLAtransportationName
import hla.rti1516.*;  // For CouldNotDecode and the various handle types
import java.util.Collection;
//import java.util.Iterator;

/**
 * Type-safe variable byte array data type, used to store various HLA handle types.
 * It is essentially identical to HLAopaqueData, with added convenience constructors for
 * AttributeHandle, DimensionHandle, FederateHandle, InteractionClassHandle, ObjectClassHandle,
 * ObjectInstanceHandle and ParameterHandle but not MessageRetractionHandle and RegionHandle,
 * which lack the encodedLength and encode methods.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.2
 */

public class HLAhandle
extends HLAopaqueData
{

/**
 * Constructs an empty <code>HLAhandle</code>.
 */
public HLAhandle()
{
    super();
}

/**
 * Constructs an <code>HLAhandle</code> from a Collection of HLAbyte.
 * @param c a Collection of HLAbyte objects
 */
public HLAhandle(Collection c)
{
    super(c);
}
/**
 * Constructs an `<code>HLAHandle</code>` containing the specified byte value.
 * @param value a byte specifying `<code>this</code>`'s value
 */
public HLAHandle(byte value)
{
    super(value);
}

/**
 * Creates an `<code>HLAHandle</code>` from the network representation in the provided `<code>byte[]</code>`.
 * @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAHandle</code>`
 * @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
 */
public HLAHandle(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}

/**
 * Creates an `<code>HLAHandle</code>` from the network representation in the provided `<code>byte[]</code>` at the indicated `<code>offset</code>.
 * @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>HLAHandle</code>`
 * @param offset where in the `<code>buffer</code>` the `<code>HLAHandle</code>` representation begins
 * @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
 */
public HLAHandle(byte[] buffer, int offset)
    throws CouldNotDecode
{
    super(buffer, offset);
}
/**
 * Creates an <code>HLAhandle</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAhandle</code> begins 
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded 
 */
public HLAhandle(ByteWrapper byteWrapper)
throws CouldNotDecode
    super (byteWrapper);

/**
 * Constructs an <code>HLAhandle</code> containing the specified AttributeHandle. 
 * @param value an AttributeHandle specifying <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded 
 */
public HLAhandle(AttributeHandle value)
throws CouldNotDecode
    this ();
    byte[] buffer = new byte[value.encodedLength()];
    value.encode(buffer, 0);
    decode (new ByteWrapper(buffer, 0));

/**
 * Constructs an <code>HLAhandle</code> containing the specified DimensionHandle. 
 * @param value a DimensionHandle specifying <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded 
 */
public HLAhandle(DimensionHandle value)
throws CouldNotDecode
    this ();
    byte[] buffer = new byte[value.encodedLength()];
    value.encode(buffer, 0);
    decode (new ByteWrapper(buffer, 0));
/**
 * Constructs an <code>HLAhandle</code> containing the specified
 * FederateHandle.
 * @param value a FederateHandle specifying <code>this</code>'
 * value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
    HLAhandle(FederateHandle value)
 throws CouldNotDecode
 { }

/**
 * Constructs an <code>HLAhandle</code> containing the specified
 * InteractionClassHandle.
 * @param value an InteractionClassHandle specifying <code>this</code>'
 * value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
    HLAhandle(InteractionClassHandle value)
 throws CouldNotDecode
 { }

/**
 * Constructs an <code>HLAhandle</code> containing the specified
 * ObjectClassHandle.
 * @param value an ObjectClassHandle specifying <code>this</code>'
 * value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
    HLAhandle(ObjectClassHandle value)
 throws CouldNotDecode
 { }
/**
 * Constructs an <code>HLAhandle</code> containing the specified
 * ObjectInstanceHandle.
 * @param value an ObjectInstanceHandle specifying <code>this</code>' value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAhandle(ObjectInstanceHandle value)
 throws CouldNotDecode
 {
     this();
     byte[] buffer = new byte[value.encodedLength()];
     value.encode(buffer, 0);
     decode(new ByteWrapper(buffer, 0));
 }

 /**
 * Constructs an <code>HLAhandle</code> containing the specified
 * ParameterHandle.
 * @param value a ParameterHandle specifying <code>this</code>' value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAhandle(ParameterHandle value)
 throws CouldNotDecode
 {
     this();
     byte[] buffer = new byte[value.encodedLength()];
     value.encode(buffer, 0);
     decode(new ByteWrapper(buffer, 0));
 }
} //end HLAhandle
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.*;
import java.util.Collection;
import java.util.Iterator;

/**
 * Type-safe variable byte array data type, used to store a logical
time.
 * It is essentially identical to HLAopaqueData, with an added
convenience constructor for LogicalTime.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
Valcartier])
 * @version 1.2
 */
public class HLAlogicalTime
    extends HLAopaqueData
{
    /**
     * Octet boundary of this class. */
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;

    /**
     * Constructs an empty <code>HLAlogicalTime</code>.
     */
    public HLAlogicalTime()
    {
        super();
    }

    /**
     * Constructs an <code>HLAlogicalTime</code> containing the
specified byte value.
     * @param value a byte specifying <code>this</code>'s value
     */
    public HLAlogicalTime(byte value)
    {
        super(value);
    }

    /**
     * Constructs an <code>HLAlogicalTime</code> from a Collection of
HLAbyte.
     * @param c a Collection of HLAbyte objects
     */
    public HLAlogicalTime(Collection c)
    {
        super(c);
    }

    /**
     * Constructs an <code>HLAlogicalTime</code> from a Collection of
HLAbyte.
     * @param c a Collection of HLAbyte objects
     */
    public HLAlogicalTime(Collection c)
    {
        super(c);
    }

    /**
     * Constructs an <code>HLAlogicalTime</code> from a Collection of
HLAbyte.
     * @param c a Collection of HLAbyte objects
     */
    public HLAlogicalTime(Collection c)
    {
        super(c);
    }

    /**
     * Constructs an <code>HLAlogicalTime</code> containing the
specified byte value.
     * @param value a byte specifying <code>this</code>'s value
     */
    public HLAlogicalTime(byte value)
    {
        super(value);
    }
}
/**
 * Creates an <code>HLAlogicalTime</code> from the network representation in the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAlogicalTime</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAlogicalTime(byte[] buffer)
throws CouldNotDecode
{
    super(buffer);
}

/**
 * Creates an <code>HLAlogicalTime</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAlogicalTime</code>
 * @param offset where in the <code>buffer</code> the <code>HLAlogicalTime</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAlogicalTime(byte[] buffer,
    int offset)
throws CouldNotDecode
{
    super(buffer, offset);
}

/**
 * Creates an <code>HLAlogicalTime</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAlogicalTime</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public
HLAlogicalTime(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    super(byteWrapper);
}
/**
 * Constructs an <code>HLAlogicalTime</code> containing the
 * specified LogicalTime.
 * @param value a LogicalTime specifying <code>this</code>' value
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */

public HLAlogicalTime(LogicalTime value)
  throws CouldNotDecode
{
  this();
  byte[] buffer = new byte[value.encodedLength()];
  value.encode(buffer, 0);
  decode(new ByteWrapper(buffer, 0));
}

//end HLAlogicalTime
// File: HLAtimeInterval.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.*;
import java.util.Collection;
import java.util.Iterator;

/**
 * Type-safe variable byte array data type, used to store a logical time interval.
 * It is essentially identical to HLAopaqueData, with an added convenience constructor for LogicalTimeInterval.
 * @author Daniel Thibault (Daniel.U.Thibault@DRDC-RDDC.gc.ca)
 * @version 1.2
 */
public class HLAtimeInterval
extends HLAopaqueData
{
    /**
     * Octet boundary of this class.
     */
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;

    /**
     * Constructs an empty <code>HLAtimeInterval</code>.
     */
    public HLAtimeInterval()
    {
        super();
    }

    /**
     * Constructs an <code>HLAtimeInterval</code> from a Collection of HLAByte.
     * @param c a Collection of HLAByte objects
     */
    public HLAtimeInterval(Collection c)
    {
        super(c);
    }

    /**
     * Constructs an <code>HLAtimeInterval</code> containing the specified byte value.
     * @param value a byte specifying <code>this</code>' value
     */
    public HLAtimeInterval(byte value)
    {
        super(value);
    }
}
/**
 * Creates an <code>HLAtimeInterval</code> from the network representation in the provided <code>byte[]</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAtimeInterval</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAtimeInterval(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}

/**
 * Creates an <code>HLAtimeInterval</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAtimeInterval</code>
 * @param offset where in the <code>buffer</code> the <code>HLAtimeInterval</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAtimeInterval(byte[] buffer, int offset)
    throws CouldNotDecode
{
    super(buffer, offset);
}

/**
 * Creates an <code>HLAtimeInterval</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAtimeInterval</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAtimeInterval(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    super(byteWrapper);
}
/**
 * Constructs an <code>HLAtimeInterval</code> containing the specified LogicalTimeInterval.
 * @param value a LogicalTimeInterval specifying <code>this</code>' value
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAtimeInterval(LogicalTimeInterval value)
    throws CouldNotDecode
{
    this();
    byte[] buffer = new byte[value.encodedLength()];
    value.encode(buffer, 0);
    decode(new ByteWrapper(buffer, 0));
}
//end HLAtimeInterval
// File: HLAargumentList.java
package ca.gc.drdrhc.drdc_rtdc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAargumentList variable array data type.
 * It uses <code>HLAunicodeString</code> elements.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public class HLAargumentList
extends HLAobjectArray
{
    /** Octet boundary of this class. */
    //Although not necessary, declaring this is more efficient
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;

    /**
     * Constructs <code>this</code> of default value (empty array).
     */
    public HLAargumentList()
    {
        super();
    }

    /**
     * Constructs <code>this</code> from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of
     * element-compatible objects.
     * @param c a Collection specifying <code>this</code>' value
     */
    public HLAargumentList(Collection c)
    {//The only reason this isn't in AbstractList already is because of the constructor signature
        super(c);
    }
}
/**
 * Constructs <code>this</code> from the specified Object (a single-element array).
 * @param o an element-compatible Object specifying <code>this</code>' first value
 */
public
HLA_argumentList(Object o)
{
 super(o);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLA_argumentList(byte[] buffer)
    throws CouldNotDecode
{
 super(buffer);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLA_argumentList(byte[] buffer,
    int offset)
    throws CouldNotDecode
{
 super(buffer, offset);
}
/**
 * Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAargumentList(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAdatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class getElementClass()
{
    return HLAunicodeString.class;
}

//end HLAargumentList
import hla.rti1516.CouldNotDecode;
import java.util.Collection;
import java.util.Iterator;

public class HLAhandleList
extends HLAobjectArray
{
    /**
     * Octet boundary of this class. */
    //Although not necessary, declaring this is more efficient
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;

    /**
     * Constructs this of default value (empty array).
     */
    public HLAhandleList()
    {
        super();
    }

    /**
     * Constructs this from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of
     * element-compatible objects.
     */
    public HLAhandleList(Collection c)
    {
        //The only reason this isn't in AbstractList already is because of
        //the constructor signature
        super(c);
    }
}
** Constructs <code>this</code> from the specified Object (a single-element array).
* @param o an element-compatible Object specifying <code>this</code>' first value
*/
public
HLAhandleList(Object o)
{
    super(o);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
*/
public
HLAhandleList(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
*/
public
HLAhandleList(byte[] buffer,
        int offset)
    throws CouldNotDecode
{
    super(buffer, offset);
}
constructs this from the supplied ByteWrapper.

:param byteWrapper the ByteWrapper at whose current pos() the representation of this begins

:throws CouldNotDecode if the ByteWrapper could not be decoded

/*
 * HLAhandleList(ByteWrapper byteWrapper)
 * @param byteWrapper the ByteWrapper at whose current pos() the representation of this begins
 * @throws CouldNotDecode if the ByteWrapper could not be decoded
 */
public HLAhandleList(ByteWrapper byteWrapper)
  throws CouldNotDecode
{
  super(byteWrapper);
}

//HLAdatatype interface implementation

/*
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class getElementClass()
{
  return HLAhandle.class;
}

//end HLAhandleList
// File: HLAinteractionCounts.java
package ca.gc.drdc.rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAinteractionCounts variable array data type.
 * It uses <code>HLAinteractionCount</code> elements (themselves fixed
 * records).
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */
public class HLAinteractionCounts
    extends HLAobjectArray
{
    /**
     * Octet boundary of this class.
     */
    // Although not necessary, declaring this is more efficient
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;
    // Fields are HLAnHandle and HLACount (both HLAinteger32BE)

    /**
     * Constructs <code>this</code> of default value (empty array).
     */
    public HLAinteractionCounts()
    {
        super();
    }

    /**
     * Constructs <code>this</code> from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of
     * element-compatible objects.
     * @param c a Collection specifying <code>this</code>' value
     */
    public HLAinteractionCounts(Collection c)
    {//The only reason this isn't in AbstractList already is because of
     * the constructor signature
        super(c);
    }
/**
 * Constructs <code>this</code> from the specified Object (a
 * single-element array).
 * @param 0 an element-compatible Object specifying
 * <code>this</code>' first value
 */
 public
 HLAinteractionCounts(Object o)
 {
   super(o);
 }

 /**
 * Constructs <code>this</code> from the network representation in
 * the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
 HLAinteractionCounts(byte[] buffer)
   throws CouldNotDecode
 {
   super(buffer);
 }

 /**
 * Constructs <code>this</code> from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the
 * representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public
 HLAinteractionCounts(byte[] buffer,
   int offset)
   throws CouldNotDecode
 {
   super(buffer, offset);
 }
/**
 * Constructs <code>this</code> from the supplied
 * <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current
 * <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not
 * be decoded
 */

public HLAinteractionCounts(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAdatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */

public Class getElementClass()
{ return HLAinteractionCount.class; }

} //end HLAInteractionCounts
// File: HLAinteractionSubList.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAinteractionSubList variable array data type.
 * It uses <code>HLAinteractionSubscription</code> elements
 * (themselves fixed records).
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier}}
 * @version 1.1
 */
public class HLAinteractionSubList
    extends HLAobjectArray
{
    /**
     * Octet boundary of this class. */
    //Although not necessary, declaring this is more efficient
    public final static int octetBoundary =
        HLAinteger32BE.octetBoundary;
    //Fields are HLAhandle (HLAinteger32BE) and HLAboolean (likewise)

    /**
     * Constructs <code>this</code> of default value (empty array).
     */
    public HLAinteractionSubList()
    {
        super();
    }

    /**
     * Constructs <code>this</code> from the specified Collection.
     * An exception occurs if the Collection doesn’t return a series of
     * element-compatible objects.
     * @param c a Collection specifying <code>this</code>' value
     */
    public HLAinteractionSubList(Collection c)
    //The only reason this isn’t in AbstractList already is because of
    the constructor signature
    {
        super(c);
    }
/**
 * Constructs <code>this</code> from the specified Object (a single-element array).
 * @param o an element-compatible Object specifying <code>this</code>’ first value
 */
public
HLAInteractionSubList(Object o)
{
    super(o);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAInteractionSubList(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAInteractionSubList(byte[] buffer, int offset)
    throws CouldNotDecode
{
    super(buffer, offset);
}
**
* Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.  
* @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins  
* @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
*/
public
HLAinteractionSubList(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAdatatype interface implementation

/**
* Returns the Class of the array's elements.  
* @return the Class of the array's elements  
*/
public Class
getElementClass()
{
    return HLAinteractionSubscription.class;
}

//end HLAinteractionSubList
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAobjectClassBasedCounts variable array data type.
 * It uses <code>HLAobjectClassBasedCount</code> elements (themselves
 * fixed records).
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier])
 * @version 1.1
 */
public class HLAobjectClassBasedCounts
        extends HLAobjectArray
{
    /**
     * Octet boundary of this class. */
    //Although not necessary, declaring this is more efficient
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;
    //Fields are HLAHandle and HLACount (both HLAinteger32BE)

    /**
     * Constructs <code>this</code> of default value (empty array).
     */
    public HLAobjectClassBasedCounts()
    {
        super();
    }

    /**
     * Constructs <code>this</code> from the specified Collection.
     * An exception occurs if the Collection doesn't return a series of
     * element-compatible objects.
     * @param c a Collection specifying <code>this</code>' value
     */
    public HLAobjectClassBasedCounts(Collection c)
    {  
        super(c);
    }
}
public HLAobjectClassBasedCounts(Object o)
{
    super(o);
}

public HLAobjectClassBasedCounts(byte[] buffer)
throws CouldNotDecode
{
    super(buffer);
}

public HLAobjectClassBasedCounts(byte[] buffer,
int offset)
throws CouldNotDecode
{
    super(buffer, offset);
}
/**
 * Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAobjectClassBasedCounts(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAdatatype interface implementation
/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */
public Class getElementClass()
{
    return HLAobjectClassBasedCount.class;
}

//end HLAobjectClassBasedCounts
// File: HLAsyncPointFederateList.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAsyncPointFederateList variable array data type.
 * It uses <code>HLAsyncPointFederate</code> elements (themselves fixed records).
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.1
 */
public class HLAsyncPointFederateList extends HLAobjectArray {

  /**
   * Octet boundary of this class. */
  /**
   * Although not necessary, declaring this is more efficient
   * public final static int octetBoundary = HLAinteger32BE.octetBoundary;
   */
  //Fields are HLAhandle (HLAinteger32BE) and HLAsyncPointStatus
  // (HLAinteger32BE enumerated)

  /**
   * Constructs <code>this</code> of default value (empty array).
   */
  public HLAsyncPointFederateList() {
    super();
  }

  /**
   * Constructs <code>this</code> from the specified Collection.
   * An exception occurs if the Collection doesn't return a series of
   * element-compatible objects.
   * @param c a Collection specifying <code>this</code>' value
   */
  public HLAsyncPointFederateList(Collection c) {
    super(c);
  }
}
/**
 * Constructs <code>this</code> from the specified Object (a single-element array).
 * @param o an element-compatible Object specifying <code>this</code>' first value
 */
public HLAsyncPointFederateList(Object o)
{
    super(o);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAsyncPointFederateList(byte[] buffer) throws CouldNotDecode
{
    super(buffer);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAsyncPointFederateList(byte[] buffer, int offset) throws CouldNotDecode
{
    super(buffer, offset);
}
/**
 * Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public HLAAsyncPointFederateList(ByteWrapper byteWrapper)
 throws CouldNotDecode
{
 super(byteWrapper);

// HLADatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */

public Class getElementClass()
{
 return HLAAsyncPointFederate.class;
}

// end HLAAsyncPointFederateList
// File: HLAAsyncPointList.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;

/**
 * Type-safe MOM HLAAsyncPointList variable array data type.
 * It uses <code>HLAunicodeString</code> elements.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier])
 * @version 1.1
 */
public class HLAAsyncPointList
  extends HLAobjectArray
{
  /** Octet boundary of this class. */
  // Although not necessary, declaring this is more efficient
  public final static int octetBoundary = HLAinteger32BE.octetBoundary;

  /**
   * Constructs <code>this</code> of default value (empty array).
   */
  public HLAAsyncPointList()
  {
    super();
  }

  /**
   * Constructs <code>this</code> from the specified Collection.
   * An exception occurs if the Collection doesn't return a series of
   * element-compatible objects.
   * @param c a Collection specifying <code>this</code>' value
   */
  public HLAAsyncPointList(Collection c)
  {
    super(c);
  }
/**
 * Constructs <code>this</code> from the specified Object (a single-element array).
 * @param o an element-compatible Object specifying <code>this</code>' first value
 */
public HLAsyncPointList(Object o)
{
    super(o);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public HLAsyncPointList(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}

/**
 * Constructs <code>this</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code> 
 * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins 
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded 
 */
public HLAsyncPointList(byte[] buffer, int offset)
    throws CouldNotDecode
{
    super(buffer, offset);
}
/**
 * Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public HLAsyncPointList(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAdatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */

public Class getElementClass()
{
    return HLAunicodeString.class;
}

//end HLAsyncPointList
The HLAfixedrecorddatatype interface combines HLA datatype with Java’s Map. The HLAfixedRecordType abstract class extends Java’s AbstractMap.

```java
// File: HLAfixedrecorddatatype.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import java.util.Iterator;

/**
 * Interface implemented by the various HLA fixed record data types.
 * In addition to this interface, the HLA fixed record data type classes are expected to supply:
 * <ul>
 * <li> Constructors (default and specified-value)</li>
 * <li> Constructor (byte[] buffer) throws CouldNotDecode</li>
 * <li> Constructor (byte[] buffer, int offset) throws CouldNotDecode</li>
 * <li> Constructor (ByteWrapper byteWrapper) throws CouldNotDecode</li>
 * <li> java.lang.Object methods toString(); equals(Object otherObject) and hashCode()</li>
 * <li> Class-specific extensions to get and set the value as a basic java data type (int, boolean, etc.)</li>
 * </ul>
 * A fixed record is a finite unvarying sequence of fields (at the class level).
 * Each fixed record field has a Name and a Class.
 * We'll need two Iterators over the fields (one for the names, another for the classes), get/set methods
 * indexed by Name for each field.
 * The octetBoundary is static for fixed records, being equal to the largest octetBoundary of the record's fields.
 * The encodedLength will obviously depend on the total encodedLengths of the various fields
 * (some of which may be dynamic arrays, for example).
 */

public interface HLAfixedrecorddatatype
    extends java.util.Map,
    java.io.Serializable,
    java.lang.Cloneable,
    HLA datatype {

    /**
     * Returns an Iterator over the record's fields (as HLA datatypes).
     * @return an Iterator over the record's fields (as HLA datatypes)
     */
    Iterator iterator();
```
/**
 * Returns an Iterator over the record's fields' names (as Strings).
 * @return an Iterator over the record's fields' names (as Strings)
 */
Iterator
  nameIterator();
}
//end HLAfixedrecorddatatype
package ca.gc.drdc_rddc.hla.rti1516.omti;

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.ArrayList;
import java.lang.reflect.Field;

/**
 * Iterator implementation for HLA fixed record types.
 * @author Daniel Thibault (Daniel.Thibault@DRDC-RDDC.gc.ca DRDC Valcartier)
 * @version 1.1
 */
public class HLAfixedRecordIterator implements Iterator {

  /**
   * The enumerators over which to iterate.
   */
  private Object enumerators[];

  /**
   * Index of enumerator to be returned by subsequent call to next().
   */
  private int cursor;

  /**
   * Constructs the <code>Iterator</code> from an instance reference.
   * If the <code>boolean asStrings</code> is false, the <code>Iterator</code>
   * will enumerate the <code>Object</code>s; otherwise it will enumerate
   * the field names as <code>String</code>s.
   * Note that unlike HLAenumeratedIterator, the argument must be an
   * instance (instead of a Class reference) because the actual field
   * values must be enumerated.
   * Examples: <code>Iterator itr = new HLAfixedRecordIterator(this,
   * false);</code>
   * <code>Iterator nameItr = new HLAfixedRecordIterator(this,
   * true);</code>
   * @param theInstance the instance whose public fields are to be
   * enumerated
   * @param asStrings a boolean which is false if the enumeration
   * should supply Objects, true if it should supply the field names (as
   * Strings) instead
   * @throws ClassCastException if theInstance doesn’t implement the
   * HLAfixedRecorddatatype interface (or an IllegalAccessException
   * occurred internally)
   */
}
public
HLAfixedRecordIterator(Object theInstance, boolean asStrings)
throws ClassCastException
{
  if (!HLAfixedrecorddatatype.class.isInstance(theInstance)) throw new ClassCastException(theInstance.getClass().toString());
  cursor = 0;
  ArrayList al = new ArrayList();
  // Obtain the object's accessible public fields
  Field[] theFields = theInstance.getClass().getFields();
  for (int i = 0; i < theFields.length; i++)
  {
    // Skip fields which do not implement the HLA datatype interface
    // To simplify the constructor, the IllegalAccessException that this may throw is wrapped in a ClassCastException
    try
    {
      if (HLAdatatype.class.isInstance(theFields[i].get(theInstance)))
      {
        if (asStrings)
        {
          al.add(theFields[i].getName());
        }
        else
        {
          al.add(theFields[i].get(theInstance));
        }
      }
      catch (IllegalAccessException e)
      {
        ClassCastException cce = new ClassCastException(e.getMessage());
        throw (ClassCastException)cce.initCause(e);
      }
    }
    al.trimToSize();
    enumerators = al.toArray();
  }
}
/**
   * Constructs the <code>Iterator</code> from an instance reference, returning Map.Entry elements.
   * <p>
   * HLANullFixedRecord.entrySet().iterator() uses this constructor.
   * @param theInstance the instance whose public fields are to be enumerated
   * @throws ClassCastException if theInstance doesn't implement the HLAfixedRecordDatatype interface (or an IllegalAccessException occurred internally)
   */
   public HLAfixedRecordIterator(Object theInstance)
   throws ClassCastException
   {
     if (!HLAfixedRecordDatatype.class.isInstance(theInstance)) throw
     new ClassCastException(theInstance.getClass().toString());
     cursor = 0;
     ArrayList al = new ArrayList();
     //Obtain the object's accessible public fields
     Field[] theFields = theInstance.getClass().getFields();
     for (int i = 0; i < theFields.length; i++)
     {
       //Skip fields which do not implement the HLAdatatype interface
       //To simplify the constructor, the IllegalAccessException that this may throw is wrapped in a ClassCastException
       try
       {
         if (HLAdatatype.class.isInstance(theFields[i].get(theInstance)))
           {
             al.add(new HLAfixedRecordIterator.HLAfixedRecordMapEntry(theFields[i].getName(), (HLAfixedRecordDatatype)theInstance));
           }
         catch (IllegalAccessException e)
         {
           ClassCastException cce = new ClassCastException(e.getMessage());
           throw (ClassCastException) cce.initCause(e);
         }
       }
     }
     al.trimToSize();
     enumerators = al.toArray();
   }
//java.util.Map.Entry implementation

/**
 * The Map record class.
 * We enforce the expected types in the constructor and use the
 * underlying Object as the backing Map.
 */
static class HLAfixedRecordMapEntry
    implements java.util.Map.Entry
{
    //The name of the field
    String key;

    //The instance whose field it is {note that the field value
    isn't stored}
    HLAfixedrecorddatatype instance;

    /**
     * Constructs a new Map.Entry from the specified String key and
     * HLAfixedrecorddatatype instance.
     * @param key a String specifying the name of the instance's
     * field
     * @param instance an HLAfixedrecorddatatype interface whose
     * field is being mapped
     * @throws IllegalArgumentException if the key or instance is
     * <code>null</code>
     */
    public HLAfixedRecordMapEntry(String key,
                                   HLAfixedrecorddatatype instance)
        throws IllegalArgumentException
    {
        if ((key == null) || (instance == null)) throw new
IllegalArgumentException();
        this.key = key;
        this.instance = instance;
    }

    /**
     * Returns the key {a String field name} corresponding to this
     * entry.
     * @return the key corresponding to this entry
     */
    public Object getKey()
    {
        return (Object)key;
    }
}
/**
 * Returns the value (an HLAfixedrecorddatatype instance) corresponding to this entry.
 * We return the actual value stored in the backing Object.
 * Because Map.Entry does not allow this method to throw any exceptions,
 * should one occur we return <code>null</code> instead.
 * @return the value corresponding to this entry (or <code>null</code> if an exception occurred internally)
 */
public Object getValue()
{
    try
    {
        return instance.getClass().getField(key).get(instance);
    }
    catch (Exception e)
    {
        return null;
    }
}
/**
 * Replaces the value corresponding to this entry with the specified value.
 * Writes through to the Map.
 * The behaviour of this call is undefined if the mapping has already been removed from the map
 * (by the iterator's remove operation).
 * @param value an Object (HLAfixedrecorddatatype instance) to store in this entry
 * @return the old value that was stored in this entry
 * @throws UnsupportedOperationException if the operation is not supported by the backing Map
 * @throws ClassCastException if the class of the specified value prevents it from being stored in the backing Map
 * @throws IllegalArgumentException if some aspect of this value prevents it from being stored in the backing Map
 * @throws NullPointerException if the backing Map does not permit null values and the specified value is null
 */
public Object setValue(Object value) {
  //Likewise, here we set the actual value, not the stored instance reference
  if (value == null) throw new NullPointerException();
  //We wrap the other exceptions in an IllegalArgumentException
  try {
    Object oldValue = instance.getClass().getField(key).get(instance);
    instance.getClass().getField(key).set(instance, value);
    return oldValue;
  }
  catch (Exception e) //NoSuchFieldException, IllegalAccessException
  {
    IllegalArgumentException iae = new IllegalArgumentException(e.getMessage());
    throw (IllegalArgumentException)iae.initCause(e);
  }
}
/**
 * Returns true iff <code>this</code> and <code>otherObject</code> represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code> is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object o) {
    if (! (o instanceof java.util.Map.Entry)) return false;
    java.util.Map.Entry e = (java.util.Map.Entry)o;
    return key.equals(e.getKey()) && getValue().equals(e.getValue());
}

/**
 * Returns a hash code for <code>this</code>; two objects for which <code>equals()</code> is <code>true</code> should yield the same hash code.
 * @return an <code>int</code> hash code
 * @see Object#equals(java.lang.Object) Object.equals(java.lang.Object)
 * @see java.util.Hashtable Hashtable
 */
public int hashCode() {
    return key.hashCode() ^ getValue().hashCode(); //bitwise XOR
}

/**
 * Returns a <code>String</code> representation of <code>this</code>.
 * @return a {link java.lang.String} reflecting <code>this</code> value
 */
public String toString() {
    return key + "=" + getValue().toString();
}
}
//Iterator implementation

/**
 * Returns <code>true</code> if the iteration has more elements.
 * In other words, returns <code>true</code> if <code>next()</code>
 * would return an element rather than throwing an exception.
 * @return a boolean which is <code>true</code> if the iterator has
 * more elements
 */
public boolean hasNext()
{
    return cursor < enumerators.length;
}

/**
 * Returns the next element in the iteration.
 * @return the next element in the iteration
 * @throws NoSuchElementException if the iteration has no more
 * elements
 */
public Object next()
{
    if (hasNext()) return enumerators[cursor++];
    throw new NoSuchElementException();
}

/**
 * Removes from the underlying collection the last element returned
 * by the iterator (optional operation).
 * This method can be called only once per call to
 * <code>next()</code>. The behaviour of an <code>Iterator</code> is unspecified if the
 * underlying collection is modified
 * while the iteration is in progress in any way other than by
 * calling this method.
 * @throws UnsupportedOperationException if this operation is not
 * supported by this Iterator
 * @throws IllegalStateException if <code>next()</code> has not yet
 * been called, or <code>remove()</code> has already been called since
 * the last call to <code>next()</code>.
 */
public void remove()
{
    throw new UnsupportedOperationException();
}

//end HLAfixedRecordIterator
// File: HLAfixedRecordType.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Set;
import java.util.Iterator;
import java.util.AbstractSet;
import java.lang.reflect.Field;

/**
 * Abstract implementation of the HLA fixed record type.
 * User-defined HLA fixed record types can be quickly defined as extensions of this class,
 * defining the public fields and the constructors --no other code needed.
 * Note that this class is field-less.
 * <p>
 * The <code>Map</code> is specified as mapping String keys (representing the field names) to their values
 * <code>HLAdatatype</code> implementations).
 * The map <code>size()</code> will be unchanging, since the record will always have the same fields in the same order.
 * Values may not be <code>null</code> since they all must be HLA data types (i.e. all references will exist).
 * <p>
 * The <code>put</code> method behaves exactly as direct access; that is to say,
 * <code>put("fieldName", someObject)</code> is equivalent to <code>fieldName = someObject</code>.
 * The <code>remove</code> method throws an <code>UnsupportedOperationException</code>.
 * <p>
 * The helper class <code>HLAfixedRecordIterator</code> is used to generate the various <code>Iterator</code>s.
 * The <code>Map</code> interface recommends supplying a void constructor and a <code>Map</code> constructor.
 * See below for constructor templates.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public abstract class HLAfixedRecordType
    extends java.util.AbstractMap
    implements HLAfixedRecordDataType
{
    //If at all possible, it is strongly recommended that the concrete class declare this field:
    // public static final int octetBoundary = -<whatever>;
    //The fixed record's fields should be declared here, in order.
    //only those accessible public fields which implement the HLAdatatype interface will be considered.
    //This abstract class declares no such fields.
/**
 * Constructs a <code>HLAfixedRecordType</code> of default values.
 */
public // HLAfixedRecordType()
// throws CouldNotDecode
{
try
{
    initializeFields();
    catch (InstantiationException e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

/***
 * Creates a <code>HLAfixedRecordType</code> from the network
 representation in the provided <code>byte[]</code>.  
 * @param buffer the network-provided <code>byte[]</code> 
 representation of the <code>HLAfixedRecordType</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be 
 decoded
 */
public // HLAfixedRecordType(byte[] buffer)
// throws CouldNotDecode
{
    this(buffer, 0);
}

/***
 * Creates a <code>HLAfixedRecordType</code> from the network
 representation in the provided <code>byte[]</code> at the indicated 
<code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> 
 representation of the <code>HLAfixedRecordType</code>
 * @param offset where in the <code>buffer</code> the 
<code>HLAfixedRecordType</code> representation begins 
 * @throws CouldNotDecode if the <code>buffer</code> could not be 
 decoded
 */
public // HLAfixedRecordType(byte[] buffer, 
    // int offset)
// throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a <code>HLAfixedRecordType</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAfixedRecordType</code> begins 
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded 
 */
public HLAfixedRecordType(ByteWrapper byteWrapper)
  throws CouldNotDecode
  {
    this();
    decode(byteWrapper);
  }

/**
 * Creates a <code>HLAfixedRecordType</code> from the provided <code>Map</code>. 
 * @param theMap the <code>Map</code> representation of the <code>HLAfixedRecordType</code> 
 * @throws CouldNotDecode if the <code>Map</code> could not be decoded 
 */
public HLAfixedRecordType(java.util.Map theMap)
  throws CouldNotDecode
  {
    this();
    // Wrap all exceptions as CouldNotDecode
    try { putAll(theMap); }
    catch (Exception e)
    {
      CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
      throw (CouldNotDecode)cnd.initCause(e);
    }
  }
/**
 * Calls the void constructor on each of the HLAdatatype fields.
 * @throws InstantiationException if the class to instantiate is
 * abstract or an interface (or some other problem occurs)
 */
protected void initializeFields()
    throws InstantiationException
{
    //Obtain the object's accessible public fields
    Field[] theFields = this.getClass().getFields();
    for (int i = 0; i < theFields.length; i++)
    {
        //We're looking for fields which implement the HLAdatatype
        interface
        // I first thought of scanning getInterfaces() for
        HLAdatatype.class, but
        // that method only returns the interfaces *immediately*
        implemented by the
        // class --it does not fetch the inherited interfaces!
        if (HLAdatatype.class.isAssignableFrom(theFields[i].getType()))
        {
            //To simplify exception handling, we'll wrap the
            IllegalAccessExcetion in an InstantiationException
            try
            {
                theFields[i].set(this,
                theFields[i].getType().newInstance());
            }
            catch (IllegalAccessException e)
            {
                InstantiationException ie = new
                InstantiationException(e.getMessage());
                throw (InstantiationException)ie.initCause(e);
            }
        }
    }
}
/**
 * Associates the specified value with the specified key in this map (optional operation).
 * If the map previously contained a mapping for this key, the old value is replaced.
 * @param key the Object key with which the specified value is to be associated
 * @param value the Object value to be associated with the specified key
 * @return previous value associated with the specified key, or <tt>null</tt> if there was no mapping for the key.
 *         A <tt>null</tt> return can also indicate that the map previously associated <tt>null</tt> with the specified key, if the implementation supports <tt>null</tt> values.
 * @throws UnsupportedOperationException if the operation is not supported by this map
 * @throws ClassCastException if the class of the specified key or value prevents it from being stored in this map
 * @throws IllegalArgumentException if some aspect of this key or value prevents it from being stored in this map
 * @throws NullPointerException if this map does not permit <tt>null</tt> keys or values, and the specified key or value is <tt>null</tt>
 */
public Object put (Object key, Object value)
{
    try //We'll wrap the other exceptions in
    //We'll wrap the other exceptions in
    public Object put(Object key, Object value)
    {
        try //We'll wrap the other exceptions in
        //We'll wrap the other exceptions in
        try //We'll wrap the other exceptions in
        //We'll wrap the other exceptions in
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
        }
        catch (Exception e) //NoSuchFieldException,
/**
 * Removes the mapping for this key from this map if present
 * (optional operation).
 * Since the underlying Iterator does not allow remove(), neither
 * does this implementation.
 * @param key the Object key whose mapping is to be removed from
 * the map
 * @return the previous value associated with the specified key, or
 * <tt>null</tt> if there was no entry for the key.
 * A <tt>null</tt> return can also indicate that the map
 * previously associated
 * <tt>null</tt> with the specified key, if the
 * implementation supports <tt>null</tt> values.
 * @throws UnsupportedOperationException if the operation is not
 * supported by this map
 */
public Object
remove(Object key)
{
throw new UnsupportedOperationException();
}

//Map implementation: Views
/**
 * This field is initialized to contain an instance of the Set the
 * first time it is requested.
 * The view is stateless, so there’s no reason to create more than
 * one.
 */
transient volatile Set entrySet = null;

/**
 * Returns a Set view of the mappings contained in this map.
 * Each element in this set is a Map.Entry.
 * The set is backed by the map, so changes to the map are
 * reflected in the set, and vice-versa.
 * If the map is modified while an iteration over the set is in
 * progress, the results of the iteration are undefined.
 * The set supports element removal, which removes the
 * corresponding entry from the map, via the
 * <tt>Iterator.remove</tt>, <tt>Set.remove</tt>,
 * <tt>removeAll</tt>, <tt>retainAll</tt> and
 * <tt>clear</tt> operations. It does not support the <tt>add</tt>
 * or <tt>addAll</tt> operations.
 * @return a Set view of the mappings contained in this map.
 */
public Set
entrySet()
{
if (entrySet == null)
{
    //Most of the Set implementation is supplied by
    AbstractCollection and AbstractSet.
    //By default add(Object) throws
    UnsupportedOperationException, which is fine for our purpose.
entrySet = new AbstractSet() {
    transient volatile boolean sizeKnown = false;
    transient volatile int size = 0;

    /**
     * Returns an Iterator over the elements in this Set.
     * The elements are returned in no particular order
     * unless this Set is an instance of some class that
     * provides such a guarantee.
     * @return an Iterator over the elements in this Set
     */
    public Iterator iterator() {
        //We're not allowed to throw anything here
        try {
            return new HLAfixedRecordIterator(HLAfixedRecordType.this);
        }
        catch (ClassCastException iae) { return null; }
    }

    //AbstractMap.size() defers to this method
    /**
     * Returns the number of elements in this Set (its
     * cardinality).
     * If this Set contains more than Integer.MAX_VALUE
     * elements, returns Integer.MAX_VALUE.
     * @return the number of elements in this Set
     */
    public int size() {
        //First time we're asked?
        if (!sizeKnown) {
            Iterator i = iterator();
            while (i.hasNext()) {
                size++;
                i.next();
            }
            sizeKnown = true;
        }
        return size;
    }
}; //end of anonymous inner class
return entrySet;
//HLAfixedRecord datatype implementation

/**
 * Returns an Iterator over the record's fields (as HLADatatypes).
 * @return an Iterator over the record's fields (as HLADatatypes)
 */
public Iterator iterator()
{
    // We're not allowed to throw anything here
    try
    {
        return new HLAfixedRecordIterator(this, false);
    }
    catch (ClassCastException iae) { return null; }
}

/**
 * Returns an Iterator over the record's fields' names (as Strings).
 * @return an Iterator over the record's fields' names (as Strings)
 */
public Iterator nameIterator()
{
    // We're not allowed to throw anything here
    try
    {
        return new HLAfixedRecordIterator(this, true);
    }
    catch (ClassCastException iae) { return null; }
}

//java.lang.Object methods

//java.lang.Object.toString supplied by AbstractMap
//equals, as supplied by AbstractMap, isn't quite satisfactory as
//it does not ensure
//the field sequences are the same, or that the other object
descends from this class
/**
 * Returns true iff <code>this</code> and <code>otherObject</code>
represent the same object.
 * @param otherObject the <code>Object</code> to compare with
 * @return <code>true</code> iff supplied <code>otherObject</code>
is of the same type as <code>this</code> and has the same value
 * @see Object#hashCode Object.hashCode()
 * @see java.util.Hashtable Hashtable
 */
public boolean equals(Object otherObject)
{
    if (this == otherObject) return true;
    //This form is more inheritable, and will ensure that
    otherObject is an instance of this' class
    if (!this.getClass().equals(otherObject.getClass())) return false;
    //Iterate over the fields; they should be in lock-step
    Iterator itr = iterator();
    Iterator itrOther = ((HLAfixedRecordType) otherObject).iterator();
    while (itr.hasNext() && itrOther.hasNext())
    {
        if (!itr.next().equals(itrOther.next())) return false;
    }
    return !(itr.hasNext() || itrOther.hasNext());
}

//java.lang.Object.hashCode is supplied by AbstractMap

//HLA datatype interface implementation

//Several of these methods are repeats from HLAbasicType (because
we can't inherit from both it and AbstractMap)
/**
 * Returns the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>.  
 * @return the length (in bytes) of the <code>byte[]</code> representation of <code>this</code>
 */
public final int
encodedLength()
{
    HLA datatype field;
    int theLength = 0;
    //Iterate over the fields
    Iterator itr = iterator();
    while (itr.hasNext())
    {
        field = (HLA datatype)itr.next();
        while ((theLength % field.octetBoundary()) != 0) theLength++;
        theLength += field.encodedLength();
    }
    return theLength;
}

/**
 * Returns the octet boundary of <code>this</code>.  
 * The octet boundary value is defined as the smallest power of 2 which is greater than or equal to the size of the datatype in bytes.  
 * For a constructed datatype, it is the maximum octet boundary value of all components within it.  
 * @return the octet boundary of <code>this</code>
 */
public final int
octetBoundary()
{
    Object o = getThisField("octetBoundary");
    if (o != null) return ((Integer)o).intValue();
    int fieldBoundary;
    int theBoundary = 1;
    //Iterate over the fields
    Iterator itr = iterator();
    while (itr.hasNext())
    {
        fieldBoundary = ((HLA datatype)itr.next()).octetBoundary();
        if (fieldBoundary > theBoundary) theBoundary = fieldBoundary;
    }
    return theBoundary;
}
/**
 * Encodes <code>this</code> into the <code>byte[]</code> at the
 * specified <code>offset</code>.
 * @param buffer the <code>byte[]</code> into which to encode <code>this</code>
 * @param offset the offset into the <code>byte[]</code> at which to encode <code>this</code>
 * @return how many bytes were written to the buffer, including any prefix padding bytes
 */
public int
encode(byte[] buffer,
int offset)
{
    return encode(new ByteWrapper(buffer, offset)).pos() - offset;
}

/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
encode(ByteWrapper byteWrapper)
{
    byteWrapper.align(octetBoundary());
    //Iterate over the fields
    Iterator itr = iterator();
    while (itr.hasNext())
        ((HLAdatatype)itr.next()).encode(byteWrapper);
    return byteWrapper;
}

/**
 * Encodes <code>this</code> into a new <code>byte[]</code>.
 * @return a <code>byte[]</code> encoding <code>this</code>
 */
public byte[]
toByteArray()
{
    return encode(new ByteWrapper(encodedLength())).array();
}
/**
 * Sets <code>this</code> value from the network representation in the provided <code>byte[]</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @return how many bytes were read from the <code>byte[]</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 * 
 * public int
 * decode(byte[] buffer)
 * throws CouldNotDecode
 * {
 * return decode(buffer, 0);
 * }
 */

/**
 * Sets <code>this</code> value from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>. 
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> <code>this</code> representation begins
 * @return where in the <code>buffer</code> <code>this</code> representation ends
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public int
decode(byte[] buffer, 
int offset)
throws CouldNotDecode
{
try
{
return decode(new ByteWrapper(buffer, offset)).pos();
}
catch (Exception e)
{
CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
throw (CouldNotDecode)cnd.initCause(e);
}
}
/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper decode(ByteWrapper byteWrapper) throws CouldNotDecode {
    byteWrapper.align(octetBoundary());
    // Iterate over the fields
    Iterator itr = iterator();
    while (itr.hasNext()) {
        ((HLA datatype)itr.next()).decode(byteWrapper);
    }
    return byteWrapper;
}

// Cloneable implementation

/**
 * Creates and returns an independent copy of this object.
 * @return an independent copy of this Object
 * @throws CloneNotSupportedException if the object's class is not Cloneable or if the instance cannot be cloned
 */
public Object clone() throws CloneNotSupportedException {
    try {
        return new HLA fixedRecordType(toByteArray());
        return this.getClass().getConstructor(new Class[] { byte[].class }).newInstance(new Object[] { this.toByteArray() });
    }
    // Something is seriously wrong with the class if it can't encode and then decode itself...
    catch (Exception e) // CouldNotDecode
    {
        CloneNotSupportedException cnse = new CloneNotSupportedException(e.getMessage());
        throw (CloneNotSupportedException)cnse.initCause(e);
    }
}
//Supporting classes and methods

/**
 * Returns the specified (static) field's value as an Object.
 * This is just an instance-proxy for the various class fields.
 * @param fieldName a String specifying the field's name
 * @return the requested field, as an Object (null in case of failure)
 */
private Object
    getThisField(String fieldName) {
    try {
        return this.getClass().getField(fieldName).get(null);
    }
    catch (Exception e) //NoSuchFieldException, IllegalAccessException
        return null;
    }
} //end HLAfixedRecordType
The MOM’s fixed record types (HLAinteractionCount, HLAinteractionSubscription, HLAobjectClassBasedCount, and HLASyncPointFederate) serve as demonstrations of the HLAfixedRecordType design pattern.

```java
// File: HLAinteractionCount.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe MOM HLAinteractionCount fixed record data type.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier}}
 * @version 1.1
 */
public class HLAinteractionCount
    extends HLAfixedRecordType
{
    /** First field of <code>HLAinteractionCount</code>: an HLAhandle. */
    public HLAhandle HLAinteractionClass;
    /** Second field of <code>HLAinteractionCount</code>: an HLAcount. */
    public HLAcount HLAinteractionCount;

    /**
     * Constructs <code>this</code> with default values.
     * @throws CouldNotDecode if an {@link InstantiationException} occurs
     */
    public HLAinteractionCount()
    throws CouldNotDecode
    { try
        { initializeFields();
        } catch (InstantiationException e)
        { CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
            throw (CouldNotDecode)cnd.initCause(e);
        }
    }
```

DRDC Valcartier TR 2007-412 701
/**
   * Constructs <code>this</code> from the network representation in
   * the provided <code>byte[]</code>.  
   * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>  
   * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
   */
  public HLAinteractionCount(byte[] buffer)  
  throws CouldNotDecode  
  {  
    this(buffer, 0);  
  }

/**
   * Constructs <code>this</code> from the network representation in
   * the provided <code>byte[]</code> at the indicated <code>offset</code>.  
   * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>  
   * @param offset where in the <code>buffer</code> the representation of <code>this</code> begins  
   * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
   */
  public HLAinteractionCount(byte[] buffer, int offset)  
  throws CouldNotDecode  
  {  
    this(new ByteWrapper(buffer, offset));  
  }

/**
   * Constructs <code>this</code> from the supplied <code>ByteWrapper</code>.  
   * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of <code>this</code> begins  
   * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
   */
  public HLAinteractionCount(ByteWrapper byteWrapper)  
  throws CouldNotDecode  
  {  
    this();  
    decode(byteWrapper);  
  }
/**
 * Constructs <code>this</code> from the provided <code>Map</code>
 * @param theMap the <code>Map</code> representation of the
 * @throws CouldNotDecode if the <code>Map</code> could not be decoded
 */

public HLAinteractionCount(java.util.Map theMap)
    throws CouldNotDecode
{
    this();
    try {
        putAll(theMap);
    } catch (Exception e) {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//end HLAinteractionCount
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;

/**
 * Type-safe MOM HLAInteractionSubscription fixed record type.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault]
 * @version 1.1
 *
 * @throws CouldNotDecode if an {@link InstantiationException}
 * occurs
 *
 * @param buffer the network-provided <code>byte[]</code>
 * representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public class HLAinteractionSubscription
 extends HLAfixedRecordType
 {
 /** First field. */
 public HLAhandle HLAinteractionClass;
 /** Second field. */
 public HLAboolean HLAactive;

 /**
 * Constructs <code>this</code> of default values.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public HLAinteractionSubscription(byte[] buffer)
 throws CouldNotDecode
 {
 this(buffer, 0);
 }
/**
 * Constructs this from the network representation in the provided byte[] at the indicated offset.
 * @param buffer the network-provided byte[] representation of this
 * @param offset where in the buffer the representation of this begins
 * @throws CouldNotDecode if the buffer could not be decoded
 */
public HLAInteractionSubscription(byte[] buffer, int offset)
    throws CouldNotDecode
{    this(new ByteWrapper(buffer, offset));
}

/**
 * Constructs this from the supplied ByteWrapper.
 * @param byteWrapper the ByteWrapper at whose current pos() the representation of this begins
 * @throws CouldNotDecode if the ByteWrapper could not be decoded
 */
public HLAInteractionSubscription(ByteWrapper byteWrapper)
    throws CouldNotDecode
{    this();    decode(byteWrapper);
}
/**
 * Constructs <code>this</code> from the provided <code>Map</code>. 
 * @param theMap the <code>Map</code> representation of 
 * <code>this</code>
 * @throws CouldNotDecode if the <code>Map</code> could not be 
 * decoded
 */

public HLAinteractionSubscription(java.util.Map theMap) 
throws CouldNotDecode
{
    this();
    try
    {
        putAll(theMap);
    }
    // Wrap all exceptions as CouldNotDecode
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//end HLAinteractionSubscription
import hla.rti1516.CouldNotDecode;

/**
 * Type-safe MOM HLAobjectClassBasedCount fixed record data type.
 * @author Daniel Thibault (Daniel.U.Thibault@DRDC-RDDC.gc.ca)
 * @version 1.1
 */
public class HLAobjectClassBasedCount
extends HLAfixedRecordType
{
    /** First field. */
    public HLAhandle HLAinteractionClass;
    /** Second field. */
    public HLAcount HLAinteractionCount;

    /**
     * Constructs <code>this</code> with default values.
     * @throws CouldNotDecode if an InstantiationException occurs
     */
    public HLAobjectClassBasedCount()
        throws CouldNotDecode
    {
        try
        {
            initializeFields();
        }
        catch (InstantiationException e)
        {
            CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
            throw (CouldNotDecode)cnd.initCause(e);
        }
    }

    /**
     * Constructs <code>this</code> from the network representation in
     * the provided <code>byte[]</code>.
     * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
     * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
     */
    public HLAobjectClassBasedCount(byte[] buffer)
        throws CouldNotDecode
    {
        this(buffer, 0);
    }
}
/**
 * Constructs <code>this</code> from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the 
 * representation of <code>this</code> begins 
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded 
 */

public HLAObjectClassBasedCount(byte[] buffer,
                                 int offset)
    throws CouldNotDecode
{    
    this(new ByteWrapper(buffer, offset));
}

/**
 * Constructs <code>this</code> from the supplied 
 * <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current 
 * <code>pos()</code> the representation of <code>this</code> begins 
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded 
 */

public HLAObjectClassBasedCount(ByteWrapper byteWrapper)
    throws CouldNotDecode
{    
    this();
    decode(byteWrapper);
}
/**
 * Constructs <code>this</code> from the provided <code>Map</code>. 
 * @param theMap the <code>Map</code> representation of the 
 * <code>this</code>
 * @throws CouldNotDecode if the <code>Map</code> could not be 
 * decoded
 */

public HLAobjectClassBasedCount(java.util.Map theMap) 
    throws CouldNotDecode 
{ 
    this();
    try
    {
        putAll(theMap);
    }
    //Wrap all exceptions as CouldNotDecode
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//end HLAobjectClassBasedCount
package ca.gc.drdc_rddc.hla.rtiI516.omti;
import hla.rtiI516.CouldNotDecode;

/**
 * Type-safe MOM HLAsyncPointFederate fixed record data type.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault]
 * @version 1.1
 **/ public class HLAsyncPointFederate
 extends HLAfixedRecordType
 {
 /** First field. */
 public HLAhandle HLAfederate;
 /** Second field. */
 public HLAsyncPointStatus HLAfederateSyncStatus;

 /**
 * Constructs <code>this</code> with default values.
 * @throws CouldNotDecode if an [link InstantiationException] occurs
 */
 public HLAsyncPointFederate()
 throws CouldNotDecode
 {
 try
 initializeFields();
 }
 catch (InstantiationException e)
 {
 CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
 throw (CouldNotDecode)cnd.initCause(e);
 }

 /**
 * Constructs <code>this</code> from the network representation in
 the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
 public HLAsyncPointFederate(byte[] buffer)
 throws CouldNotDecode
 {
 this(buffer, 0);
 }
/**
 * Constructs <code>this</code> from the network representation in
 * the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of <code>this</code>
 * @param offset where in the <code>buffer</code> the
 * representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be
 * decoded
 */
 public HLAsyncPointFederate(byte[] buffer,
 int offset)
 throws CouldNotDecode
 {   this(new ByteWrapper(buffer, offset));
 }

 /**
 * Constructs <code>this</code> from the supplied
 * <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current
 * <code>pos()</code> the representation of <code>this</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not
 * be decoded
 */
 public HLAsyncPointFederate(ByteWrapper byteWrapper)
 throws CouldNotDecode
 {   this();
     decode(byteWrapper);
 }
/**
 * Constructs <code>this</code> from the provided <code>Map</code>
 * representation of the <code>this</code>
 * @param theMap the <code>Map</code> representation of the <code>this</code>
 * @throws CouldNotDecode if the <code>Map</code> could not be decoded
 */
public HLAsyncPointFederate(java.util.Map theMap)
    throws CouldNotDecode
{
    this();
    try {
        putAll(theMap);
    }
    //Wrap all exceptions as CouldNotDecode
    catch (Exception e) {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//end HLAsyncPointFederate
The HLAVariantRecordDatatype interface extends HLADataType. The
HLAVariantRecordType abstract class extends HLABasicType. Oddly, variant
record and fixed record classes have relatively little in common.

// File: HLAVariantRecordDatatype.java
package ca.gc.drdc.rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Iterator;

/**
 * Interface implemented by the HLA variant record data types.
 * <p>
 * In addition to this interface, the HLA variant record data type
 * classes are expected to supply:
 * <ul>
 * <li> Constructors (default and specified-value)</li>
 * <li> Constructor (byte[] buffer) throws CouldNotDecode</li>
 * <li> Constructor (byte[] buffer, int offset) throws
 * CouldNotDecode</li>
 * <li> Constructor (ByteWrapper byteWrapper) throws
 * CouldNotDecode</li>
 * <li> java.lang.Object methods toString(); equals(Object
 * otherObject); and hashCode()</li>
 * <li> Class-specific extensions as necessary</li>
 * </ul>
 * A variant record consists of a discriminant field (an enumerated
 * type), to each value of which
 * corresponds a (possibly different or null) alternative field (any
 * type).
 * The octetBoundary of a variant records is the largest of the
 * octetBoundaries of the discriminant
 * and the various possible alternatives.
 * encodedLength will obviously depend on the encodedLengths of the
 * various fields, discriminants and alternatives.
 * <p>
 * The behaviour of the type isn't clearly defined by the
 * specification; my assumptions follow.
 * Unlike the fixed record type, the variant record type's
 * discriminant and alternative fields
 * will be protected and access will occur through get and set
 * methods.
 * Changing the discriminant value will clear the previous alternative
 * and generate a default alternative.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca DRDC
 * Valcartier}}
 * @version 1.1
 */
public interface HLAvariantrecorddatatype extends HLAdatatype
{
/**
 * Returns an Iterator over the discriminant values.
 * This will be the same as the discriminant's class' iterator().
 * @return the requested Iterator
 */
Iterator discriminantIterator();

/**
 * Returns an Iterator over the alternative names (as Strings) by discriminant value.
 * This will be in lock-step with the discriminantIterator.
 * Null alternatives will have the zero-length String as name.
 * @return the requested Iterator
 */
Iterator alternativeNamesIterator();

/**
 * Returns an Iterator over the alternative Classes by discriminant value.
 * This will be in lock-step with the discriminantIterator.
 * @return the requested Iterator
 */
Iterator alternativeClassIterator();

/**
 * Returns a copy (clone) of the variant record's discriminant (an HLA enumerated type instance).
 * @return the requested discriminant
 * @see #setDiscriminant
 */
HLAenumerateddatatype getDiscriminant();

/**
 * Returns the variant record's discriminant's name (as a String).
 * @return the requested discriminant's name (as a String)
 */
String getDiscriminantName();
/**
 * Sets the discriminant's value.
 * @param newDiscriminant an HLAenumerateddatatype (whose class
 * must match the discriminant's) which will be copied to the
 * discriminant
 * @return a String specifying the resulting Alternative's name
 * (could be null)
 * @throws CouldNotDecode if something goes awry
 * @see #getDiscriminant
 */
 String
 setDiscriminant(HLAenumerateddatatype newDiscriminant)
 throws CouldNotDecode;

/**
 * Returns true if an alternative exists for the variant record's
 * current discriminant value (false if null).
 * @return a boolean which is true if there is a current
 * alternative
 */
 boolean
 hasAlternative();

/**
 * Returns a copy of the variant record's current alternative
 * (either null or some HLA data type instance).
 * @return the requested alternative
 * @see #setAlternative
 */
 HLAdatatype
 getAlternative();

/**
 * Returns the Class of the variant record alternative associated
 * with the specified discriminant value.
 * @param otherDiscriminant an HLAenumerateddatatype specifying the
 * discriminant value to query the alternatives with
 * @return the requested Class (could be null)
 * @throws IllegalArgumentException if the discriminant is not of
 * the correct class
 */
 Class
 getAlternativeClass(HLAenumerateddatatype otherDiscriminant)
 throws IllegalArgumentException;

/**
 * Returns the variant record's current alternative's name (as a
 * String).
 * @return the requested alternative's name (as a String)
 */
 String
 getAlternativeName();
/**
 * Returns the name (as a String) of the variant record's alternative associated with the specified discriminant value.
 * @param discriminant an HLAenumerateddatatype specifying the discriminant value to query the alternatives with
 * @return the requested alternative's name (as a String)
 * @throws IllegalArgumentException if the discriminant is not of the correct class
 */
String getAlternativeName(HLAenumerateddatatype discriminant) throws IllegalArgumentException;

/**
 * Sets the variant record's current alternative to the specified value.
 * @param newAlternative an HLAdatatype (whose class must match the current alternative's) whose value will be copied to the record's alternative
 * @return the current alternative's previous HLAdatatype value
 * @throws CouldNotDecode if something goes awry
 * @see #getAlternative
 */
HLAdatatype setAlternative(HLAdatatype newAlternative) throws CouldNotDecode;
} //end HLAvariantrecorddatatype
package ca.gc.drdc_rddc.hla.rti1516.omti;

import hla.rti1516.CouldNotDecode;
import java.util.Collection;
import java.util.Iterator;
import java.util.ArrayList;

/**
 * HLA variant record data type abstract base class.
 * All user-defined variant record data types should descend from this
 * class and need only specify constructors.
 * <p>
 * The discriminant and alternatives held by this class will never be
 * exposed to the outside;
 * any references passed in will have clones stored, and any instances
 * emanating from the class
 * will be clones of the stored instances.
 * <p>
 * See HLAdemoVariantRecord for an example of how to use this class
 * with variant record data types.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public abstract class HLAvariantRecordType
extends HLAbasicType
implements HLAvariantrecorddatatype
{
//If at all possible, it is strongly recommended that the concrete
class declare this field:
// public static final int
// octetBoundary = <whatever>;

//These fields are required of the concrete class:

//The discriminant Class
// public static final Class
// discriminantClass = HLAboolean.class;

//The discriminant name
// public static final String
// discriminantName = "bool";

//The array of alternative classes
// public static final Class[]
// alternativeClass = [ null, HLAunicodeString.class ];

//The array of alternative names
// public static final String[]
// alternativeName = [ "NA", "theString" ];

//The remaining fields are for this class’ consumption
/** The discriminant value. */
protected HLAenumerateddatatype
_discriminant;

/** The discriminant index (into its Iterator) */
protected int
_discriminantIndex;

/** The discriminant cardinality. */
protected int
_discriminantCardinality;

/** The current alternative. */
protected HLAdatatype
_alternative;
/**
 * Initializes the <code>HLAvariantRecordType</code> from the class
 * parameters.
 * For the concrete class to function, initialize() *must* be
 * called by the void constructor.
 * <p>
 * The discriminant is created using its void constructor, and the
 * corresponding
 * alternative (if any) created likewise.
 * The arrays of alternative classes and names must have their
 * lengths equal to
 * the cardinality of the discriminant's class' Iterator.
 * Null alternatives must have null classes and be named "NA".
 * @throws CouldNotDecode if any of the requirements are not met or
 * something else goes wrong
 */
public void
initialize()
throws CouldNotDecode
{
//Wrap all exceptions in CouldNotDecode
try {
    discriminantIndex = -1;
    discriminant =
(HLAenumerateddatatype)discriminantClass().newInstance();
    //Using the discriminant's Iterator, find its cardinality
    //and identify the current discriminantIndex in passing
    Iterator itr = discriminantIterator();
    discriminantCardinality = 0;
    while (itr.hasNext())
    {
        if (itr.next().equals(_discriminant))
            _discriminantIndex =
                discriminantCardinality;
            discriminantCardinality++;
    }
    //Badly behaved HLAenumerateddatatype will either return a
    zero discriminantSize or fail to index itself
    if (_discriminantIndex < 0) throw
        new
        IllegalArgumentException(discriminantClass().getName());
    //alternativeName and alternativeClass must have the same
    cardinality as the discriminantClass
    if (discriminantCardinality !=
    ((String[])getThisField("alternativeName").length)
        throw new
        IllegalArgumentException("alternativeName.length != " +
            discriminantClass().getName() + " cardinality");
    if (discriminantCardinality !=
    ((Class[])getThisField("alternativeClass").length)
        throw new
        IllegalArgumentException("alternativeClass.length != " +
            discriminantClass().getName() + " cardinality");
}
//Validate the alternatives
for (int i = 0; i < _discriminantCardinality; i++)
{
    if (alternativeClass(i) == null)
    {
        if (alternativeName(i).compareToIgnoreCase("NA") != 0)
            throw new IllegalArgumentException("alternativeName[" + i + "] != \"NA\")
        } else {
            if (alternativeName(i).equals("*") ||
                alternativeName(i).compareToIgnoreCase("NA") == 0)
                throw new IllegalArgumentException("alternativeName[" + i + "] = \"" +
                        alternativeName(i) + "\"");

            if (!HLAdatatype.class.isAssignableFrom(alternativeClass(i))) throw new
                IllegalArgumentException(alternativeClass(i).getName() + " !=
                        HLAdatatype")
            }

        if (hasAlternative()) _alternative =
            (HLAdatatype)alternativeClass(_discriminantIndex).newInstance();
    } catch (Exception e) { //InstantiationException,
        IllegalArgumenteException
            CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
            throw (CouldNotDecode)cnd.initCause(e);
    }

    //HLAdatatype implementation

    /**
     * Returns the length (in bytes) of the <code>byte[]</code>
     * representation of <code>this</code>
     * <p>
     * As a general rule, Variant Records [HLAvariantRecord], Dynamic
     * Arrays [HLAvariableArray], and any
     * Fixed Arrays [HLAfixedArray] or Fixed Records [HLAfixedRecord]
     * containing either of these first two
     * cannot have a constant encodedLength.
     * @return the length (in bytes) of the <code>byte[]</code>
     * representation of <code>this</code>
     */
    public final int
        encodedLength()
    { //Note that the null-alternative encodedLength may be different
        from _discriminant.encodedLength()
            if (!hasAlternative()) return _discriminant.encodedLength();
            return octetBoundary() + _alternative.encodedLength();
    }
>Returns the octet boundary of <code>this</code>.

The octet boundary value is defined as the smallest power of 2
which is greater than or equal to the size of the datatype in bytes.
For a constructed datatype, it is the maximum octet boundary
value of all components within it.

In all cases octetBoundary is constant.
For HLAvariantRecord, the octet boundary is determined as the
largest octet boundary of the
discriminant and all possible alternatives.

@returns the octet boundary of <code>this</code>

```
public final int
octetBoundary()
{
    Object o = getField("octetBoundary");
    if (o != null) return ((Integer)o).intValue();
```
Class alt;
int altBoundary;
int theBoundary = _discriminant.octetBoundary();
// Iterate over the possible alternatives
Iterator itr = alternativeClassIterator();
while (itr.hasNext())
{
    alt = (Class)itr.next();
    // We skip the case where one of the alternatives is self, to avoid infinite recursion
    // Note that this isn’t fool-proof, since we could have, for example,
    // an alternative which is a dynamic array of this class...
    if ((alt != null) && (!alt.equals(this.getClass())))
    {
        altBoundary = 0;
        try
        {
            altBoundary = alt.getField("octetBoundary").getInt(null);
        }
        catch (Exception e) // NoSuchFieldException,
        { // Most likely the class does not have a static octetBoundary field
            try
            {
                altBoundary = ((HLAdatatype)alt.newInstance()).octetBoundary();
            }
            catch (Exception ee) // ClassCastException, IllegalAccessException, InstantiationException, ExceptionInInitializerError, SecurityException
            {
                // Give up!
            }
        }
        if (altBoundary > theBoundary) theBoundary = altBoundary;
    }
}
return theBoundary;
}
/**
 * Encodes <code>this</code> into the <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> into which to encode <code>this</code>
 * @return the <code>ByteWrapper</code>
 */
public ByteWrapper
code(ByteWrapper byteWrapper) {
    byteWrapper.align(octetBoundary());
    _discriminant.encode(byteWrapper);
    if (hasAlternative()) _alternative.encode(byteWrapper);
    return byteWrapper;
}

/**
 * Sets <code>this</code> value from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the data type begins 
 * @return the <code>ByteWrapper</code>
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public ByteWrapper
decode(ByteWrapper byteWrapper) throws CouldNotDecode {
    try {
        byteWrapper.align(octetBoundary());
        _discriminant.decode(byteWrapper);
        _discriminantIndex = getIndex(_discriminant);
        if (!hasAlternative()) return byteWrapper;
        _alternative = (HLAdatatype)alternativeClass(_discriminantIndex).newInstance();
        _alternative.decode(byteWrapper);
        return byteWrapper;
    } catch (Exception e) //InstantiationException, IllegalAccessException
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//HLAVariantRecordDatatype implementation

/**
 * Returns an Iterator over the discriminant values.
 * This will be the same as the discriminant's class' iterator().
 * @return the requested Iterator
 */
public Iterator discriminantIterator()
{
    return _discriminant.iterator();
}

/**
 * Returns an Iterator over the alternative names (as Strings) by
 * discriminant value.
 * This will be in lock-step with the discriminantIterator.
 * Null alternatives will have the zero-length String as name.
 * @return the requested Iterator
 */
public Iterator alternativeNamesIterator()
{
    immutableArrayList ial = new
    immutableArrayList(_discriminantCardinality);
    for (int i = 0; i < _discriminantCardinality; i++)
        ial.add(alternativeName(i));
    ial.trimToSize();
    return ial.iterator();
}

/**
 * Returns an Iterator over the alternative Classes by discriminant
 * value.
 * This will be in lock-step with the discriminantIterator.
 * @return the requested Iterator
 */
public Iterator alternativeClassIterator()
{
    //You know, it's really annoying that ArrayList doesn't have an
    //Object[] constructor...
    immutableArrayList ial = new
    immutableArrayList(_discriminantCardinality);
    //No need to clone the Class objects as there are no methods
    //that could change their state
    for (int i = 0; i < _discriminantCardinality; i++)
        ial.add(alternativeClass(i));
    ial.trimToSize();
    return ial.iterator();
}
/**
 * Returns a copy (clone) of the variant record's discriminant (an
 * HLA enumerated type instance).
 * @return the requested discriminant
 * @see #setDiscriminant
 */
public HLAenumerateddatatype
getDiscriminant()
{
try
{
    return (HLAenumerateddatatype)cloneThroughInterface(_discriminant);
}
catch (Exception e) // (InstantiationException), CouldNotDecode
{
    return null;
}
}

/**
 * Returns the variant record's discriminant's name (as a String).
 * @return the requested discriminant's name (as a String)
 */
public String
getDiscriminantName()
{
    return (String)this.getField("discriminantName");
}
/**
 * Sets the discriminant's value.
 * @param newDiscriminant an HLAenumerateddatatype (whose class
 * must match the discriminant's) which will be copied to the
 * discriminant
 * @return a String specifying the resulting Alternative's name
 * (could be null)
 * @throws CouldNotDecode if something goes awry
 * @see #getDiscriminant
 */
 public String
 setDiscriminant(HLAenumerateddatatype newDiscriminant)
 throws CouldNotDecode
 {
   int oldIndex = _discriminantIndex;
   _discriminant = newDiscriminant.toByteArray();
   _discriminantIndex = getIndex(_discriminant);
   if (_discriminantIndex == oldIndex) return getAlternativeName();
   _alternative = null;
   if (lhasAlternative()) return getAlternativeName();
   try
   {
     _alternative = (HLADatatype)alternativeClass(_discriminantIndex).newInstance();
   }
   catch (Exception e) //InstantiationException,
   IllegalAccessRestriction
   {
     CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
     throw (CouldNotDecode)cnd.initCause(e);
   }
   return getAlternativeName();
 }

/**
 * Returns true if an alternative exists for the variant record's
 * current discriminant value (false if null).
 * @return a boolean which is true if there is a current
 * alternative
 */
 public boolean
 hasAlternative()
 {
   return (null != alternativeClass(_discriminantIndex));
 }
/**
 * Returns a copy (clone) of the variant record's current alternative (either null or some HLA data type instance).
 * @return the requested alternative
 * @see #setAlternative
 */

public HLADatatype getAlternative()
{
    if (!_hasAlternative()) return null;
    try
    {
        return cloneThroughInterface(_alternative);
    }
    catch (Exception e) // (InstantiationException), CouldNotDecode
    {
        return null;
    }
    
/**
 * Returns the Class of the variant record alternative associated with the specified discriminant value.
 * @param otherDiscriminant an HLAEnumeratedDatatype specifying the discriminant value to query the alternatives with
 * @return the requested Class (could be null)
 * @throws IllegalArgumentException if the discriminant is not of the correct class
 */

public Class getAlternativeClass(HLAEnumeratedDatatype otherDiscriminant)
throws IllegalArgumentException
{
    if (_discriminantClass().equals(otherDiscriminant.getClass()))
    {
        return new HLAEnumeratedDatatype();
    }
    throw new IllegalArgumentException()
    
/**
 * Returns the variant record's current alternative's name (as a String).
 * @return the requested alternative's name (as a String)
 */

public String getAlternativeName()
{
    return alternativeName(_discriminantIndex);
/**
 * Returns the name (as a String) of the variant record's
 * alternative associated with the specified discriminant value.
 * @param otherDiscriminant an HLAnumbereddatatype specifying the
 * discriminant value to query the alternatives with
 * @return the requested alternative's name (as a String)
 * @throws IllegalArgumentException if the discriminant is not of
 * the correct class
 */
public String getAlternativeName(PLOYnubereddatatype otherDiscriminant)
  throws IllegalArgumentException
{
  if (!discriminantClass().equals(otherDiscriminant.getClass()))
    throw new IllegalArgumentException();
  int index = getIndex(otherDiscriminant);
  if (index < 0) throw new IllegalArgumentException();
  return alternativeName(index);
}

/**
 * Sets the variant record's current alternative to the specified
 * value.
 * @param newAlternative an HLADatatype (whose class must match the
 * current alternative's) whose value will be copied to the record's
 * alternative
 * @return the current alternative's previous HLADatatype value
 * @throws CouldNotDecode if something goes awry
 * @see #getAlternative
 */
public HLADatatype setAlternative(HLADatatype newAlternative)
  throws CouldNotDecode
{
  //Setting null to null is the trivial operation
  if ((newAlternative == null) && (_alternative == null)) return null;
  //Switching from null to non-null or vice-versa is not allowed
  if ((newAlternative == null) || (_alternative == null)) throw
    new CouldNotDecode("NullPointerException");
  //Classes must otherwise match
  if (!alternativeClass(discriminantIndex).equals(newAlternative.getClass()))
    throw new CouldNotDecode(newAlternative.getClass().getName());

  //Prepare the clone that will be returned
  HLADatatype r = cloneThroughInterface(_alternative);
  //Update the stored instance's value
  _alternative.decode(newAlternative.toByteArray());
  return r;
}
//java.lang.Object methods

/**
   * Returns a String representation of this.
   * The String representation consists of fixed-record-like list of
   * key-value mappings
   * (listing the discriminant and then the alternative) enclosed in
   * braces
   * ("{}", "{}") and separated by the characters (comma and space).
   * Each key-value mapping is rendered as the key followed by an
   * equals sign ("=") followed by the associated value.
   * Keys and values are converted to strings as by
   * String.valueOf(Object).
   * In the special case of null alternatives, the key is represented
   * by "NA" and the value rendered as "null".
   * @return a String representation of this
   */
public String toString()
{
    StringBuffer buf = new StringBuffer();
    buf.append("{" + getDiscriminantName() + ":" +
        _discriminant.toString() + ", ");
    buf.append(getAlternativeName() + ":");
    buf.append(hasAlternative() ? _alternative.toString() : "null");
    buf.append("}");
    return buf.toString();
}
/**
 * Returns true iff `this` and `otherObject` represent the same object.
 * @param otherObject the `Object` to compare with
 * @return `true` if `otherObject` is of the same type as `this` and has the same value
 * @see `Object#hashCode Object.hashCode()`
 * @see `java.util.Hashtable Hashtable`
 */
public boolean equals(Object otherObject) {
    if (this == otherObject) return true;
    // This form is more inheritable, and will ensure that
    otherObject is an instance of this' class (or a sub-class)

    // We disallow subclassing
    if (!this.getClass().equals(otherObject.getClass())) return false;
    if (!this.getClass().isInstance(otherObject)) return false;

    if (!_discriminant.equals(((HLAvariantrecorddatatype)otherObject).getDiscriminant())) return false;
    // Compare alternatives now
    HLAdatatype otherField = ((HLAvariantrecorddatatype)otherObject).getAlternative();
    if (((alternative == null) && (otherField == null)) return true;
    if (((alternative == null) || (otherField == null)) return false;
    return _alternative.equals(otherField);
}

/**
 * Returns the hash code value for this.
 * @return the hash code value for this
 * @see `java.util.Map.Entry#hashCode()`
 * @see `Object#hashCode()`
 * @see `Object#equals(Object)`
 */
public int hashCode() {
    int h = _discriminant.hashCode();
    return (hasAlternative() ? h + _alternative.hashCode() : h);
}
// Cloneable implementation

/**
 * Creates and returns an independent copy of this object.
 * @return an independent copy of this Object
 * @throws CloneNotSupportedException if the object's class is not
 * Cloneable or if the instance cannot be cloned
 */

public Object clone() throws CloneNotSupportedException {
    // The method inherited from Object does a shallow copy;
    // the fields will have been copied too, so the clone's fields
    // will hold references to the same objects as the original.

    try {
        // return new HLAvariantRecordType(toByteArray()); // This class
        // doesn't have this constructor
        // return new HLAvariantRecordType(_discriminant,
        // _discriminantName, _alternatives, _alternativeNames);
        return cloneThroughInterface(this);
    }
    // Something is seriously wrong with the class if it can't encode
    // and then decode itself...
    catch (Exception e) { // Couldn'tDecode
        CloneNotSupportedException cnse = new CloneNotSupportedException(e.getMessage());
        throw (CloneNotSupportedException) cnse.initCause(e);
    }

    // Supporting classes and methods

    /**
     * Returns the specified (static) field's value as an Object.
     * This is just an instance-proxy for the various class fields.
     * @param fieldName a String specifying the field's name
     * @return the requested field, as a Object (null in case of
     * failure)
     */
    private Object getField(String fieldName) {
        try {
            return this.getClass().getField(fieldName).get(null);
        }
        catch (Exception e) { // NoSuchFieldException,
            return null;
        }
    }
}
/**
 * Returns the discriminant Class.
 * This is just an instance-proxy for the class field.
 * @return the discriminant Class
 */
private Class discriminantClass()
{
    return (Class)getField("discriminantClass");
}

/**
 * Returns the alternative name at the specified position within the array.
 * This is just an instance-proxy for the class field, with indexing.
 * @param index an int specifying the position to look up within the array
 * @return the alternative name requested
 */
private String alternativeName(int index)
{
    return ((String[])getField("alternativeName"))[index];
}

/**
 * Returns the alternative Class at the specified position within the array.
 * This is just an instance-proxy for the class field, with indexing.
 * @param index an int specifying the position to look up within the array
 * @return the requested Class
 */
private Class alternativeClass(int index)
{
    return ((Class[])getField("alternativeClass"))[index];
}
// Clone an HLADatatype object through its byte[] constructor.
private HLADatatype cloneThroughInterface(HLADatatype source)
throws CouldNotDecode
{
    if (source == null) return null;
    try
    {
        // In plain English, we fetch the object's byte[] constructor
        // and invoke it on the object's toByteArray() encoding
        return (HLADatatype) source.getClass().getConstructor(new
        Class[ ] { byte[].class }).newInstance(new Object[ ] {
            source.toByteArray() });
        // Equivalently:
        // HLADatatype theClone =
        // source.getClass().newInstance();
        // theClone.decode(source.toByteArray());
        // return theClone;
    }
    catch (Exception e) // NoSuchMethodException,
    InstantiationException
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

// Compute the index of the specified discriminant value into the
// discriminant's Iterator.
private int
getIndex(HLAenumerateddatatype theDiscriminant)
{
    int index = -1;
    Iterator itr = discriminantIterator();
    while (itr.hasNext())
    {
        index++;
        if (itr.next().equals(theDiscriminant)) return index;
    }
    return -1;
}

// An ArrayList that denies the iterator's remove() and its kin
private class immutableArrayList
extends ArrayList
{
    immutableArrayList()
    {
        super();
    }

    immutableArrayList(Collection c)
    {
        super(c);
    }
immutableArrayList(int initialCapacity)
{
    super(initialCapacity);
}

public Object
remove(int index)
{
    throw new UnsupportedOperationException();
}

// ArrayList's implementation does not rely on its remove(), so
// we must override this method too
public void
clear()
{
    throw new UnsupportedOperationException();
}

// These we do not need to override; they are inherited from
// AbstractCollection and rely on remove(int)
// public boolean
// remove(Object o)
// public boolean
// removeAll(Collection c)
// public boolean
// retainAll(Collection c)
}

// end HLAvariantRecordType
Since the MOM does not provide any variant record examples, we illustrate our design pattern through the HLA\texttt{demoVariantRecord} demonstration class.

```java
// File: HLA\texttt{demoVariantRecord}.java
package ca.gc.drdc\_rddc.hla.rti1516.\texttt{omt};
import hla.rti1516.\texttt{CouldNotDecode};

/**
 * HLA demo variant record data type.
 * @author \{\texttt{mailto:Daniel.Thibault@DRDC-RDDC.gc.ca} Daniel U.
 * Thibault\} (\{\texttt{http://www.valcartier.drdc-rddc.gc.ca} DRDC
 * Valcartier\})
 * @version 1.1
 */
public class HLA\texttt{demoVariantRecord}
    extends HLA\texttt{variantRecordType}
{
    /** The discriminant Class. */
    public static final Class \texttt{discriminantClass} = HLA\texttt{boolean.class};

    /** The discriminant name. */
    public static final String \texttt{discriminantName} = "bool";

    /** The array of alternative classes. */
    public static final Class[] \texttt{alternativeClass} = {
        null, HLA\texttt{unicodeString.class}
    };

    /** The array of alternative names. */
    public static final String[] \texttt{alternativeName} = {
        "NA", "theString"
    };

    /**
     * Constructs a default \texttt{HLA\texttt{demoVariantRecord}}.
     * @throws \texttt{CouldNotDecode} if initialization fails
     */
    public HLA\texttt{demoVariantRecord}()
        throws \texttt{CouldNotDecode}
    {
        //super() not called because super-class is abstract
        initialize();
    }
}
/**
 * Constructs a <code>HLAdemoVariantRecord</code> from another one.
 * @param other the HLAdemoVariantRecord instance to copy
 * @throws CouldNotDecode if initialization fails
 */
public
HLAdemoVariantRecord(HLAdemoVariantRecord other)
throws CouldNotDecode
{
    this();
    setDiscriminant(other.getDiscriminant());
    setAlternative(other.getAlternative());
}

// The remaining constructors are boiler-plate

/**
 * Creates a <code>HLAdemoVariantRecord</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAdemoVariantRecord</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAdemoVariantRecord(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}

/**
 * Creates a <code>HLAdemoVariantRecord</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>HLAdemoVariantRecord</code>
 * @param offset where in the <code>buffer</code> the <code>HLAdemoVariantRecord</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public
HLAdemoVariantRecord(byte[] buffer,
        int offset)
throws CouldNotDecode
{
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a <code>HLAdemoVariantRecord</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>HLAdemoVariantRecord</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public HLAdemoVariantRecord(ByteWrapper byteWrapper) throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}

//end HLAdemoVariantRecord
The `Normalization` class implements a number of normalization and unnormalization methods statically.

```java
// File: Normalization.java
package ca.gc.drdc_rddc.hla.rti1516.omti;

import java.math.BigInteger;
import hla.rti1516.*;

/**
 * Utility class for normalizing values over dimensions.
 * This first draft uses only static methods, but that is clearly inefficient.
 * I think we should have a Normalization constructor that expects two parameters:
 * the RTIambassador instance and the DimensionHandle (or, equivalently, the dimension's name).
 * This would allow the Normalization instance to cache the dimension's upper bound at the outset,
 * and all further methods would be slaved to that specific dimension.
 * This pattern could be further extended by passing the domain's bounds to the constructor as well.
 * In that latter case, not all members would be accessible (an enumerated set domain, for example,
 * would not allow the other normalization methods).
 * @author Daniel U. Thibault (Daniel.Thibault@DRDC-RDDC.gc.ca)
 * @version 1.2
 */
```
public class Normalization
{

/**
 * Linear normalization function for integer values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto
 * which to normalize the domain
 * @param domainLower the lower bound on the possible values of domain
 * @param domainUpper the upper bound on the possible values of domain
 * @param domain a nonenumerated integer value known to the federate using the dimension
 * @return a long falling within the dimension’s @link RangeBounds
 * @throws InvalidDimensionHandle if the @link DimensionHandle is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static long linear(RTIambassador rti,
DimensionHandle dimension,
long domainLower,
long domainUpper,
long domain)
throws InvalidDimensionHandle,
FederateNotExecutionMember,
RTIinternalError,
FederateInternalError
{
    try
    {
        // [(domain - domainLower) / (domainUpper - domainLower)] * (DUB - 1)
        // or more accurately
        // [(domain - domainLower) * (DUB - 1)] / (domainUpper - domainLower)
        BigInteger dub = new BigInteger(Long.toString(rti.getDimensionUpperBound(dimension) - 1));
        BigInteger l = new BigInteger(Long.toString(domainLower));
        BigInteger u = new BigInteger(Long.toString(domainUpper));
        BigInteger d = new BigInteger(Long.toString(domain));
        // return ((d.subtract(l)).multiply(dub)).divide(u.subtract(l)).longValue();
        return d.subtract(l).multiply(dub).divide(u.subtract(l)).longValue();
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Linear unnormalization function.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension
 * whence the normalized value comes
 * @param normalized a long falling within the dimension's [link
 * RangeBounds]{}
 * @param domainLower the lower bound on the possible values of
 * domain
 * @param domainUpper the upper bound on the possible values of
 * domain
 * @return a nonenumerated integer value known to the federate
 * using the dimension
 * @throws InvalidDimensionHandle if the {link DimensionHandle} is
 * invalid
 * @throws FederateNotExecutionMember if the federate is not
 * currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an
 * RTIambassador call
 * @throws FederateInternalError if something goes wrong with the
 * rest of the method
 */

public static long
unlinear(RTIambassador rti,
       DimensionHandle dimension,
       long normalized,
       long domainLower,
       long domainUpper)
  throws InvalidDimensionHandle,
         FederateNotExecutionMember,
         RTIinternalError,
         FederateInternalError
{
  try
  {
    // domainLower + ceiling([normalized * (domainUpper -
    domainLower)] / (DUB - 1))
    BigInteger dub = new
    BigInteger(Long.toString(rti.getDimensionUpperBound(dimension) - 1));
    BigInteger n = new BigInteger(Long.toString(normalized));
    BigInteger l = new BigInteger(Long.toString(domainLower));
    BigInteger u = new BigInteger(Long.toString(domainUpper));
    return l.add(ceiling(n.multiply(u.subtract(l)),
    dub)).longValue();
  } catch (NumberFormatException e) {
    throw new FederateInternalError(e.getMessage());
  }
}
/**
 * Linear normalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which
to normalize the domain
 * @param domainLower the lower bound on the possible values of
domain
 * @param domainUpper the upper bound on the possible values of
domain
 * @param domain a floating-point value known to the federate using the
dimension
 * @return a long falling within the dimension’s [link
RangeBounds]
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is
invalid
 * @throws FederateNotExecutionMember if the federate is not
currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an
RTIambassador call
 * @throws FederateInternalError if something goes wrong with the
rest of the method
*/

public static long linear(RTIambassador rti,
                            DimensionHandle dimension,
                            double domainLower,
                            double domainUpper,
                            double domain)
    throws InvalidDimensionHandle,
            FederateNotExecutionMember,
            RTIinternalError,
            FederateInternalError
{
    try
    {
        // [(domain - domainLower) / (domainUpper - domainLower)] * (DUB
- 1)
        //Converting between double and long is sure a pain in the
behind, because of the "possible loss of precision" compiler error...
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        return new Double(dub*(domain - domainLower)/(domainUpper - domainLower)).longValue();
    }
    catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Linear unnormalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension whence the normalized value comes
 * @param normalized a long falling within the dimension's [link RangeBounds]
 * @param domainLower the floating-point lower bound on the possible values of domain
 * @param domainUpper the floating-point upper bound on the possible values of domain
 * @return a floating-point value known to the federate using the dimension
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static double unlinear(RTIambassador rti,
   DimensionHandle dimension,
   long normalized,
   double domainLower,
   double domainUpper)
   throws InvalidDimensionHandle,
   FederateNotExecutionMember,
   RTIinternalError,
   FederateInternalError
{
    try
    {
      // domainLower + ceiling([normalized * (domainUpper -
      domainLower)] / (DUB - 1))
      double dub =
      Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) -
      1)).doubleValue();
      double n =
      Double.valueOf(Long.toString(normalized)).doubleValue();
      //Converting between double and long is sure a pain in the
      behind, because of the "possible loss of precision" compiler error...
      return domainLower + n*(domainUpper-domainLower)/dub;
    }
    catch (NumberFormatException e) {
      throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Linear enumerated normalization function.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which to normalize the domain
 * @param domain an enumerated data type value known to the federate using the dimension
 * @return a long falling within the dimension’s [RangeBounds]
 * @throws InvalidDimensionHandle if the [DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static long linearEnumerated(RTIambassador rti, DimensionHandle dimension, HLAenumerateddatatype domain)
throws InvalidDimensionHandle, FederateNotExecutionMember, RTIinternalError, FederateInternalError
{
    try {
        long pos = -1;
        long length = 0; // This will count the cardinality of the enumerated data type
        for (java.util.Iterator i = domain.iterator(); i.hasNext(); length++)
        {
            if (i.next().equals(domain)) pos = length;
        }
        BigInteger dub = new BigInteger(Long.toString(rti.getDimensionUpperBound(dimension) - 1));
        BigInteger p = new BigInteger(Long.toString(pos));
        BigInteger l = new BigInteger(Long.toString(length - 1));
        return (p.multiply(dub)).divide(l).longValue();
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/ **Linear enumerated unnormalization function.**
  *
  @param rti the RTI ambassador to use
  *
  @param dimension a DimensionHandle specifying the dimension
  whence the normalized value comes
  *
  @param normalized a long falling within the dimension's [link
  RangeBounds]
  *
  @param domain an enumerated data type value known to the
  federate using the dimension
  *
  @return an Object (an enumerated data type instance) known to
  the federate using the dimension
  *
  @throws InvalidDimensionHandle if the [link DimensionHandle]
  is invalid
  *
  @throws FederateNotExecutionMember if the federate is not
  currently joined to a federation execution
  *
  @throws RTIinternalError if something else goes wrong within an
  RTI ambassador call
  *
  @throws FederateInternalError if something goes wrong with the
  rest of the method
  */

public static Object unlinearEnumerated(RTIambassador rti,
           DimensionHandle dimension,
           long normalized,
           HLAenumerateddatatype domain)
  throws InvalidDimensionHandle,
           FederateNotExecutionMember,
           RTIinternalError,
           FederateInternalError
{
    try
    {
        //Because HLA enumerated datatype has no "length" or
        //"cardinality" member, we must loop twice!
        //Count the cardinality of the enumerated data type
        long length = 0;
        for (java.util.Iterator i = domain.iterator(); i.hasNext();
             i.next()) { length++;
        } //Now that we know the enumerated type's cardinality, we can
decode the normalized value
        BigInteger dub = new
        BigInteger(Long.toString(rti.getDimensionUpperBound(dimension) - 1));
        BigInteger n = new BigInteger(Long.toString(normalized));
        BigInteger l = new BigInteger(Long.toString(length-1));
        long pos = n.multiply(l).divide(dub).longValue();
        for (java.util.Iterator i = domain.iterator(); i.hasNext();
             i.next())
        {
            if (pos == 0) return i.next();
            pos--;
        }
        return null;
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Enumerated set normalization function.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which to normalize the domain
 * @param theMap a Map whose entries have the domain values as keys and the long values as values
 * @param domain an enumerated data type value known to the federate using the dimension
 * @return a long falling within the dimension’s [link RangeBounds]
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static long
enumeratedSet(RTIambassador rti,
DimensionHandle dimension,
java.util.Map theMap,
HLAenumerateddatatype domain)
throws InvalidDimensionHandle,
FederateNotExecutionMember,
RTIinternalError,
FederateInternalError
{
try
{
    if (theMap == null)
    {
        //An HLAenumerateddatatype normally descends from an HLABasictype or HLABasictype
        //And should thus have a getValue() method (which can return anything from a byte to a double or even a String)
        return unwrap(domain.getClass().getMethod("getValue",
                (Class[][])null).invoke(domain, (Object[])null));
    } else {
        return unwrap(theMap.get(domain));
    }
} catch (Exception e) {
    //ClassCastException, NullPointerException, NoSuchMethodException, SecurityException, IllegalAccessException,
    //IllegalArgumentException, java.lang.reflect.InvocationTargetException,
    //ExceptionInInitializerError
    throw new FederateInternalError(e.getMessage());
}
/**
 * Enumerated set unnormalization function.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension whence the normalized value comes
 * @param normalized a long falling within the dimension's @link RangeBounds
 * @param theMap a Map whose entries have the domain values as keys and the long values as values
 * (or null to use the enumerated data type's long constructor)
 * @param domain an enumerated data type value known to the federate using the dimension
 * @return an Object (an enumerated data type instance) known to the federate using the dimension
 * @throws InvalidDimensionHandle if the @link DimensionHandle is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */
public static Object unenumeratedSet(RTIambassador rti,
        DimensionHandle dimension,
        long normalized,
        java.util.Map theMap,
        HLAenumerateddatatype domain)
    throws InvalidDimensionHandle,
               FederateNotExecutionMember,
               RTIinternalError,
               FederateInternalError
    }
    try
    {
        if (theMap != null)
        {
            //An HLAenumerateddatatype normally descends from an
            //HLAbasicdatatype or HLAbasicdatatype
            //And should thus have a setValue() method (which can take
            anything as its argument, from a byte to a double or even a String)
            //Here we assume there is a (long) constructor
            //We should really get all Constructors() and scan its
            getParameterTypes() in order of preference for long-compatible
            forms...
            //Like we do with the unwrap method.
            return domain.getClass().getConstructor(new Class[]{
                        long.class}).newInstance(new Object[]{new Long(normalized)});
            //Oddly, the Object[] required by the newInstance does get
            converted back to the primitives required by our chosen constructor...
        } else {
            for (java.util.Iterator i = theMap.entrySet().iterator();
                    i.hasNext();)
            {
                java.util.Map.Entry me = (java.util.Map.Entry)i.next();
                if (unwrap(me.getValue()) == normalized) return
                me.getKey();
            }
            return null;
        }
    }
    catch (Exception e) {
        //NoSuchMethodException, SecurityException,
        InstantiationException, IllegalAccessException,
        IllegalArgumentException, java.lang.reflect.InvocationTargetException
        throw new FederateInternalError(e.getMessage());
    }
/**
 * Logarithmic normalization function for integer values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which to normalize the domain
 * @param domainLower the lower bound on the possible values of domain
 * @param domainUpper the upper bound on the possible values of domain
 * @param domain a nonenumerated integer value known to the federate using the dimension
 * @return a long falling within the dimension's [link RangeBounds]
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */
public static long logarithmic (RTIambassador rti, DimensionHandle dimension, long domainLower, long domainUpper, long domain)
    throws InvalidDimensionHandle, FederateNotExecutionMember, RTIinternalError, FederateInternalError
{
    try
    {
        // Note that the result is independent of the logarithm's base; here we use the natural logarithm
        // [log(domain/domainLower) / log(domainUpper/domainLower)] * (DUB - 1)
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
    }
    catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Logarithmic unnormalization function for integer values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension
 * whence the normalized value comes
 * @param normalized a long falling within the dimension's {link RangeBounds}
 * @param domainLower the lower bound on the possible values of domain
 * @param domainUpper the upper bound on the possible values of domain
 * @return a long known to the federate using the dimension
 * @throws InvalidDimensionHandle if the {link DimensionHandle} is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */
public static long
unlogarithmic(RTIambassador rti,
DimensionHandle dimension,
long normalized,
long domainLower,
long domainUpper)
throws InvalidDimensionHandle,
FederateNotExecutionMember,
RTIinternalError,
FederateInternalError
{
    try
    {
        // Note that the result is independent of the logarithm's base; here we use the natural logarithm
        // domainLower * exp(log(du/dl)*normalized/(DUB - 1))
        double dub =
        Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        double n =
        Double.valueOf(Long.toString(normalized)).doubleValue();
        double dl =
        Double.valueOf(Long.toString(domainLower)).doubleValue();
        double du =
        Double.valueOf(Long.toString(domainUpper)).doubleValue();
        return
        ceiling(dl*java.lang.Math.exp(java.lang.Math.log(du/dl)*n/dub));
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Logarithmic normalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which to normalize the domain
 * @param domainLower the floating-point lower bound on the possible values of domain
 * @param domainUpper the floating-point upper bound on the possible values of domain
 * @param domain a floating-point value known to the federate using the dimension
 * @return a long falling within the dimension's {@link RangeBounds}
 * @throws InvalidDimensionHandle if the {@link DimensionHandle} is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static long logarithmic(RTIambassador rti,
                               DimensionHandle dimension,
                               double domainLower,
                               double domainUpper,
                               double domain)
                               throws InvalidDimensionHandle,
                               FederateNotExecutionMember,
                               RTIinternalError,
                               FederateInternalError
{
    try {
        // (log(domain/domainLower) / log(domainUpper/domainLower)) * (DUB - 1)
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        return new Double(dub * java.lang.Math.log(domain/domainLower) / java.lang.Math.log(domainUpper/domainLower)).longValue();
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Logarithmic unnormalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension whence the normalized value comes
 * @param normalized a long falling within the dimension's [link RangeBounds]
 * @param domainLower the floating-point lower bound on the possible values of domain
 * @param domainUpper the floating-point upper bound on the possible values of domain
 * @return a double known to the federate using the dimension
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static double unlogarithmic(RTIambassador rti,
        DimensionHandle dimension,
        long normalized,
        double domainLower,
        double domainUpper)
        throws InvalidDimensionHandle,
        FederateNotExecutionMember,
        RTIinternalError,
        FederateInternalError
{
    try
    {
        // Note that the result is independent of the logarithm's base; here we use the natural logarithm,
        // domainLower * exp(log(du/dl)*normalized)/(DUB - 1))
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        double n = Double.valueOf(Long.toString(normalized)).doubleValue();
        return domainLower*java.lang.Math.exp(java.lang.Math.log(domainUpper/domainLower)*n/dub);
    } catch (NumberFormatException e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Hyperbolic tangent normalization function for integer values.
 * @param rti the RTI ambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto
 * which to normalize the domain
 * @param domainCentre the value of domain around which the user
 * desires the greatest precision
 * @param domainSize a generic measure of the distance around the
 * domainCentre for which the user desires the relatively high precision
 * @param domain a non-enumerated integer value known to the
 * federate using the dimension
 * @return a long falling within the dimension's RangeBounds
 * @throws InvalidDimensionHandle if the DimensionHandle is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTI ambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static long tanh(RTIambassador rti,
            DimensionHandle dimension,
            long domainCentre,
            long domainSize,
            long domain)
            throws InvalidDimensionHandle,
            FederateNotExecutionMember,
            RTIinternalError,
            FederateInternalError
{
    try {
        // [{tanh([domain - domainCenter]/domainSize) + 1]/2] * (DUB - 1)
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        return new Double(dub * (tanh(((double)domain - (double)domainCentre)/(double)domainSize) + 1.0)/2.0).longValue();
    } catch (Exception e) { //NumberFormatException
        throw new FederateInternalError(e.getMessage());
    }
}
public static long untanhl(RTIambassador rti, DimensionHandle dimension, long normalized, long domainCentre, long domainSize) throws InvalidDimensionHandle, FederateNotExecutionMember, RTIinternalError, FederateInternalError
{
    try
    {
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        double n = Double.valueOf(Long.toString(normalized)).doubleValue();
        double dc = Double.valueOf(Long.toString(domainCentre)).doubleValue();
        double ds = Double.valueOf(Long.toString(domainSize)).doubleValue();
        return ceiling(dc + ds * atanh((2.0 * n / dub) - 1.0));
    } catch (Exception e) {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Hyperbolic tangent normalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension unto which to normalize the domain
 * @param domainCentre the floating-point value of domain around which the user desires the greatest precision
 * @param domainSize a generic measure of the distance around the domainCentre for which the user desires the relatively high precision
 * @param domain a floating-point value known to the federate using the dimension
 * @return a long falling within the dimension's {@link RangeBounds}
 * @throws InvalidDimensionHandle if the {@link DimensionHandle} is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */
public static long tanh(RTIambassador rti,
    DimensionHandle dimension,
    double domainCentre,
    double domainSize,
    double domain)
    throws InvalidDimensionHandle,
        FederateNotExecutionMember,
        RTIinternalError,
        FederateInternalError
{
    try
    {
        // [tanh((domain - domainCenter)/domainSize) + 1]/2 * (DUB - 1)
        double dub = Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
        return new Double(dub * (tanh((domain-domainCentre)/domainSize) + 1.0)/2.0).longValue();
    }
    catch (Exception e) //NumberFormatException
    {
        throw new FederateInternalError(e.getMessage());
    }
}
/**
 * Hyperbolic tangent unnormalization function for floating-point values.
 * @param rti the RTIambassador to use
 * @param dimension a DimensionHandle specifying the dimension whence the normalized value comes
 * @param normalized a long falling within the dimension's [link RangeBounds]
 * @param domainCentre the floating-point value of the domain around which the user desires the greatest precision
 * @param domainSize a generic measure of the distance around the domainCentre for which the user desires the relatively high precision
 * @return a non-enumerated integer value known to the federate using the dimension
 * @throws InvalidDimensionHandle if the [link DimensionHandle] is invalid
 * @throws FederateNotExecutionMember if the federate is not currently joined to a federation execution
 * @throws RTIinternalError if something else goes wrong within an RTIambassador call
 * @throws FederateInternalError if something goes wrong with the rest of the method
 */

public static double
untanh(RTIambassador rti,
     DimensionHandle dimension,
     long normalized,
     double domainCentre,
     double domainSize)
     throws InvalidDimensionHandle,
         FederateNotExecutionMember,
         RTIinternalError,
         FederateInternalError

try
{
    // domainCentre + domainSize*atanh([2*normalized/(DUB - 1)] - 1)
    double dub =
        Double.valueOf(Long.toString(rti.getDimensionUpperBound(dimension) - 1)).doubleValue();
    double n =
        Double.valueOf(Long.toString(normalized)).doubleValue();
    return domainCentre + domainSize*atanh((2.0*n/dub) - 1.0);
} catch (Exception e) { //NumberFormatException
    throw new FederateInternalError(e.getMessage());
}
// Support methods

/**
 * Returns the ceiling of the integer division of the dividend by the divisor.
 * The ceiling is defined as the smallest integer to be larger than or equal to the real quotient.
 * Thus ceiling(3, 2) is 2 but ceiling(-3, 2) and ceiling(3, -2) is -1
 * @param dividend a BigInteger dividend
 * @param divisor a BigInteger divisor
 * @return a BigInteger representing the ceiling of the quotient
 * @throws ArithmeticException if the divisor is BigInteger.ZERO
 */
public static BigInteger
   ceiling(BigInteger dividend,
           BigInteger divisor)
   throws ArithmeticException
{
   // In Java, division of a negative by a positive yields a negative (or zero) remainder
   // division of a positive by a negative yields a positive (or zero) remainder
   BigInteger[] ab = dividend.divideAndRemainder(divisor);
   if ((divisor.signum()*ab[1].signum()) > 0)
   { return ab[0].add(BigInteger.ONE); }
   else
   { return ab[0]; }
}

/**
 * Returns the floor of the integer division of the dividend by the divisor.
 * The floor is defined as the largest integer to be smaller than or equal to the real quotient.
 * Thus floor(3, 2) is 1 but floor(-3, 2) and floor(3, -2) is -2
 * @param dividend a BigInteger dividend
 * @param divisor a BigInteger divisor
 * @return a BigInteger representing the floor of the quotient
 * @throws ArithmeticException if the divisor is BigInteger.ZERO
 */
public static BigInteger
   floor(BigInteger dividend,
          BigInteger divisor)
   throws ArithmeticException
{
   // In Java, division of a negative by a positive yields a negative (or zero) remainder
   // division of a positive by a negative yields a positive (or zero) remainder
   BigInteger[] ab = dividend.divideAndRemainder(divisor);
   if ((divisor.signum()*ab[1].signum()) < 0)
   { return ab[0].subtract(BigInteger.ONE); }
   else
   { return ab[0]; }
}
/**
 * Returns the ceiling of the specified double as a long.
 * The ceiling is defined as the smallest integer to be larger than
 * or equal to the specified double.
 * Thus ceiling(1.5) is 2 but ceiling(-1.5) is -1
 * @param aDouble a double value to convert
 * @return a long representing the ceiling of the specified double
 * @throws ArithmeticException if the double is Double.NaN,
 *        Double.NEGATIVE_INFINITY, Double.POSITIVE_INFINITY, or outside the
 *        range Long.MIN_VALUE to Long.MAX_VALUE
 */
public static long
ceiling(double aDouble)
throws ArithmeticException
{
    if (Double.isNaN(aDouble) || Double.isInfinite(aDouble) || (aDouble > (double)Long.MAX_VALUE) || (aDouble <
        (double)Long.MIN_VALUE))
        throw new ArithmeticException();
    Double d = new Double(aDouble);
    //d.longValue() truncates, so...
    //...We convert the resulting long back into a double:
    Double dd = new Double(new Long(d.longValue()).toString());
    double dd = (double)d.longValue();
    //If the original double is larger, there is some remainder
    if (aDouble > dd)
        [ return d.longValue()+1; ]
    else
        [ return d.longValue(); ]
}

/**
 * Returns the hyperbolic tangent of the specified value.
 * Weirdly, this method isn't supplied by java.lang.Math.
 * <p>
 * \[ \tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} = \frac{(\exp(2z)-1)}{(\exp(2z)+1)} \]
 * @param z a double specifying the value whose hyperbolic tangent
 * is desired
 * @return a double specifying the hyperbolic tangent of the
 * specified value
 */
public static double
tanh(double z)
{
    double r = java.lang.Math.exp(z*z);
    return (r - 1.0)/(r + 1.0);
}
/**
 * Returns the hyperbolic arc-tangent of the specified value.
 * Weirdly, this method isn't supplied by java.lang.Math.
 * 
 * @param z a double specifying the value whose hyperbolic arc-
 * tangent is desired
 * @return a double specifying the hyperbolic arc-tangent of the
 * specified value
 */

public static double
atanh(double z)
{
    return java.lang.Math.log((1+z)/(1-z))/2.0;
}

/**
 * Unwraps the primitive contained in the specified Object.
 * @param o an Object which should be a wrapper around a primitive
 * type
 * @return a long representing the primitive wrapped by the
 * specified Object
 * @throws ClassCastException if the Object does not wrap a
 * primitive type (or it wraps the Void.TYPE)
 */

public static long
unwrap(Object o)
    throws ClassCastException
{
    //Must unwrap the primitive
    Class c = o.getClass();
    if (c.equals(Boolean.TYPE))
        return ((Boolean) o).booleanValue() ? 1 : 0;
    else if (c.equals(Character.TYPE))
        return (Character) o).charValue();
    else if (c.equals(Byte.TYPE))
        return (Byte) o).longValue();
    else if (c.equals(Short.TYPE))
        return (Short) o).longValue();
    else if (c.equals(Integer.TYPE))
        return (Integer) o).longValue();
    else if (c.equals(Long.TYPE))
        return (Long) o).longValue();
    else if (c.equals(Float.TYPE))
        return (Float) o).longValue();
    else if (c.equals(Double.TYPE))
        return (Double) o).longValue();
    else //includes Void.TYPE and all non-primitive others
        throw new ClassCastException();
}

//end Normalization
The `ca.gc.drde_rddc.hla.rti1516.FedAmb` package consists of three sets of classes. The first (`Validate...`) defines callback validation interfaces, allowing callback handling to be broken down into a validation step followed by the event handling proper. The second (`FederateAmbassador...` and `FedAmb...`) breaks up the `FederateAmbassador` interface so that it can be implemented piecemeal. In particular, this allows the federate to dynamically change parts of its implementation during execution, by allocating responsibility to distinct event handlers.

The third, `NativeFederateAmbassador`, which is not included here because it is somewhat out of the scope of this work, is a utility class designed to facilitate the implementation of `FederateAmbassador` as a native class. That is to say, it allows other languages than Java to integrate themselves into a Java-mediated HLA federation by supplying a library (a Windows DLL or a Unix SO). The “foreign” application is expected to access the RTI ambassador through the Java Native Interface (JNI), using the `NativeFederateAmbassador` class to hook its library in. This library is invoked by the federate service thread during `FederateAmbassador` callbacks; after doing validation as required, the library then hands off any remaining processing to another thread, just like in Java. Under Windows, this hand-off can be achieved through messaging. A Delphi 7 prototype demonstrator is available upon request.

In implementing the `FederateAmbassador` interface, keeping all of the code in a single class is extremely unwieldy. In any case, because of the ConcurrentAccess limitation, the amount of actual work done by the class invoked within the federate service thread by the RTI is perforce quite limited. For a given callback, the only processing that must occur immediately is the callback validation, because the RTI expects various exceptions to be thrown by the federate ambassador if things appear awry to the federate. Once that validation is done, any lengthy work is best threaded off to another part of the federate, and such hand-off becomes imperative if RTI ambassador calls are to be made as part of the event handling.

A frequent design pattern in graphical user-interfaces (GUI) applications is to have handlers attached to events. Typically, an event corresponds to a user action, such as button click, but events can also correspond to internal state changes or operating system messages. Elements of the GUI can be re-used when the context changes, simply by changing some of their aspect (e.g. a caption changes from “Log On” to “Log Off”) and re-assigning one or more event handlers. This avoids having a single larger handler begin with inefficient and repetitive context detection code (if such and such flags are set, etc.), and simplifies code maintenance.
In similar fashion, the `FederateAmbassador` interface can be broken down into sub-interfaces (`FederateAmbassador...`), each one regrouping callbacks relevant to a distinct aspect of the HLA. The sub-interface set is complete, in the sense that the `FederateAmbassador` interface is the union of all the sub-interfaces (every `FederateAmbassador` callback occurs in exactly one of the sub-interfaces, and every `FederateAmbassador...` callback occurs in the `FederateAmbassador` interface).

We provide a null-like implementation for each such sub-interface, allowing the designer to override just those few callbacks of interest to him. To facilitate the design further, callbacks that occur in varying forms are “rolled-up” by these implementations into the most complete form, substituting nulls where appropriate.
There is one Validate... interface for each of the FederateAmbassador callbacks. Each interface has but one method, validate, which expects the same arguments as the callback and throws the same exceptions. When the callback is overloaded (InitiateFederateSave, ReceiveInteraction, ReflectAttributeValues, and RemoveObjectInstance), so is the validate method.

// File: ValidateAnnounceSynchronizationPoint.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler. In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateAnnounceSynchronizationPoint {
    /**
     * Validates the announceSynchronizationPoint callback arguments.
     * @param synchronizationPointLabel a String giving the synchronization point's identifier
     * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#announceSynchronizationPoint
     */
    public void validate(String synchronizationPointLabel, byte[] userSuppliedTag)
        throws FederateInternalError;
}
//end ValidateAnnounceSynchronizationPoint
// File: ValidateAttributeIsNotOwned.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateAttributeIsNotOwned
{
    /**
     * Validates the attributeIsNotOwned callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttribute an AttributeHandle specifying the attribute
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#attributeIsNotOwned
     */
    public void validate(
        ObjectInstanceHandle theObject,
        AttributeHandle theAttribute)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            FederateInternalError;
}
//end ValidateAttributeIsNotOwned
// File: ValidateAttributeIsOwnedByRTI.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback
 * (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier}}
 * @version 1.1
 */
public interface ValidateAttributeIsOwnedByRTI
{
    /**
     * Validates the attributeIsOwnedByRTI callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttribute an AttributeHandle specifying the attribute
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#attributeIsOwnedByRTI
     */
    public void validate(
        ObjectInstanceHandle theObject,
        AttributeHandle theAttribute
    ) throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        FederateInternalError;
}
//end ValidateAttributeIsOwnedByRTI
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateAttributeOwnershipAcquisitionNotification
/**
 * Validates the attributeOwnershipAcquisitionNotification callback.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param securedAttributes an AttributeHandleSet specifying the secured attributes
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeAcquisitionWasNotRequested should be thrown if the federate repudiates its attribute ownership acquisition request
 * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
 * @throws AttributeNotPublished should be thrown if the federate denies publishing an attribute
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#attributeOwnershipAcquisitionNotification
 */

public void validate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
    throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAcquisitionWasNotRequested,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError;

//end ValidateAttributeOwnershipAcquisitionNotification
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier])
 * @version 1.1
 */
public interface ValidateAttributeOwnershipUnavailable
{
    /**
     * Validates the attributeOwnershipUnavailable callback.
     * @param theObject the ObjectInstanceHandle of the concerned
     * object instance
     * @param theAttributes an AttributeHandleSet specifying the
     * declined attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate
     * denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the
     * supplied attributes isn't recognized in the supplied context
     * @throws AttributeAlreadyOwned should be thrown if the federate
     * thinks it already owns an attribute
     * @throws AttributeAcquisitionWasNotRequested should be thrown if the
     * federate repudiates its attribute ownership acquisition request
     * @throws FederateInternalError should be thrown if something else
     * is wrong
     * @see hla.rti1516.FederateAmbassador#attributeOwnershipUnavailable
     */
    public void validate(
            ObjectInstanceHandle theObject,
            AttributeHandleSet theAttributes)
            throws ObjectInstanceNotKnown,
                    AttributeNotRecognized,
                    AttributeAlreadyOwned,
                    AttributeAcquisitionWasNotRequested,
                    FederateInternalError;

    //end ValidateAttributeOwnershipUnavailable
// File: ValidateAttributesInScope.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateAttributesInScope
{
    /**
     * Validates the attributesInScope callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttributes an AttributeHandleSet specifying the pertinent attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn’t recognized in the supplied context
     * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#attributesInScope
     */
    public void validate(
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
        throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError;
}

//end ValidateAttributesInScope
package ca.gc.drdc.rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public interface ValidateAttributesOutOfScope {

    /**
     * Validates the attributesOutOfScope callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttributes an AttributeHandleSet specifying the pertinent attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#attributesOutOfScope
     */
    public void validate(ObjectInstanceHandle theObject, AttributeHandleSet theAttributes)
            throws ObjectInstanceNotKnown, AttributeNotRecognized, AttributeNotSubscribed, FederateInternalError;
}

// end ValidateAttributesOutOfScope
package ca.gc.drdc.rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} (http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier)
 * @version 1.1
 */
public interface ValidateConfirmAttributeOwnershipAcquisitionCancellation {

    /**
     * Validates the confirmAttributeOwnershipAcquisitionCancellation
     * callback.
     * @param theObject the ObjectInstanceHandle of the concerned
     * object instance
     * @param theAttributes an AttributeHandleSet specifying the
     * subject attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate
     * denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the
     * supplied attributes isn't recognized in the supplied context
     * @throws AttributeAlreadyOwned should be thrown if the federate
     * thinks it already owns an attribute
     * @throws AttributeAcquisitionWasNotCanceled should be thrown if
     * the federate repudiates the attribute ownership acquisition
     * cancellation
     * @throws FederateInternalError should be thrown if something else
     * is wrong
     * @see
     * hla.rti1516.FederateAmbassador#confirmAttributeOwnershipAcquisitionCan
     * cellation
     */
    public void validate( 
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeAlreadyOwned,
            AttributeAcquisitionWasNotCanceled,
            FederateInternalError;
}

//end ValidateConfirmAttributeOwnershipAcquisitionCancellation
package ca.gc.drdc_rddc.hla.rti516.FedAmb;

import hla.rti516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateDiscoverObjectInstance
{
    /**
     * Validates the discoverObjectInstance callback.
     * @param theObject the ObjectInstanceHandle of the newly discovered object instance
     * @param theObjectClass the ObjectClassHandle of the class the instance was discovered as
     * @param objectName a String holding the newly discovered object instance's name
     * @throws CouldNotDiscover should be thrown if the object instance could not be discovered for some reason other than an unrecognized object class
     * @throws ObjectClassNotRecognized should be thrown if the federate does not recognize <code>theObjectClass</code>
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti516.FederateAmbassador#discoverObjectInstance
     */
    public void validate(
            ObjectInstanceHandle theObject,
            ObjectClassHandle theObjectClass,
            String objectName)
            throws CouldNotDiscover,
                    ObjectClassNotRecognized,
                    FederateInternalError;
}

//end ValidateDiscoverObjectInstance
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault
 * @version 1.1
 */

public interface ValidateFederationNotRestored
{
    /**
     * Validates the federationNotRestored callback.
     * @param reason a RestoreFailureReason specifying the reason for the failure
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationNotRestored
     */
    public void validate(RestoreFailureReason reason) throws FederateInternalError;
}

//end ValidateFederationNotRestored
// File: ValidateFederationNotSaved.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public interface ValidateFederationNotSaved {

    /**
     * Validates the federationNotSaved callback.
     * @param reason a SaveFailureReason specifying why the save failed
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationNotSaved
     */
    public void validate(SaveFailureReason reason) throws FederateInternalError;
}

//end ValidateFederationNotSaved
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({{link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier}})
 * @version 1.1
 */
public interface ValidateFederationRestoreBegun
{
    /**
     * Validates the federationRestoreBegun callback.
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationRestoreBegun
     */
    public void validate() throws FederateInternalError;
}

//end ValidateFederationRestoreBegun
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */

public interface ValidateFederationRestored {
    /**
     * Validates the federationRestored callback.
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationRestored
     */
    public void validate() throws FederateInternalError;
}

// File: ValidateFederationRestored.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateFederationRestoreStatusResponse
{
    /**
     * Validates the federationRestoreStatusResponse callback.
     * @param response a FederateHandleRestoreStatusPair[] specifying the RestoreStatus of each federate
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationRestoreStatusResponse
     */
    public void validate(FederateHandleRestoreStatusPair[] response)
        throws FederateInternalError;
}

//end ValidateFederationRestoreStatusResponse
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler. In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well. 
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public interface ValidateFederationSaved
{
    /**
     * Validates the federationSaved callback.
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#federationSaved
     */
    public void validate() throws FederateInternalError;
}

//end ValidateFederationSaved
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public interface ValidateFederationSaveStatusResponse
{
    /**
     * Validates the federationSaveStatusResponse callback.
     * @param response a FederateHandleSaveStatusPair[] specifying the
     * SaveStatus of each federate
     * @throws FederateInternalError should be thrown if something is
     * wrong
     * @see hla.rti1516.FederateAmbassador#federationSaveStatusResponse
     */
    public void validate(FederateHandleSaveStatusPair[] response)
        throws FederateInternalError;
}

//end ValidateFederationSaveStatusResponse
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier})
 * @version 1.1
 */
public interface ValidateFederationSynchronized
{
    /**
     * Validates the federationSynchronized callback arguments.
     * @param synchronizationPointLabel a String giving the
     * synchronization point's identifier
     * @throws FederateInternalError should be thrown if something is
     * wrong
     * @see hla.rti1516.FederateAmbassador#federationSynchronized
     */
    public void validate(String synchronizationPointLabel)
        throws FederateInternalError;
}
//end ValidateFederationSynchronized
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are
 * called first by the callback handler. In each case, there is but one method:
 * validate(), which accepts the same arguments as the corresponding callback
 * (overloaded as needed). Likewise, the method throws the same exceptions as
 * the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public interface ValidateInformAttributeOwnership {

    /**
     * Validates the informAttributeOwnership callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttribute an AttributeHandle specifying the attribute
     * @param theOwner the FederateHandle of the federate owning the attribute
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies
     * having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't
     * recognized in the supplied context
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#informAttributeOwnership
     */
    public void validate(ObjectInstanceHandle theObject, AttributeHandle theAttribute, FederateHandle theOwner)
        throws ObjectInstanceNotKnown, AttributeNotRecognized, FederateInternalError;
}

//end ValidateInformAttributeOwnership
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateInitiateFederateRestore
{
    /**
     * Validates the initiateFederateRestore callback.
     * @param label a String holding the saved state's identifier
     * @param federateHandle the FederateHandle that the federate will assume if and once it receives the federationRestored callback
     * @throws SpecifiedSaveLabelDoesNotExist should be thrown if the label isn't recognized
     * @throws CouldNotInitiateRestore should be thrown if the federate is unwilling or unable to initiate the restore operation
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#initiateFederateRestore
     */
    public void validate(String label, FederateHandle federateHandle) throws SpecifiedSaveLabelDoesNotExist, CouldNotInitiateRestore, FederateInternalError;
}

//end ValidateInitiateFederateRestore
import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault
 * @version 1.1
 */

public interface ValidateInitiateFederateSave
{
    // Validates the initiateFederateSave callback arguments (timeless form).
    // @param label a String holding the saved state's identifier
    // @throws UnableToPerformSave should be thrown if the save operation seems doomed
    // @throws FederateInternalError should be thrown if something else is wrong
    // @see hla.rti1516.FederateAmbassador#initiateFederateSave
    public void validate(String label)
        throws UnableToPerformSave,
                FederateInternalError;
}
/**
 * Validates the initiateFederateSave callback arguments (timefull form).
 * @param label a String holding the saved state's identifier
 * @param time a LogicalTime specifying when the save was scheduled
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
 * @throws UnableToPerformSave should be thrown if the save operation seems doomed
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#initiateFederateSave
 */
 public void validate(
     String label,
     LogicalTime time)
 throws InvalidLogicalTime,
     UnableToPerformSave,
     FederateInternalError;
}
//end ValidateInitiateFederateSave
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler. In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateObjectInstanceNameReservationFailed {
    /**
     * Validates the objectInstanceNameReservationFailed callback.
     * @param objectName a String holding the requested object name
     * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#objectInstanceNameReservationFailed
     */
    public void validate(String objectName) throws UnknownName, FederateInternalError;
}

// end ValidateObjectInstanceNameReservationFailed
// File: ValidateObjectInstanceNameReservationSucceeded.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateObjectInstanceNameReservationSucceeded {
  /**
   * Validates the objectInstanceNameReservationSucceeded callback.
   * @param objectName a String holding the requested object name
   * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
   * @throws FederateInternalError should be thrown if something else is wrong
   * @see hla.rti1516.FederateAmbassador#objectInstanceNameReservationSucceeded
   */
  public void validate(String objectName) throws UnknownName, FederateInternalError;
}

//end ValidateObjectInstanceNameReservationSucceeded
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault (DaniellThibault@DRDC-RDDC.gc.ca) (http://www.valcartier.drdc-rddc.gc.ca)
 * @version 1.1
 */
public interface ValidateProvideAttributeValueUpdate
{
    /**
     * Validates the provideAttributeValueUpdate callback.
     * @param theObject the ObjectInstanceHandle of the subject object instance
     * @param theAttributes an AttributeHandleSet specifying the requested attributes
     * @param userSuppliedTag a byte[] tag (this parameter may be null)
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#provideAttributeValueUpdate
     */
    public void validate(ObjectInstanceHandle theObject, AttributeHandleSet theAttributes, byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown, AttributeNotRecognized, AttributeNotOwned, FederateInternalError;
}

//end ValidateProvideAttributeValueUpdate
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are
called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault
 * @version 1.1
 */
public interface ValidateReceiveInteraction
{
    /**
     * Validates the receiveInteraction callback (base form).
     * @param interactionClass the InteractionClassHandle of the received interaction
     * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
     * @param userSuppliedTag a byte[] tag (this parameter may be null)
     * @param sentOrdering the OrderType the interaction was sent as
     * @param theTransport the TransportationType used to send the interaction
     * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
     * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
     * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#receiveInteraction
     */
    public void validate(
        InteractionClassHandle interactionClass,
        ParameterHandleValueMap theParameters,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport)
        throws InteractionClassNotRecognized,
                    InteractionParameterNotRecognized,
                    InteractionClassNotSubscribed,
                    FederateInternalError;
}
/**
 * Validates the receiveInteraction callback (second form).
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param sentRegions the RegionHandleSet used to send the interaction
 * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
 * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
 * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#receiveInteraction
 */
public void validate(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
           InteractionParameterNotRecognized,
           InteractionClassNotSubscribed,
           FederateInternalError;
/**
 * Validates the receiveInteraction callback (third form).
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param theTime the LogicalTime at which the interaction occurs
 * @param receivedOrdering the OrderType the passel was received as
 * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
 * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn’t recognized in the supplied context
 * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#receiveInteraction
 */
public void validate(
        InteractionClassHandle interactionClass,
        ParameterHandleValueMap theParameters,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport,
        LogicalTime theTime,
        OrderType receivedOrdering)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError;
/**
 * Validates the receiveInteraction callback (fourth form).
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param theTime the LogicalTime at which the interaction occurs
 * @param receivedOrdering the OrderType the passel was received as
 * @param sentRegions the RegionHandleSet used to send the interaction
 * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
 * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
 * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#receiveInteraction
 */
public void validate(
  InteractionClassHandle interactionClass,
  ParameterHandleValueMap theParameters,
  byte[] userSuppliedTag,
  OrderType sentOrdering,
  TransportationType theTransport,
  LogicalTime theTime,
  OrderType receivedOrdering,
  RegionHandleSet sentRegions)
  throws InteractionClassNotRecognized,
  InteractionParameterNotRecognized,
  InteractionClassNotSubscribed,
  FederateInternalError;

/**
 * Validates the receiveInteraction callback (fifth form).
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param theTime the LogicalTime at which the interaction occurs
 * @param receivedOrdering the OrderType the passel was received as
 * @param messageRetractionHandle the MessageRetractionHandle of the message
 * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
 * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
 * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid (only if the <code>receivedOrdering</code> is TIMESTAMP)
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#receiveInteraction
 */
public void validate(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
    throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;
/**
 * Validates the receiveInteraction callback (full form).
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param theTime the LogicalTime at which the interaction occurs
 * @param receivedOrdering the OrderType the passel was received as
 * @param messageRetractionHandle the MessageRetractionHandle of the message
 * @param sentRegions the RegionHandleSet used to send the interaction
 * @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
 * @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
 * @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
 * @throws InvalidLogicalTime should be thrown if the specified LogicalTime is invalid (only if the receivedOrdering is TIMESTAMP)
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#receiveInteraction
 */
public void validate(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;
} //end ValidateReceiveInteraction
// File: ValidateReflectAttributeValues.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */

public interface ValidateReflectAttributeValues
{
    /**
     * Validates the reflectAttributeValues callback (base form).
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param theAttributes an AttributeHandleValueMap specifying the new attribute values
     * @param userSuppliedTag a byte[] tag (this parameter may be <code>null</code>)
     * @param sentOrdering the OrderType the passel was sent as
     * @param theTransport the TransportationType used to send the passel
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
     */

    public void validate(
            ObjectInstanceHandle theObject,
            AttributeHandleValueMap theAttributes,
            byte[] userSuppliedTag,
            OrderType sentOrdering,
            TransportationType theTransport)
            throws ObjectInstanceNotKnown,
                    AttributeNotRecognized,
                    AttributeNotSubscribed,
                    FederateInternalError;
/**
 * Validates the reflectAttributeValues callback (second form).
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleValueMap specifying the new attribute values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the OrderType the passel was sent as
 * @param sentRegions the RegionHandleSet used to send the passel
 * @param theTransport the TransportationType used to send the passel
 * @param sentOrdering the OrderType the passel was sent as
 * @param sentRegions the RegionHandleSet used to send the update
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
 */

public void validate(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            FederateInternalError;
/**
 * Validates the reflectAttributeValues callback (third form).
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleValueMap specifying the new attribute values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the OrderType the passel was sent as
 * @param theTransport the TransportationType used to send the passel
 * @param theTime the LogicalTime at which the update comes into effect
 * @param receivedOrdering the OrderType the passel was received as
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
 */

public void validate(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            FederateInternalError;
/**
 * Validates the reflectAttributeValues callback (fourth form).
 * @param theObject the ObjectInstanceHandle of the concerned
 * object instance
 * @param theAttributes an AttributeHandleValueMap specifying the
 * new attribute values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @param sentOrdering the OrderType the passel was sent as
 * @param theTransport the TransportationType used to send the
 * passel
 * @param theTime the LogicalTime at which the update comes into
 * effect
 * @param receivedOrdering the OrderType the passel was received as
 * @param sentRegions the RegionHandleSet used to send the update
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate
 * denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else
 * is wrong
 * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
 */
public void
validate(
    ObjectInstanceHandle  theObject,
    AttributeHandleValueMap theAttributes,
    byte[]     userSuppliedTag,
    OrderType  sentOrdering,
    TransportationType  theTransport,
    LogicalTime  theTime,
    OrderType  receivedOrdering,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
/**
 * Validates the reflectAttributeValues callback (fifth form).
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleValueMap specifying the new attribute values
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the passel was sent as
 * @param theTransport the TransportationType used to send the passel
 * @param theTime the LogicalTime at which the update comes into effect
 * @param receivedOrdering the OrderType the passel was received as
 * @param retractionHandle the MessageRetractionHandle of the message
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws InvalidLogicalTime should be thrown if the specified logical time is invalid (only if the receivedOrdering is TIMESTAMP)
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
 */
public void
validate(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;

/**
 * Validates the reflectAttributeValues callback (full form).
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleValueMap specifying the new attribute values
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
 * @param sentOrdering the OrderType the passel was sent as
 * @param theTransport the TransportationType used to send the passel
 * @param theTime the LogicalTime at which the update comes into effect
 * @param receivedOrdering the OrderType the passel was received as
 * @param retractionHandle the MessageRetractionHandle of the message
 * @param sentRegions the RegionHandleSet used to send the update
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid (only if the <code>receivedOrdering</code> is <code>TIMESTAMP</code>)
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#reflectAttributeValues
 */

public void validate(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        InvalidLogicalTime,
        FederateInternalError;
}

//end ValidateReflectAttributeValues
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these
 * are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier])
 * @version 1.1
 */
public interface ValidateRemoveObjectInstance {

  /**
   * Validates the removeObjectInstance callback (base form).
   * @param theObject the ObjectInstanceHandle of the concerned
   * object instance
   * @param userSuppliedTag a byte[] tag (this parameter may be null)
   * @param sentOrdering the OrderType the message was sent as
   * @throws ObjectInstanceNotKnown should be thrown if the federate
denies having previously discovered the object instance
   * @throws FederateInternalError should be thrown if something else
   * is wrong
   * @see hla.rti1516.FederateAmbassador#removeObjectInstance
   */
  public void validate(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering)
    throws ObjectInstanceNotKnown,
    FederateInternalError;
@**
* Validates the removeObjectlnstance callback (second form).
* @param theObject the ObjectlnstanceHandle of the concerned object instance
* @param userSuppliedTag a byte[] tag (this parameter may be null)
* @param sentOrdering the OrderType the message was sent as
* @param theTime the LogicalTime at which the deletion occurs
* @param receivedOrdering the OrderType the message was received as
* @throws ObjectlnstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws FederateInternalError should be thrown if something else is wrong
* @see hla.rti1516.FederateAmbassador#removeObjectInstance
*/
public void validate(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectlnstanceNotKnown,
    FederateInternalError;
/**
   * Validates the removeObjectlnstance callback (full form).
   * @param theObject the ObjectlnstanceHandle of the concerned
   * object instance
   * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
   * may be <code>null</code>)
   * @param sentOrdering the OrderType the message was sent as
   * @param theTime the LogicalTime at which the deletion occurs
   * @param receivedOrdering the OrderType the message was received
   * as
   * @param retractionHandle the MessageRetractionHandle of the
   * message
   * @throws ObjectlnstanceNotKnown should be thrown if the federate
   * denies having previously discovered the object instance
   * @throws InvalidLogicalTime should be thrown if the specified
   * <code>LogicalTime</code> is invalid (only if the
   * <code>receivedOrdering</code> is TIMESTAMP)
   * @throws FederateInternalError should be thrown if something else
   * is wrong
   * @see hla.rti1516.FederateAmbassador#removeObjectInstance
   */
   public void
   validate(
      ObjectInstanceHandle theObject,
      byte[] userSuppliedTag,
      OrderType sentOrdering,
      LogicalTime theTime,
      OrderType receivedOrdering,
      MessageRetractionHandle retractionHandle)
      throws ObjectlnstanceNotKnown,
         InvalidLogicalTime,
         FederateInternalError;
   }
   //end ValidateRemoveObjectInstance
import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the
 * callback handler. In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as needed). Likewise, the method
 * throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */

public interface ValidateRequestAttributeOwnershipAssumption
{
    /**
     * Validates the requestAttributeOwnershipAssumption callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param offeredAttributes an AttributeHandleSet specifying the offered attributes
     * @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
     * @throws AttributeNotPublished should be thrown if the federate denies publishing an attribute
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#requestAttributeOwnershipAssumption
     */
    public void validate(
        ObjectInstanceHandle theObject,
        AttributeHandleSet offeredAttributes,
        byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeAlreadyOwned,
                AttributeNotPublished,
                FederateInternalError;

    //end ValidateRequestAttributeOwnershipAssumption
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1 */

public interface ValidateRequestAttributeOwnershipRelease {

/**
 * Validates the requestAttributeOwnershipRelease callback.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param candidateAttributes an AttributeHandleSet specifying the candidate attributes
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#requestAttributeOwnershipRelease */

public void validate(ObjectInstanceHandle theObject, AttributeHandleSet candidateAttributes, byte[] userSuppliedTag)
    throws ObjectInstanceNotKnown, AttributeNotRecognized, AttributeNotOwned, FederateInternalError;

//end ValidateRequestAttributeOwnershipRelease
// File: ValidateRequestDivestitureConfirmation.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (){http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateRequestDivestitureConfirmation {

    /**
     * Validates the requestDivestitureConfirmation callback.
     * @param theObject the ObjectInstanceHandle of the concerned object instance
     * @param offeredAttributes an AttributeHandleSet specifying the offered attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
     * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
     * @throws AttributeDivestitureWasNotRequested should be thrown if the federate repudiates the divestiture
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#requestDivestitureConfirmation
     */
    public void validate (ObjectInstanceHandle theObject, AttributeHandleSet offeredAttributes)
        throws ObjectInstanceNotKnown, AttributeNotRecognized, AttributeNotOwned, AttributeDivestitureWasNotRequested, FederateInternalError;

    //end ValidateRequestDivestitureConfirmation
// File: ValidateRequestFederationRestoreFailed.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateRequestFederationRestoreFailed {
    /**
     * Validates the requestFederationRestoreFailed callback.
     * @param label a String holding the saved state's identifier
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#requestFederationRestoreFailed
     */
    public void validate(String label) throws FederateInternalError;
}

//end ValidateRequestFederationRestoreFailed
package ca.gc.drdoc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault (RDDC Valcartier)
 * @version 1.1
 */
public interface ValidateRequestFederationRestoreSucceeded
{
    /**
     * Validates the requestFederationRestoreSucceeded callback.
     * @param label a String holding the saved state's identifier
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#requestFederationRestoreSucceeded
     */
    public void validate(String label)
        throws FederateInternalError;
}
//end ValidateRequestFederationRestoreSucceeded
// File: ValidateRequestRetraction.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateRequestRetraction
{

    /**
     * Validates the requestRetraction callback.
     * @param theHandle the MessageRetractionHandle specifying the
     * retracted message
     * @throws FederateInternalError should be thrown if something is
     * wrong
     * @see hla.rti1516.FederateAmbassador#requestRetraction
     */
    public void validate(
        MessageRetractionHandle theHandle
    ) throws FederateInternalError;
}

//end ValidateRequestRetraction
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler. In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateStartRegistrationForObjectClass {
  /**
   * Validates the startRegistrationForObjectClass callback.
   * @param theClass the ObjectClassHandle of the subject object class
   * @throws ObjectClassNotPublished should be thrown if the federate denies publishing <code>theClass</code>
   * @throws FederateInternalError should be thrown if something else is wrong
   * @see hla.rti1516.FederateAmbassador#startRegistrationForObjectClass
   */
  public void validate(ObjectClassHandle theClass)
      throws ObjectClassNotPublished, FederateInternalError;
}

// File: ValidateStartRegistrationForObjectClass.java

//end ValidateStartRegistrationForObjectClass
// File: ValidateStopRegistrationForObjectClass.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface ValidateStopRegistrationForObjectClass {
    /**
     * Validates the stopRegistrationForObjectClass callback.
     * @param theClass the ObjectClassHandle of the subject object class
     * @throws ObjectClassNotPublished should be thrown if the federate denies publishing <code>theClass</code>
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#stopRegistrationForObjectClass
     */
    public void validate(ObjectClassHandle theClass)
        throws ObjectClassNotPublished, FederateInternalError;
}

//end ValidateStopRegistrationForObjectClass
// File: ValidateSynchronizationPointRegistrationFailed.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (mailto:http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier)
 * @version 1.1
 */

public interface ValidateSynchronizationPointRegistrationFailed
{
    /**
     * Validates the synchronizationPointRegistrationFailed callback arguments.
     * @param synchronizationPointLabel a String giving the synchronization point's identifier
     * @param reason a SynchronizationPointFailureReason specifying what went wrong
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#synchronizationPointRegistrationFailed
     */
    public void validate(String synchronizationPointLabel, SynchronizationPointFailureReason reason) throws FederateInternalError;
}

//end ValidateSynchronizationPointRegistrationFailed
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */

public interface ValidateSynchronizationPointRegistrationSucceeded
{
    /**
     * Validates the SynchronizationPointRegistrationSucceeded callback arguments.
     * @param synchronizationPointLabel a String giving the synchronization point's identifier
     * @throws FederateInternalError should be thrown if something is wrong
     * @see hla.rti1516.FederateAmbassador#synchronizationPointRegistrationSucceeded
     */
    public void validate(String synchronizationPointLabel)
        throws FederateInternalError;
}

//end ValidateSynchronizationPointRegistrationSucceeded
// File: ValidateTimeAdvanceGrant.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateTimeAdvanceGrant
{
    /**
     * Validates the timeAdvanceGrant callback.
     * @param theTime the LogicalTime to which the federate's clock has been set
     * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
     * @throws JoinedFederateIsNotInTimeAdvancingState should be thrown if the federate does not consider itself in the time-advancing state
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#timeAdvanceGrant
     */
    public void validate
    (LogicalTime theTime)
    throws InvalidLogicalTime,
    JoinedFederateIsNotInTimeAdvancingState,
    FederateInternalError;
}

//end ValidateTimeAdvanceGrant
// File: ValidateTimeConstrainedEnabled.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier])
 * @version 1.1
 */
public interface ValidateTimeConstrainedEnabled
{
    /**
     * Validates the timeConstrainedEnabled callback.
     * @param time the LogicalTime to which the federate's clock has
     * been set
     * @throws InvalidLogicalTime should be thrown if the specified
     * <code>LogicalTime</code> is invalid
     * @throws NoRequestToEnableTimeConstrainedWasPending should be
     * thrown if the federate repudiates the time constraint request
     * @throws FederateInternalError should be thrown if something else
     * is wrong
     * @see hla.rti1516.FederateAmbassador#timeConstrainedEnabled
     */
    public void validate(
            LogicalTime time)
            throws InvalidLogicalTime,
            NoRequestToEnableTimeConstrainedWasPending,
            FederateInternalError;
}
//end ValidateTimeConstrainedEnabled
// File: ValidateTimeRegulationEnabled.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier)
 * @version 1.1
 */
public interface ValidateTimeRegulationEnabled
{
    /**
     * Validates the timeRegulationEnabled callback.
     * @param time the LogicalTime to which the federate’s clock has been set
     * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
     * @throws NoRequestToEnableTimeRegulationWasPending should be thrown if the federate repudiates the time regulation request
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#timeRegulationEnabled
     */
    public void validate(LogicalTime time) throws InvalidLogicalTime, NoRequestToEnableTimeRegulationWasPending, FederateInternalError;
}

//end ValidateTimeRegulationEnabled
package ca.gc.drdc_rddc_hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier}}
 * @version 1.1
 */

public interface ValidateTurnInteractionsOff
{
    /**
     * Validates the turnInteractionsOff callback.
     * @param theHandle the InteractionClassHandle of the subject interaction class
     * @throws InteractionClassNotPublished should be thrown if the federate denies publishing <code>theHandle</code>
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#turnInteractionsOff
     */
    public void validate(
            InteractionClassHandle theHandle) throws InteractionClassNotPublished, FederateInternalError;
}

//end ValidateTurnInteractionsOff
package ca.gc.drdc_rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author Daniel U. Thibault ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface ValidateTurnInteractionsOn
{
    /**
     * Validates the turnInteractionsOn callback.
     * @param theHandle the InteractionClassHandle of the subject interaction class
     * @throws InteractionClassNotPublished should be thrown if the federate denies publishing <code>theHandle</code>
     * @throws FederateInternalError should be thrown if something else is wrong
     * @see hla.rti1516.FederateAmbassador#turnInteractionsOn
     */
    public void validate(InteractionClassHandle theHandle)
        throws InteractionClassNotPublished, FederateInternalError;
}

//end ValidateTurnInteractionsOn
import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers;
 * these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts
 * the same arguments as the corresponding callback (overloaded as
 * needed). Likewise, the method throws the same exceptions as the
 * callback. If no exception is thrown, all is well.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
 * Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC
 * Valcartier])
 * @version 1.1
 */

public interface ValidateTurnUpdatesOffForObjectInstance
{
    /**
     * Validates the turnUpdatesOffForObjectInstance callback.
     * @param theObject the ObjectInstanceHandle of the subject object
     * instance
     * @param theAttributes an AttributeHandleSet specifying the
     * subject attributes
     * @throws ObjectInstanceNotKnown should be thrown if the federate
     * denies having previously discovered the object instance
     * @throws AttributeNotRecognized should be thrown if one of the
     * supplied attributes isn't recognized in the supplied context
     * @throws AttributeNotOwnned should be thrown if the federate
     * denies owning an attribute
     * @throws FederateInternalError should be thrown if something else
     * is wrong
     * @see hla.rti1516.FederateAmbassador#turnUpdatesOffForObjectInstance
     */
    public void validate(
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
        throws ObjectInstanceNotKnown,
                  AttributeNotRecognized,
                  AttributeNotOwnned,
                  FederateInternalError;
}

//end ValidateTurnUpdatesOffForObjectInstance
package ca.gc.drdc.rddc.hla.rti1516.FedAmbi;

import hla.rti1516.*;

/**
 * This interface is meant to be implemented by validation handlers; these are called first by the callback handler.
 * In each case, there is but one method: validate(), which accepts the same arguments as the corresponding callback (overloaded as needed). Likewise, the method throws the same exceptions as the callback. If no exception is thrown, all is well.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier)
 * @version 1.1
 */
public interface ValidateTurnUpdatesOnForObjectInstance
{

/**
 * Validates the turnUpdatesOnForObjectInstance callback.
 * @param theObject the ObjectInstanceHandle of the subject object instance
 * @param theAttributes an AttributeHandleSet specifying the subject attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
 * @throws FederateInternalError should be thrown if something else is wrong
 * @see hla.rti1516.FederateAmbassador#turnUpdatesOnForObjectInstance
 */
public void validate(  
    ObjectInstanceHandle theObject,  
    AttributeHandleSet theAttributes)  
    throws ObjectInstanceNotKnown,  
       AttributeNotRecognized,  
       AttributeNotOwned,  
       FederateInternalError;
}

//end ValidateTurnUpdatesOnForObjectInstance
The **FederateAmbassadorSynchronization** interface is the part devoted to the federation synchronization callbacks.
/**
 * Notifies the federate that it has failed to registered a
 * federation synchronization point.
 * @param synchronizationPointLabel a {@link java.lang.String}
 * giving the synchronization point's identifier
 * @param reason a {@link SynchronizationPointFailureReason}
 * specifying what went wrong
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador#registerFederationSynchronizationPoint
 * registerFederationSynchronizationPoint
 * @see RTIambassador#synchronizationPointAchieved
 * synchronizationPointAchieved
 * @see #synchronizationPointRegistrationSucceeded
 * synchronizationPointRegistrationSucceeded
 * @see #announceSynchronizationPoint
 * announceSynchronizationPoint
 * @see #federationSynchronized federationSynchronized
 */

public void
synchronizationPointRegistrationFailed(
    String synchronizationPointLabel,
    SynchronizationPointFailureReason reason)
throws FederateInternalError;
public void announceSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag)
throws FederateInternalError;

// 4.8
/**
 * Notifies the federate that a synchronization point exists.
 * Achievement of the point is signalled to the RTI through the
 * @link RTIambassador#synchronizationPointAchieved
 * synchronizationPointAchieved} method.
 * <p>
 * Federates are by default part of the synchronization set, unless
 * a @link FederateHandleSet} has been specifically
 * supplied to the @link
 * RTIambassador#registerFederationSynchronizationPoint(String,byte[] ,Fed
 * erateHandleSet}) method.
 * A federate that resigns is simply removed from the
 * synchronization set.
 * The synchronization point exists until it has been achieved by
 * all concerned federates, which could be
 * until the federation execution concludes.
 * @param synchronizationPointLabel a @link java.lang.String}
giving the synchronization point's identifier
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 */
@see RTIambassador#registerFederationSynchronizationPoint
registerFederationSynchronizationPoint
@see RTIambassador#synchronizationPointAchieved
synchronizationPointAchieved
@see #synchronizationPointRegistrationSucceeded
synchronizationPointRegistrationSucceeded
@see #synchronizationPointRegistrationFailed
synchronizationPointRegistrationFailed
@see #federationSynchronized federationSynchronized
*/
/**
 * Informs the joined federate that all members of the synchronization set of the specified synchronization point
 * have invoked the {\link RTIambassador#synchronizationPointAchieved synchronizationPointAchieved} method for that point.
 * The synchronization point ceases to exist once this callback has been invoked on all concerned federates.
 * @param synchronizationPointLabel a {\link java.lang.String} giving the synchronization point's identifier
 * @throws FederateInternalError should be thrown if something goes wrong
 */
public void federationSynchronized(String synchronizationPointLabel) throws FederateInternalError {
The `FederateAmbassadorSave` interface is the `FederateAmbassador` part devoted to the federation save callbacks.

```java
// File: FederateAmbassadorSave.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;
/
/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Save callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface FederateAmbassadorSave
{ }
```
/**
 * Instructs the joined federate that it is now in the Instructed To Save state
 * and should therefore save state as soon as possible.
 * The joined federate should use the supplied <code>label</code>,
 * the name of the federation execution (see the <code>federationExecutionName</code>)
 * of the [@link RTIambassador#joinFederationExecution joinFederationExecution] invocation,
 * its joined federate designator (returned by the aforementioned invocation) and
 * its federate type (the aforementioned invocation's <code>federateType</code> argument)
 * to distinguish the saved state information.
 * A federate that is not time constrained should expect this callback at any point.
 * A time constrained federate can receive this callback only whilst in the Time Advancing state.
 * Once in the Instructed To Save state, the federate is severely limited in which RTIambassador methods it can invoke. This lasts through the Saving and Waiting For Federation To Save states,
 * concluding with the [@link #federationSaved federationSaved] or [@link #federationNotSaved federationNotSaved] callbacks.
 * @param label a [@link java.lang.String] holding the saved stateTs identifier
 * @throws UnableToPerformSave should be thrown if the save operation seems doomed
 * @throws FederateInternalError should be thrown if something else goes wrong
 */

public void initiateFederateSave(String label)
        throws UnableToPerformSave, FederateInternalError;
/**
 * Instructs the joined federate that it is now in the Instructed To Save state
 * (as of the specified <code>time</code>) and should therefore
 * save state as soon as possible.
 * <p>
 * For details, see the {@link #initiateFederateSave(String)}
callback.
 * @param label a {@link java.lang.String} holding the saved
 * state's identifier
 * @param time a {@link LogicalTime} specifying when the save was
 * scheduled
 * @throws InvalidLogicalTime should be thrown if the specified
 * <code>LogicalTime</code> is invalid
 * @throws UnableToPerformSave should be thrown if the save
 * operation seems doomed
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #federationSaved federationSaved
 * @see #federationNotSaved federationNotSaved
 * @see #federationSaveStatusResponse federationSaveStatusResponse
 */
public void initiateFederateSave(
    String label,
    LogicalTime time)
throws InvalidLogicalTime,
    UnableToPerformSave,
    FederateInternalError;
/**
 * Informs the joined federate that the federation save process is complete and successful.
 * <p>
 * All joined federates at which the [link #initiateFederateSave initiateFederateSave] callback
 * was invoked have in turn invoked the [link RTIambassador#federateSaveComplete federateSaveComplete] method.
 * @throws FederateInternalError should be thrown if something goes wrong.
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #initiateFederateSave initiateFederateSave
 * @see #federationNotSaved federationNotSaved
 * @see #federationSaveStatusResponse federationSaveStatusResponse
 */

public void federationSaved()
throws FederateInternalError;
/**
 * Informs the joined federate that the federation save process has completed in failure.
 * <p>
 * The possible save failure reasons are:
 * <ul>
 * <li> RTI_UNABLE_TO_SAVE: The RTI was unable to save
 * <li> FEDERATE_REPORTED_FAILURE: One or more joined federates have invoked the [\@Link RTIambassador#federateSaveNotComplete federateSaveNotComplete] method
 * <li> FEDERATE_RESIGNED: One or more joined federates have resigned from the federation execution
 * <li> RTI_DETECTED_FAILURE: The RTI has detected failure at one or more of the joined federates
 * <li> SAVE_TIME_CANNOT_BE_HONORED: The time stamp specified by the [\@Link RTIambassador#requestFederationSave(String,LogicalTime)] request cannot be honored, due to possible race conditions in the distributed calculation of GALT (Greatest Available Logical Time)
 * </ul>
 * @param reason a [\@Link SaveFailureReason] specifying why the save failed
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #initiateFederateSave initiateFederateSave
 * @see #federationSaved federationSaved
 * @see #federationSaveStatusResponse federationSaveStatusResponse 
 */ 
public void federationNotSaved(
    SaveFailureReason reason)
throws FederateInternalError;
/**
 * Supplies the federate with the previously requested save status indicators.
 * @param response a [@link FederateHandleSaveStatusPair][20] specifying the [@link SaveStatus][21] of each federate
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationSave requestFederationSave
 * @see RTIambassador#federateSaveBegun federateSaveBegun
 * @see RTIambassador#federateSaveComplete federateSaveComplete
 * @see RTIambassador#federateSaveNotComplete federateSaveNotComplete
 * @see RTIambassador#queryFederationSaveStatus queryFederationSaveStatus
 * @see #initiateFederateSave initiateFederateSave
 * @see #federationSaved federationSaved
 * @see #federationNotSaved federationNotSaved
 */
public void federationSaveStatusResponse(  
FederalHandleSaveStatusPair[] response)  
throws FederateInternalError;  
// end FederateAmbassadorSave
The FederateAmbassadorRestore interface is the FederateAmbassador part devoted to the federation restoration callbacks.

```
// File: FederateAmbassadorRestore.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Restore callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} [http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier]
 * @version 1.1
 */

public interface FederateAmbassadorRestore
{
}
```
public void requestFederationRestoreSucceeded(String label) throws FederateInternalError {
    /* Indicates that the federate's previous {@link RTIambassador#requestFederationRestore requestFederationRestore} has been granted.
     * <p>
     * This means the RTI has located the RTI-specific saved state information matching the previously supplied
     * <code>label</code>, {@link RTIambassador#joinFederationExecution federationExecutionName} and
     * {@link RTIambassador#createFederationExecution fdd}, and that
     * the census of currently joined
     * federates matches in number and {@link RTIambassador#federateType} that of the
     * RTI's saved state.
     * @param label a {@link java.lang.String} holding the saved state's identifier
     * @throws FederateInternalError should be thrown if something goes wrong
     */
    requestFederationRestoreRequestFederationRestore
    federateRestoreComplete
    federateRestoreNotComplete
    queryFederationRestoreStatus
    requestFederationRestoreFailed
    federationRestoreBegun
    initiateFederateRestore
    federationRestored
    federationNotRestored
    federationRestoreStatusResponse
}

throws FederateInternalError;
/**
 * Indicates that the federate's previous [@link RTIambassador#requestFederationRestore requestFederationRestore] has
 been denied.  
 * This means the RTI failed to locate its specific saved state
 information or that the census of
 * currently joined federates does not match in number and [@link RTIambassador#joinFederationExecution federateType]
 * that of the retrieved RTI saved state.
 * Failures by individual federates to complete the restoration
 process lead to the
 * [@link #federationNotRestored federationNotRestored] callback
 instead.
 * @param label a [@link java.lang.String] holding the saved
 state's identifier
 * @throws FederateInternalError should be thrown if something goes
 wrong
 * @see RTIambassador#requestFederationRestore requestFederationRestore
 * @see RTIambassador#federateRestoreComplete federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus queryFederationRestoreStatus
 * @see #requestFederationRestoreSucceeded requestFederationRestoreSucceeded
 * @see #federationRestoreBegun federationRestoreBegun
 * @see #initiateFederateRestore initiateFederateRestore
 * @see #federationRestored federationRestored
 * @see #federationNotRestored federationNotRestored
 * @see #federationRestoreStatusResponse federationRestoreStatusResponse
 */

public void
requestFederationRestoreFailed(
    String label)
throws FederateInternalError;
Instructs the joined federate that it is now in the Prepared To Restore state
and should prepare to proceed with saved state restoration. The necessary information
is later provided by the \{@link \#\{initiateFederateRestore initiateFederateRestore\}\} callback.

Once in the Prepared To Restore state, the federate is severely limited in which RTIambassador
methods it can invoke. This lasts through the Restoring and Waiting For Federation To Restore states,
concluding with the \{@link \#\{federationRestored federationRestored\}\} or
\{@link \#\{federationNotRestored federationNotRestored\}\} callbacks.

@throws FederateInternalError should be thrown if something goes wrong

@see RTIambassador\#requestFederationRestore requestFederationRestore
@see RTIambassador\#federateRestoreComplete federateRestoreComplete
@see RTIambassador\#federateRestoreNotComplete federateRestoreNotComplete
@see RTIambassador\#queryFederationRestoreStatus queryFederationRestoreStatus
@see \#\{requestFederationRestoreSucceeded requestFederationRestoreSucceeded\}
@see \#\{requestFederationRestoreFailed requestFederationRestoreFailed\}
@see \#\{initiateFederateRestore initiateFederateRestore\}
@see \#\{federationRestored federationRestored\}
@see \#\{federationNotRestored federationNotRestored\}
@see \#\{federationRestoreStatusResponse federationRestoreStatusResponse\}

public void federationRestoreBegun()
throws FederateInternalError;
Instructs the joined federate to return to a previously saved state. The joined federate should use the supplied `<code>label</code>`, the name of the federation execution (see the `<code>federationExecutionName</code>`) of the `{@link RTIambassador#joinFederationExecution joinFederationExecution}` invocation), the supplied federate designator `<code>federateHandle</code>` and its federate type (the aforementioned invocation's `<code>federateType</code>` argument) to retrieve the saved state information.

Note that the supplied `<code>federateHandle</code>` may differ from the federate's current `{@link FederateHandle} handle`. It will assume this new designator if and once it receives the `{@link #federationRestored federationRestored}` callback.

@see RTIambassador#requestFederationRestore
@see RTIambassador#federateRestoreComplete
@see RTIambassador#federateRestoreNotComplete
@see RTIambassador#queryFederationRestoreStatus
@see #requestFederationRestoreSucceeded
@see #requestFederationRestoreFailed
@see #federationRestoreBegun
@see #federationRestored
@see #federationNotRestored
@see #federationRestoreStatusResponse

```java
public void initiateFederateRestore(
    String label,
    FederateHandle federateHandle)
    throws SpecifiedSaveLabelDoesNotExist,
            CouldNotInitiateRestore,
            FederateInternalError;
```
/**
 * Informs the joined federate that the federation restore process is complete and successful.
 * This means that all joined federates which received the {@link RTIambassador#federateRestoreComplete federateRestoreComplete} method.
 * The federate's [{@link FederateHandle} federateHandle] is now the one that was supplied by the [{@link #initiateFederateRestore initiateFederateRestore}] callback.
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore
 * @see RTIambassador#federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus
 * @see #requestFederationRestoreSucceeded
 * @see #requestFederationRestoreFailed
 * @see #federationRestoreBegun
 * @see #initiateFederateRestore
 * @see #federationNotRestored
 * @see #federationRestoreStatusResponse
 */

public void federationRestored() throws FederateInternalError;
/**
 * Informs the joined federate that the federation restore process has completed in failure.
 * <p>
 * The possible save failure reasons are:
 * <ul>
 * <li> RTI_UNABLE_TO_RESTORE: The RTI was unable to restore
 * <li> FEDERATE_REPORTED_FAILURE: One or more federates have invoked the [link RTIambassador#federateRestoreNotComplete](federateRestoreNotComplete) method
 * <li> FEDERATE_RESIGNED: One or more joined federates have resigned from the federation execution
 * <li> RTI_DETECTED_FAILURE: The RTI has detected failure at one or more of the joined federates
 * </ul>
 * @param reason a [link RestoreFailureReason](RestoreFailureReason) specifying the reason for the failure
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore
 * @see RTIambassador#federateRestoreComplete
 * @see RTIambassador#federateRestoreNotComplete
 * @see RTIambassador#queryFederationRestoreStatus
 * @see #requestFederationRestoreSucceeded
 * @see #requestFederationRestoreFailed
 * @see #federationRestoreBegun
 * @see #initiateFederateRestore
 * @see #federationRestored
 * @see #federationRestoreStatusResponse
 */

public void federationNotRestored(RestoreFailureReason reason) throws FederateInternalError;
public void federationRestoreStatusResponse(
    FederateHandleRestoreStatusPair[] response)
    throws FederateInternalError;
The FederateAmbassadorObjectRegistrationAdvisory interface is the FederateAmbassador part devoted to the object class relevance advisories.

// File: FederateAmbassadorObjectRegistrationAdvisory.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the object class relevance advisories (i.e. the Registration Advisory callbacks).
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public interface FederateAmbassadorObjectRegistrationAdvisory
{
    // Declaration Management Services - Registration Advisories

    // 5.10
    /**
     * Notifies the federate that registration of new object instances of the specified object class is advised because at least one of the federate-published class attributes is actively subscribed to by at least one other federate.
     * This occurs only if the federate's Object Class Relevance Advisory Switch is turned on.
     * @param theClass the ObjectClassHandle of the subject object class
     * @throws ObjectClassNotPublished should be thrown if the federate denies publishing <code>theClass</code>
     * @throws FederateInternalError should be thrown if something else goes wrong
     * @see #stopRegistrationForObjectClass
     * @see RTIambassador#enableObjectClassRelevanceAdvisorySwitch
     * @see RTIambassador#disableObjectClassRelevanceAdvisorySwitch
     */
    public void startRegistrationForObjectClass(ObjectClassHandle theClass)
        throws ObjectClassNotPublished, FederateInternalError;

    stopRegistrationForObjectClass;
    enableObjectClassRelevanceAdvisorySwitch;
    disableObjectClassRelevanceAdvisorySwitch;

    */

    public void startRegistrationForObjectClass(ObjectClassHandle theClass)
        throws ObjectClassNotPublished, FederateInternalError;
}
// 5.11
/**
 * Notifies the federate that registration of new object instances 
 * of the specified object class 
 * is not advised because there are no active subscribers to any of 
 * the federate-published class attributes.
 * This occurs only if the federate’s Object Class Relevance 
 * Advisory Switch is turned on.
 * @param theClass the ObjectClassHandle of the subject object class
 * @throws ObjectClassNotPublished should be thrown if the federate 
 * denies publishing theClass
 * @throws FederateInternalError should be thrown if something else 
 * goes wrong
 * @see startRegistrationForObjectClass 
 * @see RTIambassador#enableObjectClassRelevanceAdvisorySwitch 
 * @see RTIambassador#disableObjectClassRelevanceAdvisorySwitch 
 */
public void stopRegistrationForObjectClass(ObjectClassHandle theClass)
    throws ObjectClassNotPublished,
    FederateInternalError;

//end FederateAmbassadorObjectRegistrationAdvisory
The `FederateAmbassadorInteractionAdvisory` interface is the FederateAmbassador part devoted to the server (publisher) interaction scope advisories.

```java
// File: FederateAmbassadorInteractionAdvisory.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Interaction Scope Advisory callbacks.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface FederateAmbassadorInteractionAdvisory
{
    // Declaration Management Services - Interaction Advisory

    // 5.12
    /**
     * Notifies the federate that the specified class of interactions is relevant because there is at least one active subscription by another federate. This occurs only if the federate's Interaction Relevance Advisory Switch is turned on.
     * @param theHandle the {@link InteractionClassHandle} of the subject interaction class
     * @throws InteractionClassNotPublished should be thrown if the federate denies publishing <code>theHandle</code>
     * @throws FederateInternalError should be thrown if something else goes wrong
     * @see #turnInteractionsOff turnInteractionsOff
     * @see RTIambassador#enableInteractionRelevanceAdvisorySwitch enableInteractionRelevanceAdvisorySwitch
     * @see RTIambassador#disableInteractionRelevanceAdvisorySwitch disableInteractionRelevanceAdvisorySwitch
     */
    public void turnInteractionsOn(
        InteractionClassHandle theHandle)
    throws InteractionClassNotPublished,
            FederateInternalError;
```
/ 5.13
/**
 * Notifies the federate that the specified class of interactions is not relevant because
 * there are no active subscriptions by other federates.
 * This occurs only if the federate's Interaction Relevance Advisory Switch is turned on.
 * @param theHandle the InteractionClassHandle of the subject interaction class
 * @throws InteractionClassNotPublished should be thrown if the federate denies publishing <code>theHandle</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see #turnInteractionsOn turnInteractionsOn
 * @see RTIambassador#enableInteractionRelevanceAdvisorySwitch enableInteractionRelevanceAdvisorySwitch
 * @see RTIambassador#disableInteractionRelevanceAdvisorySwitch disableInteractionRelevanceAdvisorySwitch
 */
public void turnInteractionsOff(InteractionClassHandle theHandle)
    throws InteractionClassNotPublished, FederateInternalError;
}
The `FederateAmbassadorNameReservation` interface is the part devoted to the name reservation outcome callbacks.

// File: FederateAmbassadorNameReservation.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the `hla.rti1516.FederateAmbassador` interface that contains only the Name Reservation callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public interface FederateAmbassadorNameReservation {

    // 6.3
    /**
     * Notifies the federate that the <code>objectName</code> provided in a previous invocation of the
     * `<link RTIambassador#reserveObjectInstanceName reserveObjectInstanceName>` method has been reserved.
     * <p>
     * DoD Interpretations of IEEE 1516-2000v2 changes the service name from 
     * `<code>objectInstanceNameReservationSucceeded</code>`
     * to `<code>objectInstanceNameReservationSucceeded</code>`.
     * @param objectName a {link java.lang.String} holding the requested object name
     * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
     * @throws FederateInternalError should be thrown if something else goes wrong
     * @see RTIambassador#reserveObjectInstanceName reserveObjectInstanceName
     * @see #objectInstanceNameReservationFailed
     */

    public void objectInstanceNameReservationSucceeded(String objectName)
        throws UnknownName, FederateInternalError;


/**
 * Notifies the federate that the <code>objectName</code> provided in a previous invocation of the
 * [link RTIambassador#reserveObjectInstanceName reserveObjectInstanceName] method could not be reserved.
 * @param objectName a [link java.lang.String] holding the requested object name
 * @throws UnknownName should be thrown if the federate denies requesting to reserve the <code>objectName</code>
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#reserveObjectInstanceName reserveObjectInstanceName
 * @see #objectInstanceNameReservationSucceeded objectInstanceNameReservationSucceeded
 */

public void objectInstanceNameReservationFailed(String objectName)
    throws UnknownName, FederateInternalError;

//end FederateAmbassadorNameReservation
The *FederateAmbassadorObjectDiscovery* interface is the *FederateAmbassador* part devoted to the object discovery callbacks.

```java
// File: FederateAmbassadorObjectDiscovery.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that
 * contains only the Object Discovery callbacks.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface FederateAmbassadorObjectDiscovery
{

   // Object Management Services - Object Discovery

   // 6.5
   /**
    * Notifies the federate that it has discovered an object instance.
    * @param theObject the [ObjectInstanceHandle] of the newly discovered object instance
    * @param theObjectClass the [ObjectClassHandle] of the class the instance was discovered as
    * @param objectName a [java.lang.String] holding the newly discovered object instanceTs name
    * @throws CouldNotDiscover should be thrown if the object instance could not be discovered for some reason other than an unrecognized object class
    * @throws ObjectClassNotRecognized should be thrown if the federate does not recognize <code>theObjectClass</code>
    * @throws FederateInternalError should be thrown if something else goes wrong
    * @see RTIambassador#registerObjectInstance registerObjectInstance
    */
   public void discoverObjectInstance(
      ObjectInstanceHandle theObject,
      ObjectClassHandle theObjectClass,
      String objectName)
         throws CouldNotDiscover,
            ObjectClassNotRecognized,
            FederateInternalError;
}
//end FederateAmbassadorObjectDiscovery
```
The `FederateAmbassadorAttributeUpdateClient` interface is the part devoted to the client (subscriber) object instance attribute updates.

```java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Instance Attribute Update (Receive) callbacks.
 * @author Daniel Thibault (Daniel.U.Thibault@DRDC-RDDC.gc.ca) (http://www.valcartier.drdc-rrdc.gc.ca RDDC Valcartier)
 * @version 1.1
 */
public interface FederateAmbassadorAttributeUpdateClient {

    // Object Management Services - Instance Attribute Update (Receive)

    // 6.7
    /**
     * Provides the federate with new values for the specified instance attributes.
     * This callback, coupled with the `RTIambassador#updateAttributeValues` method, forms the primary data exchange mechanism supported by the RTI.
     * This form is invoked by the RTI only if the sent order type was RECEIVE.
     * no time-stamp was provided
     * and an update region set was not used by the sender (or is not pertinent or is being filtered out by the receiver).
     * @param sentOrdering
     * @param theTransport
     * @param theTransportType
     * @param theTransportStatus
     * @param theTransportStatusCode
     */
}
```
* @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
* @param theAttributes an {@link AttributeHandleValueMap} specifying the new attribute values
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
* @param sentOrdering the {@link OrderType} the passel was sent as
* @param theTransport the {@link TransportationType} used to send the passel
* @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
* @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
* @throws FederateInternalError should be thrown if something else goes wrong
* @see RTIambassador#updateAttributeValues(ObjectInstanceHandle,AttributeHandleValueMap,byte[])

```java
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
```
/**
 * Provides the federate with new values for the specified instance
 * attributes
 * and specifies the update regions used.
 * This callback, coupled with the [RTIambassador#updateAttributeValues
 * updateAttributeValues] method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * no time-stamp was provided,
 * the instance attributes have available dimensions, the
 * federate's Convey Region
 * Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the
 * absence of a [MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the [InvalidOrderType],
 * [InvalidTransportationType], [InvalidRegion] or
 * [InvalidRegionContext]
 * exceptions (nor a notional new <code>InvalidRegionSet</code>
 * exception); in other words the RTI
 * guarantees the validity and pertinence of the supplied
 * <code>sentOrdering</code>,
 * <code>theTransport</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
/**
 * Provides the federate with new values for the specified instance attributes
 * and specifies the time-stamp at which this comes into effect.
 * This callback, coupled with the {@link RTIambassador#updateAttributeValues updateAttributeValues} method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was RECEIVE,
 * a time-stamp was provided
 * and an update region set was not used by the sender (or is not pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the absence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the <code>receivedOrdering</code> is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the {@link InvalidOrderType},
 * {@link InvalidTransportationType} or {@link InvalidLogicalTime} exceptions; in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>, <code>theTransport</code>, <code>theTime</code>
 * and <code>receivedOrdering</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
 */
ReflectAttributeValues

```java
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
```

Throws:

- `ObjectInstanceNotKnown`
- `AttributeNotRecognized`
- `AttributeNotSubscribed`
- `FederateInternalError`

See:

- `RTIambassador#updateAttributeValues(ObjectInstanceHandle, AttributeHandleValueMap, byte[], LogicalTime)`
/**
 * Provides the federate with new values for the specified instance attributes,
 * specifies the update regions used
 * and specifies the time-stamp at which this comes into effect.
 * This callback, coupled with the {@link RTIambassador#updateAttributeValues updateAttributeValues} method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE,
 * a time-stamp was provided,
 * the instance attributes have available dimensions, the
 * federate's
 * Convey Region Designator Sets Switch is enabled and an update region set
 * was used by the sender.
 * <p>
 * Note that the
 * absence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the <code>receivedOrdering</code> is also
 * RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the {@link InvalidOrderType},
 * {@link InvalidTransportationType}, {@link InvalidLogicalTime},
 * {@link InvalidRegion} or
 * {@link InvalidRegionContext} exceptions (nor a notional new
 * <code>InvalidRegionSet</code> exception); in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>, <code>theTransport</code>,
 * <code>theTime</code>,
 * <code>receivedOrdering</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
/**
 * Provides the federate with new values for the specified instance attributes
 * and specifies the time-stamp at which this comes into effect as well as a retraction handle.
 * This callback, coupled with the @link RTIambassador#updateAttributeValues updateAttributeValues@link method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was TIMESTAMP
 * (thus a time-stamp was provided)
 * and an update region set was not used by the sender (or is not pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the presence of a @link MessageRetractionHandle@link implies the
 * <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the @link InvalidOrderType@link,
 * @link InvalidTransportationType@link or @link InvalidMessageRetractionHandle@link exceptions;
 * in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>, <code>theTransport</code>,
 * <code>receivedOrdering</code> and
 * <code>retractionHandle</code>.
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
    throws ObjectInstanceNotKnown,
               AttributeNotRecognized,
               AttributeNotSubscribed,
               InvalidLogicalTime,
               FederateInternalError;
/**
 * Provides the federate with new values for the specified instance attributes,
 * specifies the update regions used
 * and specifies the time-stamp at which this comes into effect as well as a retraction handle.
 * This callback, coupled with the {@link RTIambassador#updateAttributeValues updateAttributeValues} method,
 * forms the primary data exchange mechanism supported by the RTI.
 * This form is invoked by the RTI only if the sent order type was TIMESTAMP
 * (thus a time-stamp was provided),
 * the instance attributes have available dimensions, the federate's Convey Region Designator Sets Switch
 * is enabled and an update region set was used by the sender.
 * <p>
 * Note that the presence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the {@link InvalidOrderType},
 * {@link InvalidTransportationType}, {@link MessageRetractionHandle},
 * {@link InvalidRegion} or {@link InvalidRegionContext} exceptions
 * (nor a notional new <code>InvalidRegionSet</code> exception); in other words the
 * RTI guarantees the validity of the supplied <code>sentOrdering</code>,
 * <code>theTransport</code>, <code>receivedOrdering</code>,
 * <code>retractionHandle</code> and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void reflectAttributeValues(
  ObjectInstanceHandle theObject,
  AttributeHandleValueMap theAttributes,
  byte[] userSuppliedTag,
  OrderType sentOrdering,
  TransportationType theTransport,
  LogicalTime theTime,
  OrderType receivedOrdering,
  MessageRetractionHandle retractionHandle,
  RegionHandleSet sentRegions)
  throws ObjectInstanceNotKnown,
          AttributeNotRecognized,
          AttributeNotSubscribed,
          InvalidLogicalTime,
          FederateInternalError;
}
The FederateAmbassadorInteractionOccurrence interface is the part devoted to interaction occurrence reception.

// File: FederateAmbassadorInteractionOccurrence.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Interaction Update callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public interface FederateAmbassadorInteractionOccurrence
{

   // /////////////////////////////////////////////
   // Object Management Services - Interaction Update
   // /////////////////////////////////////////////

   // 6.9
   /**
    * Provides the federate with a sent interaction.
    * This form is invoked by the RTI only if the sent order type was RECEIVE,
    * no time-stamp was provided
    * and an update region set was not used by the sender (or is not pertinent or is
    * being filtered out by the receiver).
    * <p>
    * Note that the absence of a {@link MessageRetractionHandle} implies the
    * <code>sentOrdering</code> is RECEIVE
    * (and that therefore the received ordering is also RECEIVE).
    * <p>
    * Note that the federate is not expected to throw the {@link InvalidOrderType} or
    * {@link InvalidTransportationType} exceptions; in other words the RTI
    * guarantees the validity of the supplied
    * <code>sentOrdering</code> and <code>theTransport</code>.
    * <p>
    * The time stamp and receive message order type arguments are supplied together or not at all,
    * which explains the absence of some other possible forms of this callback.
   */
* @param interactionClass the {@link InteractionClassHandle} of the received interaction
* @param theParameters a {@link ParameterHandleValueMap} specifying the interaction parameter values
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
* @param sentOrdering the {@link OrderType} the interaction was sent as
* @param theTransport the {@link TransportationType} used to send the interaction
* @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
* @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
* @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
* @throws FederateInternalError should be thrown if something else goes wrong
* @see RTIambassador#sendInteraction(InteractionClassHandle,ParameterHandleValueMap,byte[]) */

    public void receiveInteraction(
        InteractionClassHandle interactionClass,
        ParameterHandleValueMap theParameters,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport)
    throws InteractionClassNotRecognized,
        InteractionParameterNotRecognized,
        InteractionClassNotSubscribed,
        FederateInternalError;
/**
 * Provides the federate with a sent interaction and
 * specifies the broadcasting regions used.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE, no time-stamp was provided,
 * the parameters have available dimensions, the federate’s Convey
 * Region Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the absence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the {@link
 * InvalidOrderType},
 * {@link InvalidTransportationType}, {@link InvalidRegion} or
 * {@link InvalidRegionContext} exceptions
 * (nor a notional new <code>InvalidRegionSet</code> exception); in
 * other words the RTI guarantees
 * the validity of the supplied <code>sentOrdering</code>,
 * <code>theTransport</code>
 * and <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
* @param interactionClass the {@link InteractionClassHandle} of the received interaction
* @param theParameters a {@link ParameterHandleValueMap} specifying the interaction parameter values
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
* @param sentOrdering the {@link OrderType} the interaction was sent as
* @param theTransport the {@link TransportationType} used to send the interaction
* @param sentRegions the {@link RegionHandleSet} used to send the interaction
* @throws InteractionClassNotRecognized should be thrown if the federate does not recognize the interaction class
* @throws InteractionParameterNotRecognized should be thrown if one of the supplied parameters isn't recognized in the supplied context
* @throws InteractionClassNotSubscribed should be thrown if the federate denies subscribing to the interaction
* @throws FederateInternalError should be thrown if something else goes wrong
* @see
* RTIambassador#sendInteractionWithRegions(InteractionClassHandle,ParameterHandleValueMap,RegionHandleSet,byte[])
*/

```java
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
           InteractionParameterNotRecognized,
           InteractionClassNotSubscribed,
           FederateInternalError;
```
/**
 * Provides the federate with a sent interaction and specifies the time-stamp
 * at which this occurs.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE, a time-stamp was provided, and an update region set was not
 * used by the sender (or is not pertinent or is being filtered out by the receiver).
 * Note that the absence of a `MessageRetractionHandle` implies the
 * `sentOrdering` is RECEIVE (and that therefore the received ordering is also RECEIVE).
 * Note that the federate is not expected to throw the `InvalidOrderType` or
 * `InvalidTransportationType` exceptions; in other words the RTI
 * guarantees the validity of the supplied `sentOrdering` and `theTransport`.
 * Note that the federate is not expected to throw the `InvalidOrderType`,
 * `InvalidTransportationType` or `InvalidLogicalTime` exceptions; in other words the
 * RTI guarantees the validity of the supplied `sentOrdering`, `theTransport`,
 * `receivedOrdering` and `theTime`. The time stamp and receive message order type arguments are
 * supplied together or not at all, which explains the absence of some other possible forms of this callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError;
/**
 * Provides the federate with a sent interaction, specifies the broadcasting regions used
 * and specifies the time-stamp at which this occurs.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE, a time-stamp was provided, the parameters have available dimensions, the federate's Convey Region
 * Designator Sets Switch is enabled and an update region set was used by the sender.
 * <p>
 * Note that the absence of a \{@link MessageRetractionHandle\} implies the
 * \(<\texttt{code}>\text{sentOrdering}</\texttt{code}>\) is RECEIVE
 * (and that therefore the received ordering is also RECEIVE).
 * <p>
 * Note that the federate is not expected to throw the \{@link InvalidOrderType\} or
 * \{@link InvalidTransportationType\} exceptions; in other words the RTI
 * guarantees the validity of the supplied
 * \(<\texttt{code}>\text{sentOrdering}</\texttt{code}>\) and \(<\texttt{code}>\text{theTransport}</\texttt{code}>\).
 * Note that the federate is not expected to throw the \{@link InvalidOrderType\},
 * \{@link InvalidTransportationType\}, \{@link InvalidLogicalTime\}, \{@link InvalidRegion\} or
 * \{@link InvalidRegionContext\} exceptions (nor a notional new
 * \(<\texttt{code}>\text{InvalidRegionSet}</\texttt{code}>\) exception);
 * in other words the RTI guarantees the validity of the supplied
 * \(<\texttt{code}>\text{sentOrdering}</\texttt{code}>\),
 * \(<\texttt{code}>\text{theTransport}</\texttt{code}>\), \(<\texttt{code}>\text{receivedOrdering}</\texttt{code}>\),
 * \(<\texttt{code}>\text{theTime}</\texttt{code}>\)
 * and \(<\texttt{code}>\text{sentRegions}</\texttt{code}>\).
 * <p>
 * The time stamp and receive message order type arguments are supplied together or not at all,
 * which explains the absence of some other possible forms of this callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError;
/**
 * Provides the federate with a sent interaction
 * and specifies the time-stamp at which this occurs as well as a
 * retraction handle.
 * This form is invoked by the RTI only if the sent order type was
 * TIMESTAMP
 * (thus a time-stamp was provided)
 * and an update region set was not used by the sender (or is not
 * pertinent or is
 * being filtered out by the receiver).
 * <p>
 * Note that the
 * presence of a [\link MessageRetractionHandle] implies the
 * <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the [\link
 * InvalidOrderType] or
 * [\link InvalidTransportationType] exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * <code>sentOrdering</code> and <code>theTransport</code>.
 * Note that the federate is not expected to throw the [\link
 * InvalidOrderType],
 * [\link InvalidTransportationType] or [\link
 * InvalidMessageRetractionHandle] exceptions;
 * in other words the RTI guarantees the validity of the supplied
 * <code>sentOrdering</code>,
 * <code>theTransport</code>, <code>receivedOrdering</code> and
 * <code>messageRetractionHandle</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError;
/**
 * Provides the federate with a sent interaction,
 * specifies the broadcasting regions used
 * and specifies the time-stamp at which this occurs as well as a
 * retraction handle.
 * This form is invoked by the RTI only if the sent order type was
 * TIMESTAMP
 * (thus a time-stamp was provided),
 * the parameters have available dimensions, the federate’s Convey
 * Region
 * Designator Sets Switch is enabled and an update region set was
 * used by the sender.
 * <p>
 * Note that the
 * presence of a {@link MessageRetractionHandle} implies the
 * <code>sentOrdering</code> is TIMESTAMP.
 * <p>
 * Note that the federate is not expected to throw the {@link
 * InvalidOrderType} or
 * @link InvalidTransportationType} exceptions; in other words the
 * RTI
 * guarantees the validity of the supplied
 * <code>sentOrdering</code> and <code>theTransport</code>,.
 * Note that the federate is not expected to throw the {@link
 * InvalidOrderType},
 * @link InvalidTransportationType}, {@link
 * InvalidMessageRetractionHandle},
 * @link InvalidRegion} or {@link InvalidRegionContext} exceptions
 * (nor a notional
 * new <code>InvalidRegionSet</code> exception); in other words the
 * RTI guarantees the validity
 * of the supplied <code>sentOrdering</code>,
 * <code>theTransport</code>, <code>receivedOrdering</code>,
 * <code>messageRetractionHandle</code> and
 * <code>sentRegions</code>.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
public void receiveInteraction(
    InteractionClassHandle interactionClass,  
    ParameterHandleValueMap theParameters,  
    byte[] userSuppliedTag,  
    OrderType sentOrdering,  
    TransportationType theTransport,  
    LogicalTime theTime,  
    OrderType receivedOrdering,  
    MessageRetractionHandle messageRetractionHandle,  
    RegionHandleSet sentRegions)  
throws InteractionClassNotRecognized,  
    InteractionParameterNotRecognized,  
    InteractionClassNotSubscribed,  
    InvalidLogicalTime,  
    FederateInternalError;
} //end FederateAmbassadorInteractionOccurrence
The FederateAmbassadorObjectRemoval interface is the FederateAmbassador part devoted to the object instance removal callbacks.

// File: FederateAmbassadorObjectRemoval.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Object Removal callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface FederateAmbassadorObjectRemoval
{

///////////////////////////////////////////////
// Object Management Services - Object Removal
///////////////////////////////////////////////

// 6.11
/**
 * Notifies the federate that an object instance has been deleted from the federation execution.
 * This form is invoked by the RTI only if the sent order type was RECEIVE and no time-stamp was provided.
 * The time stamp and receive message order type arguments are supplied together or not at all, which explains the absence of some other possible forms of this callback.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the message was sent as
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#deleteObjectInstance deleteObjectInstance
 */
public void removeObjectInstance,
(ObjectInstanceHandle theObject,
byte[] userSuppliedTag,
OrderType sentOrdering)
throws ObjectInstanceNotKnown,
FederateInternalError;
/**
 * Notifies the federate that an object instance has been deleted
 * from the federation execution
 * at the specified time stamp.
 * This form is invoked by the RTI only if the sent order type was
 * RECEIVE
 * and a time-stamp was provided.
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the message was sent as
 * @param theTime the LogicalTime at which the deletion occurs
 * @param receivedOrdering the OrderType the message was received as
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#deleteObjectInstance deleteObjectInstance
 */
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering)
    throws ObjectInstanceNotKnown,
            FederateInternalError;
/**
 * Notifies the federate that an object instance has been deleted from
 * the federation execution
 * at the specified time stamp and specifies a retraction handle.
 * This form is invoked by the RTI only if the sent order type was
 * TIMESTAMP
 * (and thus a time-stamp was provided).
 * <p>
 * The time stamp and receive message order type arguments are
 * supplied together or not at all,
 * which explains the absence of some other possible forms of this
 * callback.
 * @param theObject the {@link ObjectInstanceHandle} of the
 * concerned object instance
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @param sentOrdering the {@link OrderType} the message was sent
 * as
 * @param theTime the {@link LogicalTime} at which the deletion
 * occurs
 * @param receivedOrdering the {@link OrderType} the message was
 * received as
 * @param retractionHandle the {@link MessageRetractionHandle} of
 * the message
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws InvalidLogicalTime should be thrown if the specified
 * <code>LogicalTime</code> is invalid (only if the
 * <code>receivedOrdering</code> is TIMESTAMP)
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see RTIambassador#deleteObjectInstance deleteObjectInstance
 */

public void
removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
    InvalidLogicalTime,
    FederateInternalError;

//end FederateAmbassadorObjectRemoval
The `FederateAmbassadorAttributeScopeAdvisoryClient` interface is the FederateAmbassador part devoted to the client (subscriber) object instance attribute scope advisories.

```java
// File: FederateAmbassadorAttributeScopeAdvisoryClient.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Instance Attribute Scope Advisory callbacks.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} {{@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier}}
 * @version 1.1
 */

public interface FederateAmbassadorAttributeScopeAdvisoryClient {

```
```
// Object Management Services - Instance Attribute Scope Advisory
(Receive)

/**
 * Notifies the federate that the specified attributes for the
 * object instance are in its scope.
 * This occurs only if the Attribute Scope Advisory Switch is on
 * for the federate.
 * @param theObject the ObjectInstanceHandle of the
 * concerned object instance
 * @param theAttributes an AttributeHandleSet specifying
 * the pertinent attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate
 * denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see #attributesOutOfScope
 * @see RTIambassador#enableAttributeScopeAdvisorySwitch
 * @see RTIambassador#disableAttributeScopeAdvisorySwitch
 */
public void attributesInScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError;
/**
 * Notifies the federate that the specified attributes for the object instance are out of its scope.
 * This occurs only if the Attribute Scope Advisory Switch is on for the federate.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleSet specifying the pertinent attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered
 * the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in
 * the supplied context
 * @throws AttributeNotSubscribed should be thrown if the federate denies subscribing to one of the attributes
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see attributesInScope attributesInScope
 * @see enableAttributeScopeAdvisorySwitch
 * @see disableAttributeScopeAdvisorySwitch
 */

public void attributesOutOfScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
             AttributeNotRecognized,
             AttributeNotSubscribed,
             FederateInternalError;

// end FederateAmbassadorAttributeScopeAdvisoryClient
The **FederateAmbassadorAttributeUpdateServer** interface is the part devoted to the server (publisher) object instance attribute update request callback.

```java
// File: FederateAmbassadorAttributeUpdateServer.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Instance Attribute Update (Send) callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface FederateAmbassadorAttributeUpdateServer
{
```
Object Management Services - Instance Attribute Update (Send)

6.18

* Requests of the federate the current values of the specified
  instance attributes, which it owns.
  * The federate should respond through the [@link
    RTIambassador#updateAttributeValues updateAttributeValues] method.
  * @param theObject the [@link ObjectInstanceHandle] of the subject
    object instance
  * @param theAttributes an [@link AttributeHandleSet] specifying
    the requested attributes
  * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
    may be <code>null</code>)
  * @throws ObjectInstanceNotKnown should be thrown if the federate
    denies having previously discovered the object instance
  * @throws AttributeNotRecognized should be thrown if one of the
    supplied attributes isn't recognized in the supplied context
  * @throws AttributeNotOwned should be thrown if the federate
    denies owning an attribute
  * @throws FederateInternalError should be thrown if something else
    goes wrong
  * @see RTIambassador#requestAttributeValueUpdate
    requestAttributeValueUpdate
*

```java
public void provideAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
    throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError;
}
```

//end FederateAmbassadorAttributeUpdateServer
The `FederateAmbassadorAttributeScopeAdvisoryServer` interface is the FederateAmbassador part devoted to the server (publisher) object instance attribute scope advisories.

```java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Instance Attribute Scope Advisory (Send) callbacks.
 * <p>
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */
public interface FederateAmbassadorAttributeScopeAdvisoryServer {
```
/**
   * Indicates to the federate that the values of the specified instance attributes are required somewhere in the federation execution. The federate should therefore update them as needed.
   * This occurs only if the Attribute Relevance Advisory Switch is on for the federate.
   * @param theObject the ObjectInstanceHandle of the subject object instance
   * @param theAttributes an AttributeHandleSet specifying the subject attributes
   * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
   * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
   * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
   * @throws FederateInternalError should be thrown if something else goes wrong
   * @see #turnUpdatesOffForObjectInstance
   * @see RTIambassador#enableAttributeRelevanceAdvisorySwitch
   * @see RTIambassador#disableAttributeRelevanceAdvisorySwitch
   * @see RTIambassador#updateAttributeValues
   */

public void turnUpdatesOnForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError;

// 6.20
/**
 * Indicates to the federate that the values of the specified
 * instance attributes are no
 * longer required anywhere in the federation execution.
 * This occurs only if the Attribute Relevance Advisory Switch is
 * on for the federate.
 * @param theObject the {@link ObjectInstanceHandle} of the subject
 * object instance
 * @param theAttributes an {@link AttributeHandleSet} specifying
 * the subject attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate
 * denies owning an attribute
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see #turnUpdatesOnForObjectInstance
 * turnUpdatesOnForObjectInstance
 * @see RTIambassador#enableAttributeRelevanceAdvisorySwitch
 * enableAttributeRelevanceAdvisorySwitch
 * @see RTIambassador#disableAttributeRelevanceAdvisorySwitch
 * disableAttributeRelevanceAdvisorySwitch
 * @see RTIambassador#updateAttributeValues updateAttributeValues
 */
public void turnUpdatesOffForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
        throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotOwned,
            FederateInternalError;
}  //end FederateAmbassadorAttributeScopeAdvisoryServer
The FederateAmbassadorAttributeOwnership interface is the part devoted to object ownership management.

// File: FederateAmbassadorAttributeOwnership.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Instance Attribute Ownership callbacks.
 */
public interface FederateAmbassadorAttributeOwnership
{

Ownership Management Services

7.4

**Requests that the federate acquire ownership of the specified instance attributes.**
* This can occur because the original owner invoked [RTIambassador#unconditionalAttributeOwnershipDivestiture](RTIambassador#unconditionalAttributeOwnershipDivestiture), [RTIambassador#unpublishObjectClassAttributes](RTIambassador#unpublishObjectClassAttributes) or [RTIambassador#resignFederationExecution](RTIambassador#resignFederationExecution) (with the UNCONDITIONALLY_DIVEST_ATTRIBUTES, DELETE_OBJECTS_THEN_DIVEST or CANCEL_THEN_DELETE_THEN_DIVEST policy).

* The federate may return a subset of the offeredAttributes for which it is willing to assume ownership through the [RTIambassador#attributeOwnershipAcquisition](RTIambassador#attributeOwnershipAcquisition) or [RTIambassador#attributeOwnershipAcquisitionIfAvailable](RTIambassador#attributeOwnershipAcquisitionIfAvailable) methods.

* @param theObject the [ObjectInstanceHandle](ObjectInstanceHandle) of the concerned object instance
* @param offeredAttributes an [AttributeHandleSet](AttributeHandleSet) specifying the offered attributes
* @param userSuppliedTag a byte[] tag (this parameter may be null)

* @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
* @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
* @throws AttributeNotPublished should be thrown if the federate denies publishing an attribute
* @throws FederateInternalError should be thrown if something else goes wrong

```java
public void requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeNotPublished,
        FederateInternalError;
```
// 7.5
/**
 * Notifies the federate that potential new owners have been found for the
 * specified instance attributes and that the negotiated divestiture of these
 * can now be completed. The federate can either complete the negotiated
 * divestiture using [link RTIambassador#confirmDivestiture confirmDivestiture],
 * divest ownership of the instance attributes by some other means (e.g.,
 * using [link RTIambassador#unconditionalAttributeOwnershipDivestiture
 * unconditionalAttributeOwnershipDivestiture]) or it can
 * [link RTIambassador#cancelNegotiatedAttributeOwnershipDivestiture
 * cancelNegotiatedAttributeOwnershipDivestiture].
 * @param theObject the [link ObjectInstanceHandle] of the concerned object instance
 * @param offeredAttributes an [link AttributeHandleSet] specifying the offered attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
 * @throws AttributeDivestitureWasNotRequested should be thrown if the federate repudiates the divestiture
 * @throws FederateInternalError should be thrown if something else goes wrong
 */
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotOwned,
            AttributeDivestitureWasNotRequested,
            FederateInternalError;
/**
 * Notifies the federate that it now owns the specified set of
 * instance attributes.
 * The federate may receive multiple notifications for a single
 * invocation of
 * @link RTIambassador#attributeOwnershipAcquisition
 * attributeOwnershipAcquisition} or
 * @link RTIambassador#attributeOwnershipAcquisitionIfAvailable
 * attributeOwnershipAcquisitionIfAvailable} if the requested
 * instance attributes are owned by different federates.
 * @param theObject the {link ObjectInstanceHandle} of the
 * concerned object instance
 * @param securedAttributes an {link AttributeHandleSet}
 * specifying the secured attributes
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @throws ObjectInstanceNotKnown should be thrown if the federate
 * denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the
 * supplied attributes isn't recognized in the supplied context
 * @throws AttributeAcquisitionWasNotRequested should be thrown if
 * the federate repudiates its attribute ownership acquisition request
 * @throws AttributeAlreadyOwned should be thrown if the federate
 * thinks it already owns an attribute
 * @throws AttributeNotPublished should be thrown if the federate
 * denies publishing an attribute
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 */
public void
attributeOwnershipAcquisitionNotification(
   ObjectInstanceHandle theObject,
   AttributeHandleSet   securedAttributes,
   byte[]               userSuppliedTag)
throws ObjectInstanceNotKnown,
   AttributeNotRecognized,
   AttributeAcquisitionWasNotRequested,
   AttributeAlreadyOwned,
   AttributeNotPublished,
   FederateInternalError;
// 7.10
/**
 * Notifies the federate that the specified instance attributes
 * were not available for ownership acquisition.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleSet specifying the declined attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
 * @throws AttributeAcquisitionWasNotRequested should be thrown if the federate repudiates its attribute ownership acquisition request
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#attributeOwnershipAcquisitionIfAvailable
 */
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
  throws ObjectInstanceNotKnown,
         AttributeNotRecognized,
         AttributeAlreadyOwned,
         AttributeAcquisitionWasNotRequested,
         FederateInternalError;
---

```java
// 7.11
/**
 * Requests that the federate release ownership of the specified instance attributes.
 * The federate may return the subset of instance attributes for which it is willing
to release ownership through {@link RTIambassador#attributeOwnershipDivestitureIfWanted attributeOwnershipDivestitureIfWanted},
* {@link RTIambassador#unconditionalAttributeOwnershipDivestiture unconditionalAttributeOwnershipDivestiture} or
* {@link RTIambassador#negotiatedAttributeOwnershipDivestiture negotiatedAttributeOwnershipDivestiture}.
* @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
* @param candidateAttributes an {@link AttributeHandleSet} specifying the candidate attributes
* @param userSuppliedTag a <code>byte[]</code> tag (this parameter may be <code>null</code>)
* @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
* @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
* @throws AttributeNotOwned should be thrown if the federate denies owning an attribute
* @throws FederateInternalError should be thrown if something else goes wrong
* @see RTIambassador#attributeOwnershipAcquisition attributeOwnershipAcquisition
*/
public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError;
```
/**
 * Notifies the federate that the pending attribute ownership acquisition requests for the specified instance attributes have been canceled as requested.
 * @param theObject the ObjectInstanceHandle of the concerned object instance
 * @param theAttributes an AttributeHandleSet specifying the subject attributes
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws AttributeAlreadyOwned should be thrown if the federate thinks it already owns an attribute
 * @throws AttributeAcquisitionWasNotCanceled should be thrown if the federate repudiates the attribute ownership acquisition cancellation
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#cancelAttributeOwnershipAcquisition
 */
public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError;

// 7.17
/**
 * In response to an attribute ownership query by the federate,
 * specifies that the instance attribute is unowned.
 * @param theObject the {ObjectInstanceHandle} of the concerned object instance
 * @param theAttribute an {AttributeHandle} specifying the attribute
 * @param theOwner the {FederateHandle} of the federate owning the attribute
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#queryAttributeOwnership
 */
public void attributeIsNotOwned(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
    throws ObjectInstanceNotKnown,
          AttributeNotRecognized,
          FederateInternalError;

/**
 * In response to an attribute ownership query by the federate,
 * specifies that the instance attribute is unowned.
 * @param theObject the {ObjectInstanceHandle} of the concerned object instance
 * @param theAttribute an {AttributeHandle} specifying the attribute
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#queryAttributeOwnership
 */
public void informAttributeOwnership(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute,
    FederateHandle theOwner)
    throws ObjectInstanceNotKnown,
          AttributeNotRecognized,
          FederateInternalError;
/**
 * In response to an attribute ownership query by the federate, specifies the instance attribute's owner: the RTI.
 * @param theObject the {@link ObjectInstanceHandle} of the concerned object instance
 * @param theAttribute an {@link AttributeHandle} specifying the attribute
 * @throws ObjectInstanceNotKnown should be thrown if the federate denies having previously discovered the object instance
 * @throws AttributeNotRecognized should be thrown if one of the supplied attributes isn't recognized in the supplied context
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#queryAttributeOwnership
 */
public void attributeIsOwnedByRTI(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError;
}
//end FederateAmbassadorAttributeOwnership
The FederateAmbassadorTime interface is the FederateAmbassador part devoted to the time management callbacks.

```java
// File: FederateAmbassadorTime.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A subset of the hla.rti1516.FederateAmbassador interface that contains only the Time callbacks.
 */
public interface FederateAmbassadorTime {

    // Time Management Services

    // 8.3
    /**
     * Notifies the federate that its request to enable time-regulation has been honored.
     * @param time the LogicalTime to which the federate's clock has been set
     * @throws InvalidLogicalTime should be thrown if the specified LogicalTime is invalid
     * @throws NoRequestToEnableTimeRegulationWasPending should be thrown if the federate repudiates the time regulation request
     * @throws FederateInternalError should be thrown if something else goes wrong
     * @see RTIambassador#enableTimeRegulation enableTimeRegulation
     */
    public void timeRegulationEnabled(
            LogicalTime time)
            throws InvalidLogicalTime,
            NoRequestToEnableTimeRegulationWasPending,
            FederateInternalError;
```
/**
 * Notifies the federate that its request to enable time-constraint has been honored.
 * @param time the {@link LogicalTime} to which the federate's clock has been set
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
 * @throws NoRequestToEnableTimeConstrainedWasPending should be thrown if the federate repudiates the time constraint request
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#enableTimeConstrained enableTimeConstrained
 */
public void
timeConstrainedEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
    NoRequestToEnableTimeConstrainedWasPending,
    FederateInternalError;

/**
 * Notifies the federate that its request to advance its logical time has been honored.
 * @param theTime the {@link LogicalTime} to which the federate's clock has been set
 * @throws InvalidLogicalTime should be thrown if the specified <code>LogicalTime</code> is invalid
 * @throws JoinedFederateIsNotInTimeAdvancingState should be thrown if the federate does not consider itself in the time-advancing state
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see RTIambassador#timeAdvanceRequest timeAdvanceRequest
 * @see RTIambassador#timeAdvanceRequestAvailable
 * @see RTIambassador#nextMessageRequest nextMessageRequest
 * @see RTIambassador#nextMessageRequestAvailable
 * @see RTIambassador#flushQueueRequest flushQueueRequest
 */
public void
timeAdvanceGrant(
    LogicalTime theTime)
throws InvalidLogicalTime,
    JoinedFederateIsNotInTimeAdvancingState,
    FederateInternalError;
// 8.22
/**
 * Notifies the federate that the previously delivered message
 * specified by the supplied
 * {@link MessageRetractionHandle} has been retracted.
 * <p>
 * Time-constrained federates that do not use the {@link
 * RTIambassador#flushQueueRequest flushQueueRequest} method
 * are not subject to invocation of this service because they will
 * never receive a
 * {@link OrderType TIMESTAMP} message eligible for retraction.
 * Non-constrained federates, however, must be prepared to deal
 * with invocations of this
 * service because any received message that was sent {@link
 * OrderType TIMESTAMP} may be eligible for retraction.
 * @param theHandle the {@link MessageRetractionHandle} specifying
 * the retracted message
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 * @see RTIambassador#retract retract
 */
public void
requestRetraction(
   MessageRetractionHandle theHandle)
throws FederateInternalError;
//end FederateAmbassadorTime
The **FedAmbAttributeScopeAdvisoryListener** class implements the **FederateAmbassadorAttributeScopeAdvisoryClient** interface. It funnels both callbacks to a single method, `doScope` (by appending a Boolean argument), allowing the programmer to either override the latter or the individual `FederateAmbassador` attributes `attributesInScope` or `attributesOutOfScope` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doScope`.

```java
import hla.rti1516.*;
/**
 * Null implementation of
 * FederateAmbassadorAttributeScopeAdvisoryClient interface.
 * This implementation funnels both callbacks to the `doScope` method,
 * so you can either override the `attributesInScope` or `attributesOutOfScope` methods or
 * `doScope`.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} [http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier]
 * @version 1.1
 */
public class FedAmbAttributeScopeAdvisoryListener
  implements FederateAmbassadorAttributeScopeAdvisoryClient
{
  /**
   * The `ValidateAttributesInScope` validation interface.
   */
  public ValidateAttributesInScope attributesInScopeValidator;

  /**
   * The `ValidateAttributesOutOfScope` validation interface.
   */
  public ValidateAttributesOutOfScope attributesOutOfScopeValidator;

  /**
   * Default constructor. Creates an instance with no validators in place.
   */
  public FedAmbAttributeScopeAdvisoryListener()
  {
  }
```
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param attributesInScope_Validator A ValidateAttributesInScope interface
 * @param attributesOutOfScope_Validator A ValidateAttributesOutOfScope interface
 */
public FedAmbAttributeScopeAdvisoryListener(
    ValidateAttributesInScope attributesInScope_Validator,
    ValidateAttributesOutOfScope attributesOutOfScope_Validator)
{
    this();
    attributesInScopeValidator = attributesInScope_Validator;
    attributesOutOfScopeValidator = attributesOutOfScope_Validator;
}

//FederateAmbassadorAttributeScopeAdvisoryClient implementation

public void attributesInScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    if (null != attributesInScopeValidator)
    attributesInScopeValidator.validate(theObject, theAttributes);
    doScope(theObject, theAttributes, true);
}

public void attributesOutOfScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    if (null != attributesOutOfScopeValidator)
    attributesOutOfScopeValidator.validate(theObject, theAttributes);
    doScope(theObject, theAttributes, false);
}
/**
 * This implementation simply funnels both callbacks to this
 * method, so you can either override the attributesIn/OutOfScope methods
 * or this one.
 * Validation is handled by the callbacks.
 * @param theObject the {@link ObjectInstanceHandle} of the
 * concerned object instance
 * @param theAttributes an {@link AttributeHandleSet} specifying
 * the pertinent attributes
 * @param inScope a boolean indicating whether the specified
 * attributes are in or out of scope
 * @throws FederateInternalError should be thrown if something goes
 * wrong
 */

protected void
doScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    boolean inScope)
throws FederateInternalError
{
}

//end FedAmbAttributeScopeAdvisoryListener
The **FedAmbAttributeScopeAdvisoryResponder** class implements the **FederateAmbassadorAttributeScopeAdvisoryServer** interface. It funnels both callbacks to a single method, `doScope` (by appending a Boolean argument), allowing the programmer to either override the latter or the individual `FederateAmbassador turnUpdatesOn/OffForObjectInstance` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doScope`.

```java
// File: FedAmbAttributeScopeAdvisoryResponder.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of the FederateAmbassadorAttributeScopeAdvisoryServer interface.
 * This implementation funnels both callbacks to the `doScope` method, so you can either override
 * the `turnUpdatesOn/OffForObjectInstance` methods or `doScope`.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier]
 * @version 1.1
 */

public class FedAmbAttributeScopeAdvisoryResponder
    implements FederateAmbassadorAttributeScopeAdvisoryServer
{
    /**
     * The ValidateTurnUpdatesOnForObjectInstance validation interface.
     */
    public ValidateTurnUpdatesOnForObjectInstance
        turnUpdatesOnForObjectInstanceValidator;

    /**
     * The ValidateTurnUpdatesOffForObjectInstance validation interface.
     */
    public ValidateTurnUpdatesOffForObjectInstance
        turnUpdatesOffForObjectInstanceValidator;

    /**
     * Default constructor. Creates an instance with no validators in place.
     */
    public FedAmbAttributeScopeAdvisoryResponder()
    { }
}
```
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param turnUpdatesOnForObjectInstance_Validator A ValidateTurnUpdatesOnForObjectInstance interface
 * @param turnUpdatesOffForObjectInstance_Validator A ValidateTurnUpdatesOffForObjectInstance interface
 */

public FedAmbAttributeScopeAdvisoryResponder(
    ValidateTurnUpdatesOnForObjectInstance
    turnUpdatesOnForObjectInstance_Validator,
    ValidateTurnUpdatesOffForObjectInstance
    turnUpdatesOffForObjectInstance_Validator)
{
    this();
    turnUpdatesOnForObjectInstanceValidator =
    turnUpdatesOnForObjectInstance_Validator;
    turnUpdatesOffForObjectInstanceValidator =
    turnUpdatesOffForObjectInstance_Validator;
}

//FederateAmbassadorAttributeScopeAdvisoryServer implementation

public void
turnUpdatesOnForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    if (null != turnUpdatesOnForObjectInstanceValidator)
        turnUpdatesOnForObjectInstanceValidator.validate(theObject,
            theAttributes);
        doScope(theObject, theAttributes, true);
}

public void
turnUpdatesOffForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    if (null != turnUpdatesOffForObjectInstanceValidator)
        turnUpdatesOffForObjectInstanceValidator.validate(theObject,
            theAttributes);
        doScope(theObject, theAttributes, false);
}
/**
 * This implementation simply funnels both callbacks to this method,
 * so you can either override the turnUpdatesOn/OffForObjectInstance methods or this one.
 * Validation is handled by the callbacks.
 * @param theObject the {@link ObjectInstanceHandle} of the subject object instance
 * @param theAttributes an {@link AttributeHandleSet} specifying the subject attributes
 * @param inScope a boolean indicating whether the updates should be turned on or not
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see FederateAmbassador#turnUpdatesOnForObjectInstance
 * turnUpdatesOnForObjectInstance
 * @see FederateAmbassador#turnUpdatesOffForObjectInstance
 * turnUpdatesOffForObjectInstance
 * @see RTIambassador#enableAttributeRelevanceAdvisorySwitch
 * enableAttributeRelevanceAdvisorySwitch
 * @see RTIambassador#disableAttributeRelevanceAdvisorySwitch
 * disableAttributeRelevanceAdvisorySwitch
 */
protected void
doScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    boolean inScope)
    throws FederateInternalError
{
}
//end FedAmbAttributeScopeAdvisoryResponder
The **FedAmbDiscoveryListener** class implements the **FederateAmbassadorObjectDiscovery** interface. Being a null implementation, it does nothing with the callback.

The class has a public member holding the validation interface reference. The callback method invokes this if specified.

```java
package ca.gc.drdr_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
   * Null implementation of FederateAmbassadorObjectDiscovery interface.
   * The user should extend this class and override the discoverObjectInstance method.
   * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} [http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier]}
   * @version 1.1
   */
public class FedAmbDiscoveryListener
   implements FederateAmbassadorObjectDiscovery
{
   /**
      * The ValidateDiscoverObjectInstance validation interface.
      */
   public ValidateDiscoverObjectInstance discoverObjectInstanceValidator;

   /**
      * Default constructor. Creates an instance with no validator in place.
      */
   public FedAmbDiscoveryListener()
   {
   }

   /**
      * Alternate constructor. Creates an instance with the specified validator in place.
      * @param discoverObjectInstance_Validator A ValidateDiscoverObjectInstance interface
      */
   public FedAmbDiscoveryListener(ValidateDiscoverObjectInstance discoverObjectInstance_Validator)
   {
      this();
      discoverObjectInstanceValidator = discoverObjectInstance_Validator;
   }
```
public void discoverObjectInstance(
   ObjectInstanceHandle theObject,
   ObjectClassHandle theObjectClass,
   String objectName)
throws CouldNotDiscover,
   ObjectClassNotRecognized,
   FederateInternalError
{
   if (null != discoverObjectInstanceValidator)
      discoverObjectInstanceValidator.validate(theObject, theObjectClass, objectName);
}  // end FedAmbDiscoveryListener
The FedAmbInstanceAttributeListener class implements the FederateAmbassadorAttributeUpdateClient interface. It funnels all six forms of the callback to a single method, doUpdate (by supplying nulls for any missing arguments), allowing the programmer to either override the latter or the individual FederateAmbassador reflectAttributeValues methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has a public member holding the validation interface reference. The callback methods invoke this if specified, before handing off to doUpdate.

// File: FedAmbInstanceAttributeListener.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;
import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorAttributeUpdateClient interface.
 * All of the reflectAttributeValues callbacks are funneled to the doUpdate method,
 * so you can either override the relevant callback(s) or doUpdate.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} [{http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier}]
 * @version 1.1
 */
public class FedAmbInstanceAttributeListener
    implements FederateAmbassadorAttributeUpdateClient
{
    /**
     * The ValidateReflectAttributeValues validation interface.
     */
    public ValidateReflectAttributeValues reflectAttributeValuesValidator;

    /**
     * Default constructor. Creates an instance with no validator in place.
     */
    public FedAmbInstanceAttributeListener(){}
/**
 * Alternate constructor. Creates an instance with the specified
 * validator in place.
 * @param reflectAttributeValues_Validator A
 * ValidateReflectAttributeValues interface
 */

publicFedAmbInstanceAttributeListener(ValidateReflectAttributeValues reflectAttributeValues_Validator)
{
    // this();
    reflectAttributeValuesValidator = reflectAttributeValues_Validator;
}

//FederateAmbassadorAttributeUpdateClient implementation

public voidreflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport);

    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, null, null, null, null);
}

public voidreflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport, sentRegions);

    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, null, null, null, sentRegions);
}
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport, theTime, receivedOrdering);
    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering, null, null);
}

public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport, theTime, receivedOrdering, sentRegions);
    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering, null, sentRegions);
}
public void reflectAttributeValueValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            InvalidLogicalTime,
            FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport, theTime, receivedOrdering, retractionHandle);
    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        retractionHandle, null);
}

public void reflectAttributeValueValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            InvalidLogicalTime,
            FederateInternalError
{
    if (null != reflectAttributeValuesValidator)
        reflectAttributeValuesValidator.validate(
            theObject, theAttributes, userSuppliedTag, sentOrdering,
            theTransport, theTime, receivedOrdering, retractionHandle, sentRegions);
    doUpdate(theObject, theAttributes, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        retractionHandle, sentRegions);
}

DRDC Valcartier TR 2007-412 901
/**
 * This implementation simply funnels all reflectAttributeValues
 * callbacks to this method,
 * so you can either override the relevant method(s) or this one.
 * Missing arguments will be null.
 * Validation is handled by the callbacks.
 * @param theObject the ObjectInstanceHandle of the
 * concerned object instance
 * @param theAttributes an AttributeHandleValueMap
 * specifying the new attribute values
 * @param userSuppliedTag a byte[] tag (this parameter
 * may be null)
 * @param sentOrdering the OrderType the passel was sent as
 * @param theTransport the TransportationType used to send
 * the passel
 * @param theTime the LogicalTime at which the update comes
 * into effect
 * @param receivedOrdering the OrderType the passel was
 * received as
 * @param retractionHandle the MessageRetractionHandle of
 * the message
 * @param sentRegions the RegionHandleSet used to send the
 * update
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see FederateAmbassador#reflectAttributeValues
 * reflectAttributeValues
 */
 protected void doUpdate(
   ObjectInstanceHandle theObject,
   AttributeHandleValueMap theAttributes,
   byte[] userSuppliedTag,
   OrderType sentOrdering,
   TransportationType theTransport,
   LogicalTime theTime,
   OrderType receivedOrdering,
   MessageRetractionHandle retractionHandle,
   RegionHandleSet sentRegions)
   throws FederateInternalError
   {
   }
   //end FedAmbInstanceAttributeListener
The **FedAmbInstanceAttributeResponder** class implements the **FederateAmbassadorAttributeUpdateServer** interface. Being a null implementation, it does nothing with the callback.

The class has a public member holding the validation interface reference. The callback method invokes this if specified.

```java
// File: FedAmbInstanceAttributeResponder.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorAttributeUpdateServer interface.
 * The user should extend this class and override the provideValueUpdate method.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public class FedAmbInstanceAttributeResponder
    implements FederateAmbassadorAttributeUpdateServer
{
    /**
     * The ValidateProvideAttributeValueUpdate validation interface.
     */
    public ValidateProvideAttributeValueUpdate provideAttributeValueUpdateValidator;

    /**
     * Default constructor. Creates an instance with no validator in place.
     */
    public FedAmbInstanceAttributeResponder()
    {
    }
}
```
/**
 * Alternate constructor. Creates an instance with the specified validator in place.
 * @param provideAttributeValueUpdate_Validator A ValidateProvideAttributeValueUpdate interface
 */

public FedAmbInstanceAttributeResponder(
    ValidateProvideAttributeValueUpdate
    provideAttributeValueUpdate_Validator
)
{
    //
    provideAttributeValueUpdateValidator = provideAttributeValueUpdate_Validator;
}

//FederateAmbassadorAttributeUpdateServer implementation

public void provideAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
    throws ObjectInstanceNotKnown,
           AttributeNotRecognized,
           AttributeNotOwned,
           FederateInternalError
{
    if (null != provideAttributeValueUpdateValidator)
    provideAttributeValueUpdateValidator.validate(theObject,
        theAttributes, userSuppliedTag);
}

//end FedAmbInstanceAttributeResponder
The `FedAmbInteractionAdvisoryResponder` class implements the `FederateAmbassadorInteractionAdvisory` interface. It funnels both callbacks to a single method, `doRelevance` (by appending a Boolean argument), allowing the programmer to either override the latter or the individual `FederateAmbassador turnInteractionOn/off` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doRelevance`.

```java
// File: FedAmbInteractionAdvisoryResponder.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorInteractionAdvisory interface.
 * This implementation funnels both callbacks to the doRelevance method,
 * so you can either override the turnInteractionsOn/Off methods or
doRelevance.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault}
 * @version 1.1
 */
public class FedAmbInteractionAdvisoryResponder
    implements FederateAmbassadorInteractionAdvisory {
    /**
     * The ValidateTurnInteractionsOn validation interface.
     */
    public ValidateTurnInteractionsOn
        turnInteractionsOnValidator;

    /**
     * The ValidateTurnInteractionsOff validation interface.
     */
    public ValidateTurnInteractionsOff
        turnInteractionsOffValidator;

    /**
     * Default constructor. Creates an instance with no validators in place.
     */
    public FedAmbInteractionAdvisoryResponder() {
    }
```
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param turnInteractionsOn_Validator A ValidateTurnInteractionsOn interface
 * @param turnInteractionsOff_Validator A ValidateTurnInteractionsOff interface
 */

public FedAmbInteractionAdvisoryResponder(
    ValidateTurnInteractionsOn turnInteractionsOn_Validator,
    ValidateTurnInteractionsOff turnInteractionsOff_Validator)
{
    this();
    turnInteractionsOnValidator = turnInteractionsOn_Validator;
    turnInteractionsOffValidator = turnInteractionsOff_Validator;
}

//FederateAmbassadorInteractionAdvisory implementation

public void turnInteractionsOn(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError
{
    if (null != turnInteractionsOnValidator)
        turnInteractionsOnValidator.validate(theHandle);
        doRelevance(theHandle, true);
}

public void turnInteractionsOff(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError
{
    if (null != turnInteractionsOffValidator)
        turnInteractionsOffValidator.validate(theHandle);
        doRelevance(theHandle, false);
}
/**
 * This implementation simply funnels both callbacks to this method,
 * so you can either override the turnInteractionsOn/Off methods or
 * this one.
 * Validation is handled by the callbacks.
 * @param theHandle the {@link InteractionClassHandle} of the
 * subject interaction class
 * @param relevant a boolean indicating whether the specified
 * interaction class is relevant or not
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see FederateAmbassador#turnInteractionsOn turnInteractionsOn
 * @see FederateAmbassador#turnInteractionsOff turnInteractionsOff
 * @see RTIambassador#enableInteractionRelevanceAdvisorySwitch
 * enableInteractionRelevanceAdvisorySwitch
 * @see RTIambassador#disableInteractionRelevanceAdvisorySwitch
 * disableInteractionRelevanceAdvisorySwitch
 */
 protected void doRelevance(InteractionClassHandle theHandle, boolean relevant)
 throws FederateInternalError
 { }
}
//end FedAmbInteractionAdvisoryResponder
The `FedAmbInteractionListener` class implements the `FederateAmbassadorInteractionOccurrence` interface. It funnels all six forms of the callback to a single method, `doInteraction` (by supplying nulls for any missing arguments), allowing the programmer to either override the latter or the individual `FederateAmbassador receiveInteraction` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has a public member holding the validation interface reference. The callback methods invoke this if specified, before handing off to `doInteraction`.

```java
// File: FedAmbInteractionListener.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorInteractionOccurrence interface.
 * This implementation funnels all callbacks to the doInteraction method,
 * so you can either override the relevant callback(s) or doInteraction.
 * @author [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */
public class FedAmbInteractionListener
    implements FederateAmbassadorInteractionOccurrence
{
    /**
     * The ValidateReceiveInteraction validation interface.
     */
    public ValidateReceiveInteraction receiveInteractionValidator;

    /**
     * Default constructor. Creates an instance with no validator in place.
     */
    public FedAmbInteractionListener()
    {
    }
```
/**
 * Alternate constructor. Creates an instance with the specified
 * validator in place.
 * @param receiveInteraction_Validator A ValidateReceiveInteraction
 * interface
 */
 public
 FedAmbInteractionListener(
   ValidateReceiveInteraction receiveInteraction_Validator)
 { // this();
   receiveInteractionValidator = receiveInteraction_Validator;
 }

//FederateAmbassadorInteractionOccurrence implementation

public void
receiveInteraction(
   InteractionClassHandle interactionClass,
   ParameterHandleValueMap theParameters,
   byte[] userSuppliedTag,
   OrderType sentOrdering,
   TransportationType theTransport)
 throws InteractionClassNotRecognized,
          InteractionParameterNotRecognized,
          InteractionClassNotSubscribed,
          FederateInternalError
 { if (null != receiveInteractionValidator)
   receiveInteractionValidator.validate(
       interactionClass, theParameters, userSuppliedTag,
       sentOrdering, theTransport);
   doInteraction(interactionClass, theParameters, userSuppliedTag,
                 sentOrdering, theTransport, null, null, null, null);
 }

public void
receiveInteraction(
   InteractionClassHandle interactionClass,
   ParameterHandleValueMap theParameters,
   byte[] userSuppliedTag,
   OrderType sentOrdering,
   TransportationType theTransport,
   RegionHandleSet sentRegions)
 throws InteractionClassNotRecognized,
          InteractionParameterNotRecognized,
          InteractionClassNotSubscribed,
          FederateInternalError
 { if (null != receiveInteractionValidator)
   receiveInteractionValidator.validate(
       interactionClass, theParameters, userSuppliedTag,
       sentOrdering, theTransport, sentRegions);
   doInteraction(interactionClass, theParameters, userSuppliedTag,
                 sentOrdering, theTransport, null, null, null, sentRegions);
 }
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    if (null != receiveInteractionValidator)
        receiveInteractionValidator.validate(
            interactionClass, theParameters, userSuppliedTag,
            sentOrdering, theTransport, theTime, receivedOrdering);
        doInteraction(interactionClass, theParameters, userSuppliedTag,
            sentOrdering, theTransport, theTime, receivedOrdering, null, null);
}

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    if (null != receiveInteractionValidator)
        receiveInteractionValidator.validate(
            interactionClass, theParameters, userSuppliedTag,
            sentOrdering, theTransport, theTime, receivedOrdering, sentRegions);
        doInteraction(interactionClass, theParameters, userSuppliedTag,
            sentOrdering, theTransport, theTime, receivedOrdering, null, sentRegions);
    }
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
throws InteractionClassNotRecognized,
InteractionParameterNotRecognized,
InteractionClassNotSubscribed,
InvalidLogicalTime,
FederateInternalError

if (null != receiveInteractionValidator)
    receiveInteractionValidator.validate(
        interactionClass, theParameters, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        messageRetractionHandle);
    doInteraction(interactionClass, theParameters, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        messageRetractionHandle, null);
}

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
throws InteractionClassNotRecognized,
InteractionParameterNotRecognized,
InteractionClassNotSubscribed,
InvalidLogicalTime,
FederateInternalError

if (null != receiveInteractionValidator)
    receiveInteractionValidator.validate(
        interactionClass, theParameters, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        messageRetractionHandle, sentRegions);
    doInteraction(interactionClass, theParameters, userSuppliedTag,
        sentOrdering, theTransport, theTime, receivedOrdering,
        messageRetractionHandle, sentRegions);
}
/**
 * This implementation funnels all callbacks to the doInteraction method,
 * so you can either override the relevant callback(s) or doInteraction.
 * Missing arguments will be null.
 * Validation is handled by the callbacks.
 * @param interactionClass the InteractionClassHandle of the received interaction
 * @param theParameters a ParameterHandleValueMap specifying the interaction parameter values
 * @param userSuppliedTag a byte[] tag (this parameter may be null)
 * @param sentOrdering the OrderType the interaction was sent as
 * @param theTransport the TransportationType used to send the interaction
 * @param theTime the LogicalTime at which the interaction occurs
 * @param receivedOrdering the OrderType the passel was received as
 * @param messageRetractionHandle the MessageRetractionHandle of the message
 * @param sentRegions the RegionHandleSet used to send the interaction
 * @throws FederateInternalError should be thrown if something else goes wrong
 * @see FederateAmbassador#receiveInteraction receiveInteraction
 */
protected void doInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
    throws FederateInternalError
    {
    }
    }
// end FedAmbInteractionListener
The `FedAmbNameReservationListener` class implements the `FederateAmbassadorNameReservation` interface. It funnels both callbacks to a single method, `doReservation` (by appending a Boolean argument), allowing the programmer to either override the latter or the individual `FederateAmbassador objectInstanceNameReservationSucceeded/Failed` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doReservation`.

```java
// File: FedAmbNameReservationListener.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorNameReservation interface. This implementation funnels both callbacks to the doReservation method, so you can either override the objectInstanceNameReservationSucceeded/Failed methods or doReservation.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} (http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier)
 * @version 1.1
 */

class FedAmbNameReservationListener
    implements FederateAmbassadorNameReservation
{
    /**
     * The ValidateObjectInstanceNameReservationSucceeded validation interface.
     */
    public ValidateObjectInstanceNameReservationSucceeded
        objectInstanceNameReservationSucceededValidator;

    /**
     * The ValidateObjectInstanceNameReservationFailed validation interface.
     */
    public ValidateObjectInstanceNameReservationFailed
        objectInstanceNameReservationFailedValidator;

    /**
     * Default constructor. Creates an instance with no validators in place.
     */
    public FedAmbNameReservationListener()
    {
    }
}
```
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param objectInstanceNameReservationSucceededValidator A ValidateObjectInstanceNameReservationSucceeded interface
 * @param objectInstanceNameReservationFailedValidator A ValidateObjectInstanceNameReservationFailed interface
 */

public FedAmbNameReservationListener(
    ValidateObjectInstanceNameReservationSucceeded
    objectInstanceNameReservationSucceeded_Validator,
    ValidateObjectInstanceNameReservationFailed
    objectInstanceNameReservationFailed_Validator)
{

    objectInstanceNameReservationSucceededValidator
    objectInstanceNameReservationSucceeded_Validator;
    objectInstanceNameReservationFailedValidator
    objectInstanceNameReservationFailed_Validator;
}

// FederateAmbassadorNameReservation implementation

public void
objectInstanceNameReservationSucceeded(String objectName)
throws UnknownName, FederateInternalError
{
    if (null != objectInstanceNameReservationSucceededValidator)
        objectInstanceNameReservationSucceededValidator.validate(objectName);
        doReservation(objectName, true);
    }

public void
objectInstanceNameReservationFailed(String objectName)
throws UnknownName, FederateInternalError
{
    if (null != objectInstanceNameReservationFailedValidator)
        objectInstanceNameReservationFailedValidator.validate(objectName);
        doReservation(objectName, false);
    }

/**
 * This implementation simply funnels both callbacks to this method,
 * so you can either override the 
 * objectInstanceNameReservationSucceeded/Failed methods or this one.
 * Validation is handled by the callbacks.
 * @param objectName a {@link java.lang.String} holding the
 * requested object name
 * @param succeeded a boolean indicating whether the reservation
 * succeeded or not
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see RTIambassador#reserveObjectInstanceName
 * reserveObjectInstanceName
 * @see FederateAmbassador#objectInstanceNameReservationSucceeded
 * objectInstanceNameReservationSucceeded
 * @see FederateAmbassador#objectInstanceNameReservationFailed
 * objectInstanceNameReservationFailed
 */
 protected void doReservation(
     String objectName,
     boolean succeeded)
     throws FederateInternalError
 { }
} //end FedAmbNameReservationListener
The `FedAmbObjectClassAdvisoryResponder` class implements the `FederateAmbassadorObjectRegistrationAdvisory` interface. It funnels both callbacks to a single method, `doRelevance` (by appending a Boolean argument), allowing the programmer to either override the latter or the individual `FederateAmbassador start/stopRegistrationForObjectClass` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doRelevance`.

```java
// File: FedAmbObjectClassAdvisoryResponder.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorObjectRegistrationAdvisory interface.
 * This implementation funnels both callbacks to the doRelevance method,
 * so you can either override the start/stopRegistrationForObjectClass methods or doRelevance.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ([@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public class FedAmbObjectClassAdvisoryResponder
    implements FederateAmbassadorObjectRegistrationAdvisory
{
    /**
     * The ValidateStartRegistrationForObjectClass validation interface.
     */
    public ValidateStartRegistrationForObjectClass startRegistrationForObjectClassValidator;

    /**
     * The ValidateStopRegistrationForObjectClass validation interface.
     */
    public ValidateStopRegistrationForObjectClass stopRegistrationForObjectClassValidator;

    /**
     * Default constructor. Creates an instance with no validators in place.
     */
    public FedAmbObjectClassAdvisoryResponder()
    {
    }
```
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param startRegistrationForObjectClass_Validator A ValidateStartRegistrationForObjectClass interface
 * @param stopRegistrationForObjectClass_Validator A ValidateStopRegistrationForObjectClass interface
 */

public FedAmbObjectClassAdvisoryResponder(ValidateStartRegistrationForObjectClass startRegistrationForObjectClass_Validator,
                                        ValidateStopRegistrationForObjectClass stopRegistrationForObjectClass_Validator)
{
    this();
    startRegistrationForObjectClassValidator = startRegistrationForObjectClass_Validator;
    stopRegistrationForObjectClassValidator = stopRegistrationForObjectClass_Validator;
}

//FederateAmbassadorObjectRegistrationAdvisory implementation

public void startRegistrationForObjectClass(
                                        ObjectClassHandle theClass)
                                        throws ObjectClassNotPublished,
                                        FederateInternalError
{
    if (null != startRegistrationForObjectClassValidator)
    startRegistrationForObjectClassValidator.validate(theClass);
    doRelevance(theClass, true);
}

public void stopRegistrationForObjectClass(
                                        ObjectClassHandle theClass)
                                        throws ObjectClassNotPublished,
                                        FederateInternalError
{
    if (null != stopRegistrationForObjectClassValidator)
    stopRegistrationForObjectClassValidator.validate(theClass);
    doRelevance(theClass, false);
/**
 * This implementation simply funnels both callbacks to this
 * method,
 * so you can either override the
 * start/stopRegistrationForObjectClass methods or this one.
 * Validation is handled by the callbacks.
 * @param theClass the {link ObjectClassHandle} of the subject
 * object class
 * @param relevant a boolean indicating whether the object class is
 * relevant or not
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see FederateAmbassador#startRegistrationForObjectClass
 * startRegistrationForObjectClass
 * @see FederateAmbassador#stopRegistrationForObjectClass
 * stopRegistrationForObjectClass
 * @see RTIambassador#enableObjectClassRelevanceAdvisorySwitch
 * enableObjectClassRelevanceAdvisorySwitch
 * @see RTIambassador#disableObjectClassRelevanceAdvisorySwitch
 * disableObjectClassRelevanceAdvisorySwitch
 */
 protected void doRelevance(
     ObjectClassHandle theClass,
     boolean relevant)
 throws FederateInternalError
 {
 ...
 } //end FedAmbObjectClassAdvisoryResponder
The `FedAmbOwnershipListener` class implements the `FederateAmbassadorAttributeOwnership` interface. Of the nine callbacks, three (`informAttributeOwnership`, `attributeIsNotOwned`, and `attributeIsOwnedByRTI`) are funneled to a single method, `doInformOwnership` (by appending a couple of Boolean arguments and supplying a null for an eventual missing argument), allowing the programmer to either override the latter or the individual `FederateAmbassador` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doInformOwnership` (if applicable).

```java
// File: FedAmbOwnershipListener.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A null implementation of the `FederateAmbassadorAttributeOwnership` interface.
 * This implementation funnels the `informAttributeOwnership`, `attributeIsNotOwned`
 * and `attributeIsOwnedByRTI` callbacks to the `doInformOwnership` method,
 * so you can either override the relevant callback(s) or `doInformOwnership`.
 * @author {Daniel Thibault@DRDC-RDDC.gc.ca} Daniel U. Thibault
 * @version 1.1
 */
public class FedAmbOwnershipListener
    implements FederateAmbassadorAttributeOwnership
{
    /**
     * The `ValidateRequestAttributeOwnershipAssumption` validation interface.
     */
    public ValidateRequestAttributeOwnershipAssumption requestAttributeOwnershipAssumptionValidator;

    /**
     * The `ValidateRequestDivestitureConfirmation` validation interface.
     */
    public ValidateRequestDivestitureConfirmation requestDivestitureConfirmationValidator;
```
/**
 * The ValidateAttributeOwnershipAcquisitionNotification validation interface.
 */
public ValidateAttributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotificationValidator;

/**
 * The ValidateAttributeOwnershipUnavailable validation interface.
 */
public ValidateAttributeOwnershipUnavailable attributeOwnershipUnavailableValidator;

/**
 * The ValidateRequestAttributeOwnershipRelease validation interface.
 */
public ValidateRequestAttributeOwnershipRelease requestAttributeOwnershipReleaseValidator;

/**
 * The ValidateConfirmAttributeOwnershipAcquisitionCancellation validation interface.
 */
public ValidateConfirmAttributeOwnershipAcquisitionCancellation confirmAttributeOwnershipAcquisitionCancellationValidator;

/**
 * The ValidateInformAttributeOwnership validation interface.
 */
public ValidateInformAttributeOwnership informAttributeOwnershipValidator;

/**
 * The ValidateAttributeIsNotOwned validation interface.
 */
public ValidateAttributeIsNotOwned attributeIsNotOwnedValidator;

/**
 * The ValidateAttributeIsOwnedByRTI validation interface.
 */
public ValidateAttributeIsOwnedByRTI attributeIsOwnedByRTIVValidator;

/**
 * Default constructor. Creates an instance with no validators in place.
 */
public FedAmbOwnershipListener()
{}
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param requestAttributeOwnershipAssumption_Validator A
 * ValidateRequestAttributeOwnershipAssumption interface
 * @param requestDivestitureConfirmation_Validator A
 * ValidateRequestDivestitureConfirmation interface
 * @param attributeOwnershipAcquisitionNotification_Validator A
 * ValidateAttributeOwnershipAcquisitionNotification interface
 * @param attributeOwnershipUnavailable_Validator A
 * ValidateAttributeOwnershipUnavailable interface
 * @param requestAttributeOwnershipRelease_Validator A
 * ValidateRequestAttributeOwnershipRelease interface
 * @param confirmAttributeOwnershipAcquisitionCancellation_Validator A
 * ValidateConfirmAttributeOwnershipAcquisitionCancellation interface
 * @param informAttributeOwnership_Validator A
 * ValidateInformAttributeOwnership interface
 * @param attributeIsNotOwned_Validator A
 * ValidateAttributeIsNotOwned interface
 * @param attributeIsOwnedByRTI_Validator A
 * ValidateAttributeIsOwnedByRTI interface
 */
public FedAmbOwnershipListener(
    ValidateRequestAttributeOwnershipAssumption requestAttributeOwnershipAssumption_Validator,
    ValidateRequestDivestitureConfirmation requestDivestitureConfirmation_Validator,
    ValidateAttributeOwnershipAcquisitionNotification attributeOwnershipAcquisitionNotification_Validator,
    ValidateAttributeOwnershipUnavailable attributeOwnershipUnavailable_Validator,
    ValidateRequestAttributeOwnershipRelease requestAttributeOwnershipRelease_Validator,
    ValidateConfirmAttributeOwnershipAcquisitionCancellation confirmAttributeOwnershipAcquisitionCancellation_Validator,
    ValidateInformAttributeOwnership informAttributeOwnership_Validator,
    ValidateAttributeIsNotOwned attributeIsNotOwned_Validator,
    ValidateAttributeIsOwnedByRTI attributeIsOwnedByRTI_Validator)
{
    this();
    requestAttributeOwnershipAssumptionValidator =
    requestDivestitureConfirmationValidator =
    attributeOwnershipAcquisitionNotificationValidator =
    attributeOwnershipUnavailableValidator =
    requestAttributeOwnershipReleaseValidator =
    confirmAttributeOwnershipAcquisitionCancellationValidator =
    informAttributeOwnershipValidator =
    attributeIsNotOwnedValidator =
    attributeIsOwnedByRTIValidator =
    attributeIsOwnedByRTI_Validator;
// FederateAmbassadorAttributeOwnership implementation

class FederateAmbassadorAttributeOwnership {
    public void requestAttributeOwnershipAssumption(ObjectInstanceHandle theObject,
                                                     AttributeHandleSet offeredAttributes,
                                                     byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeAlreadyOwned,
                AttributeNotPublished,
                FederateInternalError
    {
        if (null != requestAttributeOwnershipAssumptionValidator)
            requestAttributeOwnershipAssumptionValidator.validate(theObject,
                                                                   offeredAttributes,
                                                                   userSuppliedTag);
    }

class FederateAmbassadorAttributeOwnership {
    public void requestDivestitureConfirmation(ObjectInstanceHandle theObject,
                                                 AttributeHandleSet offeredAttributes)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotOwned,
                AttributeDivestitureWasNotRequested,
                FederateInternalError
    {
        if (null != requestDivestitureConfirmationValidator)
            requestDivestitureConfirmationValidator.validate(theObject,
                                                               offeredAttributes);
    }

class FederateAmbassadorAttributeOwnership {
    public void attributeOwnershipAcquisitionNotification(ObjectInstanceHandle theObject,
                                                            AttributeHandleSet securedAttributes,
                                                            byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeAcquisitionWasNotRequested,
                AttributeAlreadyOwned,
                AttributeNotPublished,
                FederateInternalError
    {
        if (null != attributeOwnershipAcquisitionNotificationValidator)
            attributeOwnershipAcquisitionNotificationValidator.validate(theObject,
                                                                              securedAttributes,
                                                                              userSuppliedTag);
    }
}
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateInternalError
{
    if (null != attributeOwnershipUnavailableValidator)
        attributeOwnershipUnavailableValidator.validate(theObject, theAttributes);
}

public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    if (null != requestAttributeOwnershipReleaseValidator)
        requestAttributeOwnershipReleaseValidator.validate(theObject, candidateAttributes, userSuppliedTag);
}

public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError
{
    if (null != confirmAttributeOwnershipAcquisitionCancellationValidator)
        confirmAttributeOwnershipAcquisitionCancellationValidator.validate(theObject, theAttributes);
}
public void informAttributeOwnership(
    ObjectInstanceHandle theObject,
    AttributeHandle  theAttribute,
    FederateHandle    theOwner)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        FederateInternalError
{
    if (null != informAttributeOwnershipValidator)
        informAttributeOwnershipValidator.validate(theObject, theAttribute, theOwner);
    doInformOwnership(theObject, theAttribute, false, false, theOwner);
}
public void attributeIsNotOwned(
    ObjectInstanceHandle theObject,
    AttributeHandle  theAttribute)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        FederateInternalError
{
    if (null != attributeIsNotOwnedValidator)
        attributeIsNotOwnedValidator.validate(theObject, theAttribute);
    doInformOwnership(theObject, theAttribute, false, true, null);
}
public void attributeIsOwnedByRTI(
    ObjectInstanceHandle theObject,
    AttributeHandle  theAttribute)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        FederateInternalError
{
    if (null != attributeIsOwnedByRTIValidator)
        attributeIsOwnedByRTIValidator.validate(theObject, theAttribute);
    doInformOwnership(theObject, theAttribute, false, true, null);
}
/**
   * This implementation funnels the informAttributeOwnership,
   * attributeIsNotOwned and attributeIsOwnedByRTI callbacks to the doInformOwnership method,
   * so you can either override the relevant callback(s) or doInformOwnership.
   * Missing arguments will be null.
   * Validation is handled by the callbacks.
   * @param theObject the {@link ObjectInstanceHandle} of the Object
   * whose instance attribute ownership is reported
   * @param theAttribute the {@link AttributeHandle} of the Attribute
   * whose ownership is reported
   * @param isUnowned a boolean which is true if the attribute is
   * unowned (isOwnedByRTI will be false and theOwner will be null), false
   * otherwise
   * @param isOwnedByRTI a boolean which is true if the attribute is
   * owned by the RTI (isUnowned will be false and theOwner will be null), false
   * otherwise
   * @param theOwner the {@link FederateHandle} of the owning
   * federate (when isUnowned and isOwnedByRTI are false), null otherwise
   * @throws FederateInternalError should be thrown if something else
   * goes wrong
   * @see FederateAmbassador#informAttributeOwnership
   * informAttributeOwnership
   * @see FederateAmbassador#attributeIsNotOwned attributeIsNotOwned
   * @see FederateAmbassador#attributeIsOwnedByRTI attributeIsOwnedByRTI
   */
   public void doInformOwnership(
       final ObjectInstanceHandle theObject,
       final AttributeHandle theAttribute,
       final boolean isUnowned,
       final boolean isOwnedByRTI,
       final FederateHandle theOwner)
           throws FederateInternalError
   {
   }
}

@end FedAmbOwnershipListener
The FedAmbRemovalResponder class implements the FederateAmbassadorObjectRemoval interface. It funnels all three forms of the callback to a single method, doRemove (by supplying nulls for any missing arguments), allowing the programmer to either override the latter or the individual FederateAmbassador removeObjectInstance methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has a public member holding the validation interface reference. The callback methods invoke this if specified, before handing off to doRemove.

```
// File: FedAmbRemovalResponder.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * Null implementation of FederateAmbassadorObjectRemoval interface.
 * This implementation funnels all callbacks to the doRemove method,
 * so you can either override the relevant removeObjectInstance method(s) or doRemove.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ( {http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier} )
 * @version 1.1
 */
public class FedAmbRemovalResponder
    implements FederateAmbassadorObjectRemoval
{
    /**
     * The ValidateRemoveObjectInstance validation interface.
     */
    public ValidateRemoveObjectInstance removeObjectInstanceValidator;

    /**
     * Default constructor. Creates an instance with no validator in place.
     */
    public FedAmbRemovalResponder()
    {
    }
```
/**
 * Alternate constructor. Creates an instance with the specified
 * validator in place.
 * @param removeObjectInstance_Validator A
 * ValidateRemoveObjectInstance interface
 */
  
public
FedAmbRemovalResponder(
   ValidateRemoveObjectInstance removeObjectInstance_Validator)
{
   // this();
   removeObjectInstanceValidator = removeObjectInstance_Validator;
}

//FederateAmbassadorObjectRemoval implementation

public
void
removeObjectInstance(
   ObjectInstanceHandle theObject,
   byte[] userSuppliedTag,
   OrderType sentOrdering)
throws ObjectInstanceNotKnown,
   FederateInternalError
{
   if (null != removeObjectInstanceValidator)
   removeObjectInstanceValidator.validate(
      theObject, userSuppliedTag, sentOrdering);
   doRemove(theObject, userSuppliedTag, sentOrdering, null, null, null);
}

public
void
removeObjectInstance(
   ObjectInstanceHandle theObject,
   byte[] userSuppliedTag,
   OrderType sentOrdering,
   LogicalTime theTime,
   OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
   FederateInternalError
{
   if (null != removeObjectInstanceValidator)
   removeObjectInstanceValidator.validate(
      theObject, userSuppliedTag, sentOrdering, theTime,
      receivedOrdering);
   doRemove(theObject, userSuppliedTag, sentOrdering, theTime,
      receivedOrdering, null);
}
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
        InvalidLogicalTime,
        FederateInternalError
{
    if (null != removeObjectInstanceValidator)
        removeObjectInstanceValidator.validate(
            theObject, userSuppliedTag, sentOrdering, theTime,
            receivedOrdering, retractionHandle);
    doRemove(theObject, userSuppliedTag, sentOrdering, theTime,
        receivedOrdering, retractionHandle);
}

/**
 * This implementation funnels all removeObjectInstance callbacks to
 * this method,
 * so you can either override the relevant callback(s) or this
 * method.
 * Missing arguments will be null.
 * Validation is handled by the callbacks.
 * @param theObject the {ObjectInstanceHandle} of the
 * concerned object instance
 * @param userSuppliedTag a <code>byte[]</code> tag (this parameter
 * may be <code>null</code>)
 * @param sentOrdering the {OrderType} the message was sent as
 * @param theTime the {LogicalTime} at which the deletion occurs
 * @param receivedOrdering the {OrderType} the message was received as
 * @param retractionHandle the {MessageRetractionHandle} of the
 * message
 * @throws FederateInternalError should be thrown if something else
 * goes wrong
 * @see FederateAmbassador#removeObjectInstance
 * @removeObjectInstance
 */
protected void doRemove(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws FederateInternalError
{
}
//end FedAmbRemovalResponder
The `FedAmbRestoreListener` class implements the `FederateAmbassadorRestore` interface. Of the seven callbacks, two pairs (requestFederationRestoreSucceeded/Failed and federation[Not]Restored) are funneled to a single method each, doRequest and doRestore (by appending a Boolean argument in each case), allowing the programmer to either override the latter or the individual FederateAmbassador methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to doRequest or doRestore (if applicable).

```java
// File: FedAmbRestoreListener.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**<n
 * A null implementation of the FederateAmbassadorRestore interface.
 * This implementation funnels requestFederationRestoreSucceeded/Failed
to the doRequest method,
 * so you can either override those callbacks or doRequest.
 * Likewise, the federation[Not]Restored callbacks are funneled to the
doRestore method,
 * so you can either override those callbacks or doRestore.
 * @author {@link mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U.
Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC
Valcartier})
 * @version 1.1
 */
public class
FedAmbRestoreListener
    implements FederateAmbassadorRestore
{
    /**
     * The ValidateRequestFederationRestoreSucceeded validation
interface.
     */
    public ValidateRequestFederationRestoreSucceeded
        requestFederationRestoreSucceededValidator;

    /**
     * The ValidateRequestFederationRestoreFailed validation interface.
     */
    public ValidateRequestFederationRestoreFailed
        requestFederationRestoreFailedValidator;

    /**
     * The ValidateFederationRestoreBegun validation interface.
     */
    public ValidateFederationRestoreBegun
        federationRestoreBegunValidator;
```

/**
 * The ValidateInitiateFederateRestore validation interface.
 */
public ValidateInitiateFederateRestore initiateFederateRestoreValidator;

/**
 * The ValidateFederationRestored validation interface.
 */
public ValidateFederationRestored federationRestoredValidator;

/**
 * The ValidateFederationNotRestored validation interface.
 */
public ValidateFederationNotRestored federationNotRestoredValidator;

/**
 * The ValidateFederationRestoreStatusResponse validation interface.
 */
public ValidateFederationRestoreStatusResponse federationRestoreStatusResponseValidator;

/**
 * Default constructor. Creates an instance with no validators in place.
 */
public FedAmbRestoreListener() {
}
/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param requestFederationRestoreSucceeded_Validator A ValidateRequestFederationRestoreSucceeded interface
 * @param requestFederationRestoreFailed_Validator A ValidateRequestFederationRestoreFailed interface
 * @param federationRestoreBegun_Validator A ValidateFederationRestoreBegun interface
 * @param initiateFederateRestore_Validator A ValidateInitiateFederateRestore interface
 * @param federationRestored_Validator A ValidateFederationRestored interface
 * @param federationNotRestored_Validator A ValidateFederationNotRestored interface
 * @param federationRestoreStatusResponse_Validator A ValidateFederationRestoreStatusResponse interface
 */

public FedAmbRestoreListener(ValidateRequestFederationRestoreSucceeded requestFederationRestoreSucceeded_Validator,
ValidateRequestFederationRestoreFailed requestFederationRestoreFailed_Validator,
ValidateFederationRestoreBegun federationRestoreBegun_Validator,
ValidateInitiateFederateRestore initiateFederateRestore_Validator,
ValidateFederationRestored federationRestored_Validator,
ValidateFederationNotRestored federationNotRestored_Validator,
ValidateFederationRestoreStatusResponse federationRestoreStatusResponse_Validator)
{
    // this();
    requestFederationRestoreSucceededValidator = requestFederationRestoreSucceeded_Validator;
    requestFederationRestoreFailedValidator = requestFederationRestoreFailed_Validator;
    federationRestoreBegunValidator = federationRestoreBegun_Validator;
    initiateFederateRestoreValidator = initiateFederateRestore_Validator;
    federationRestoredValidator = federationRestored_Validator;
    federationNotRestoredValidator = federationNotRestored_Validator;
    federationRestoreStatusResponseValidator = federationRestoreStatusResponse_Validator;
}
//FederateAmbassadorRestore implementation

public void requestFederationRestoreSucceeded(
    String label)
    throws FederateInternalError
{
    if (null != requestFederationRestoreSucceededValidator) 
        requestFederationRestoreSucceededValidator.validate(label);
    doRequest(label, true);
}

public void requestFederationRestoreFailed(
    String label)
    throws FederateInternalError
{
    if (null != requestFederationRestoreFailedValidator) 
        requestFederationRestoreFailedValidator.validate(label);
    doRequest(label, false);
}

public void federationRestoreBegun()
    throws FederateInternalError
{
    if (null != federationRestoreBegunValidator) 
        federationRestoreBegunValidator.validate();
}

public void initiateFederateRestore(
    String label,
    FederateHandle federateHandle)
    throws SpecifiedSaveLabelDoesNotExist,
            CouldNotInitiateRestore,
            FederateInternalError
{
    if (null != initiateFederateRestoreValidator) 
        initiateFederateRestoreValidator.validate(label, federateHandle);
    throw new CouldNotInitiateRestore("FedAmbRestoreListener cannot
            initiate a federation restore");
}

public void federationRestored()
    throws FederateInternalError
{
    if (null != federationRestoredValidator) 
        federationRestoredValidator.validate();
    doRestore(null);
}
public void federationNotRestored(RestoreFailureReason reason) throws FederateInternalError
{
    if (null != federationNotRestoredValidator)
        federationNotRestoredValidator.validate(reason);
    doRestore(reason);
}

public void federationRestoreStatusResponse(FederateHandleRestoreStatusPair[] response) throws FederateInternalError
{
    if (null != federationRestoreStatusResponseValidator)
        federationRestoreStatusResponseValidator.validate(response);
}

/**
 * This implementation funnels the requestFederationRestoreSucceeded/Failed callbacks to this method,
 * so you can either override the relevant callback(s) or this method.
 * Validation is handled by the callbacks.
 * @param label a [link java.lang.String] holding the saved state's identifier
 * @param succeeded a boolean indicating whether the request succeeded or not
 * @param FederationInternalError should be thrown if something goes wrong
 * @see RTIambassador#requestFederationRestore
 * @see FederateAmbassador#requestFederationRestoreSucceeded
 * @see FederateAmbassador#requestFederationRestoreFailed
 */
protected void doRequest(String label, boolean succeeded) throws FederateInternalError
{
}
/**
 * This implementation funnels the federation(Not)Restored callbacks to this method,
 * so you can either override the relevant callback(s) or this method.
 * Validation is handled by the callbacks.
 * @param reason a @link RestoreFailureReason} specifying the reason for the failure;
 * if null, it means the restore succeeded
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see FederateAmbassador#federationRestored federationRestored
 * @see FederateAmbassador#federationNotRestored federationNotRestored
 */
 protected void doRestore(RestoreFailureReason reason)
 throws FederateInternalError
 }
The `FedAmbSaveListener` class implements the `FederateAmbassadorSave` interface. Of the four callbacks, one pair (federation[Not]Saved) is funnelled to a single method, `doSave` (by appending a Boolean argument), and the two forms of another (initiateFederateSave) are funnelled to another single method, `doInitiateSave` (by supplying a null for an eventual missing argument). This allows the programmer to either override the latter or the individual `FederateAmbassador` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doSave` or `doInitiateSave` (if applicable).

```java
// File: FedAmbSaveListener.java
package ca.gc.drdc_rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A null implementation of the FederateAmbassadorSave interface.
 * This implementation funnels the federation[Not]Saved callbacks to
 * the doSave method,
 * so you can either override the relevant callback(s) or doSave.
 * Likewise, the initiateFederateSave callbacks are funnelled to the
 * doInitiateSave method,
 * so you can either override the relevant callback(s) or
 * doInitiateSave.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier})
 * @version 1.1
 */

public class FedAmbSaveListener
    implements FederateAmbassadorSave
{
    /**
     * The ValidateInitiateFederateSave validation interface.
     */
    public ValidateInitiateFederateSave
        initiateFederateSaveValidator;

    /**
     * The ValidateFederationSaved validation interface.
     */
    public ValidateFederationSaved
        federationSavedValidator;

    /**
     * The ValidateFederationNotSaved validation interface.
     */
    public ValidateFederationNotSaved
        federationNotSavedValidator;
}
```
/**
 * The ValidateFederationSaveStatusResponse validation interface.
 */
public ValidateFederationSaveStatusResponse federationSaveStatusResponseValidator;

/**
 * Default constructor. Creates an instance with no validators in place.
 */
public FedAmbSaveListener()
{
}

/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param initiateFederateSave_Validator A ValidateFederationSave interface
 * @param federationSaved_Validator A ValidateFederationSaved interface
 * @param federationNotSaved_Validator A ValidateFederationNotSaved interface
 * @param federationSaveStatusResponse_Validator A ValidateFederationSaveStatusResponse interface
 */
public FedAmbSaveListener(
    ValidateInitiateFederateSave initiateFederateSave_Validator,
    ValidateFederationSaved federationSaved_Validator,
    ValidateFederationNotSaved federationNotSaved_Validator,
    ValidateFederationSaveStatusResponse federationSaveStatusResponse_Validator)
{
    // this();
    initiateFederateSaveValidator = initiateFederateSave_Validator;
    federationSavedValidator = federationSaved_Validator;
    federationNotSavedValidator = federationNotSaved_Validator;
    federationSaveStatusResponseValidator = federationSaveStatusResponse_Validator;
}
//FederateAmbassadorSave implementation

public void initiateFederateSave(
    String label)
throws UnableToPerformSave,
       FederateInternalError
{
    if (null != initiateFederateSaveValidator)
        initiateFederateSaveValidator.validate(label);
    doInitiateSave(label, null);
}

public void initiateFederateSave(
    String label, 
    LogicalTime time)
throws InvalidLogicalTime, 
       UnableToPerformSave, 
       FederateInternalError
{
    if (null != initiateFederateSaveValidator)
        initiateFederateSaveValidator.validate(label, time);
    doInitiateSave(label, time);
}

public void federationSaved()
throws FederateInternalError
{
    if (null != federationSavedValidator)
        federationSavedValidator.validate();
    doSave(null);
}

public void federationNotSaved(
    SaveFailureReason reason)
throws FederateInternalError
{
    if (null != federationNotSavedValidator)
        federationNotSavedValidator.validate(reason);
    doSave(reason);
}

public void federationSaveStatusResponse(
    FederateHandleSaveStatusPair[] response)
throws FederateInternalError
{
    if (null != federationSaveStatusResponseValidator)
        federationSaveStatusResponseValidator.validate(response);
}
/**
 * This implementation funnels the initiateFederateSave callbacks to this method, so you can either override the relevant callback(s) or this method. Missing arguments will be null. Validation is handled by the callbacks.
 * @param label a String holding the saved state's identifier
 * @param time a LogicalTime specifying when the save was scheduled
 * @throws UnableToPerformSave should be thrown if the save operation seems doomed
 * @throws FederateInternalError should be thrown if something else goes wrong
 *
 */
protected void doInitiateSave(String label, LogicalTime time)
throws UnableToPerformSave, FederateInternalError
{
    //end FedAmbSaveListener
}
The `FedAmbSynchronizationListener` class implements the `FederateAmbassadorSynchronization` interface. Of the four callbacks, it funnels a pair (synchronizationPointRegistrationSucceeded/Failed) to a single method, `doRegistration` (by supplying a null for an eventual missing argument), allowing the programmer to either override the latter or the individual `FederateAmbassador` methods, depending on what is more convenient. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified, before handing off to `doRegistration` (if applicable).

```java
// File: FedAmbSynchronizationListener.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A null implementation of the FederateAmbassadorSynchronization interface.
 * Note that a more satisfactory implementation would respond to the announceSynchronizationPoint
 * callback with
 * _RTIambassador.synchronizationPointAchieved(synchronizationPointLabel)_
 * ;
 * but this cannot be done here as we're lacking an RTIambassador reference.
 * This implementation also funnels synchronizationPointRegistrationSucceeded/Failed to the doRegistration
 * method,
 * so you can either override those callbacks or doRegistration.
 * @author [mailto:Daniels.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([link http://www.valcartier.drdc-rddc.gc.ca RDDC Valcartier])
 * @version 1.1
 */

public class FedAmbSynchronizationListener
    implements FederateAmbassadorSynchronization
{
    /**
     * The ValidateSynchronizationPointRegistrationSucceeded validation
     * interface.
     */
    public ValidateSynchronizationPointRegistrationSucceeded
    synchronizationPointRegistrationSucceededValidator;

    /**
     * The ValidateSynchronizationPointRegistrationFailed validation
     * interface.
     */
    public ValidateSynchronizationPointRegistrationFailed
    synchronizationPointRegistrationFailedValidator;
```
/**
 * The ValidateAnnounceSynchronizationPoint validation interface.
 */
public ValidateAnnounceSynchronizationPoint
announceSynchronizationPointValidator;

/**
 * The ValidateFederationSynchronized validation interface.
 */
public ValidateFederationSynchronized
federationSynchronizedValidator;

/**
 * Default constructor. Creates an instance with no validators in place.
 */
public
FedAmbSynchronizationListener()
{
}

/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param synchronizationPointRegistrationSucceeded_Validator A ValidateSynchronizationPointRegistrationSucceeded interface
 * @param synchronizationPointRegistrationFailed_Validator A ValidateSynchronizationPointRegistrationFailed interface
 * @param announceSynchronizationPoint_Validator A ValidateAnnounceSynchronizationPoint interface
 * @param federationSynchronized_Validator A ValidateFederationSynchronized interface
 */
public
FedAmbSynchronizationListener(
ValidateSynchronizationPointRegistrationSucceeded
synchronizationPointRegistrationSucceeded_Validator,
ValidateSynchronizationPointRegistrationFailed
synchronizationPointRegistrationFailed_Validator,
ValidateAnnounceSynchronizationPoint
announceSynchronizationPoint_Validator,
ValidateFederationSynchronized
federationSynchronized_Validator)
{
    this();
    synchronizationPointRegistrationSucceededValidator =
    synchronizationPointRegistrationSucceeded_Validator;
    synchronizationPointRegistrationFailedValidator =
    synchronizationPointRegistrationFailed_Validator;
    announceSynchronizationPointValidator =
    announceSynchronizationPoint_Validator;
    federationSynchronizedValidator =
    federationSynchronized_Validator;
}
//FederateAmbassadorSynchronization implementation

public void
synchronizationPointRegistrationSucceeded(
    String synchronizationPointLabel)
throws FederateInternalError
{
    if (null != synchronizationPointRegistrationSucceededValidator)
        synchronizationPointRegistrationSucceededValidator.validate(synchronizationPointLabel);
    doRegistration(synchronizationPointLabel, null);
}

public void
synchronizationPointRegistrationFailed(
    String synchronizationPointLabel,
    SynchronizationPointFailureReason reason)
throws FederateInternalError
{
    if (null != synchronizationPointRegistrationFailedValidator)
        synchronizationPointRegistrationFailedValidator.validate(synchronizationPointLabel, reason);
    doRegistration(synchronizationPointLabel, reason);
}

public void
announceSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag)
throws FederateInternalError
{
    if (null != announceSynchronizationPointValidator)
        announceSynchronizationPointValidator.validate(synchronizationPointLabel, userSuppliedTag);
    //A true null implementation would immediately fire back this RTIambassador call:
    //_RTIambassador.synchronizationPointAchieved(synchronizationPointLabel);
}

public void
federationSynchronized(
    String synchronizationPointLabel)
throws FederateInternalError
{
    if (null != federationSynchronizedValidator)
        federationSynchronizedValidator.validate(synchronizationPointLabel);
}
/**
 * This implementation funnels the synchronizationPointRegistrationSucceeded/Failed callbacks to this method,
 * so you can either override the relevant callback(s) or this method.
 * Validation is handled by the callbacks.
 * @param synchronizationPointLabel a @link java.lang.String giving the synchronization point's identifier
 * @param reason a @link SynchronizationPointFailureReason specifying what went wrong;
 * if null, it means the registration succeeded
 * @throws FederateInternalError should be thrown if something goes wrong
 * @see RTIambassador#registerFederationSynchronizationPoint
 * @see FederateAmbassador#synchronizationPointRegistrationSucceeded
 * @see FederateAmbassador#synchronizationPointRegistrationFailed
 */
protected void doRegistration(String synchronizationPointLabel,
               SynchronizationPointFailureReason reason)
    throws FederateInternalError
{
}
//end FedAmbSynchronizationListener
The **FedAmbTimeListener** class implements the **FederateAmbassadorTime** interface. Being a null implementation, it does nothing with the callbacks.

The class has public members holding the validation interface references. The callback methods invoke these if specified.

```java
// File: FedAmbTimeListener.java
package ca.gc.drdc.rddc.hla.rti1516.FedAmb;

import hla.rti1516.*;

/**
 * A null implementation of the FederateAmbassadorTime interface.
 *
 * @version 1.1
 */
public class FedAmbTimeListener
    implements FederateAmbassadorTime {
    /**
     * The ValidateRequestRetraction validation interface.
     */
    public ValidateRequestRetraction requestRetractionValidator;

    /**
     * The ValidateTimeAdvanceGrant validation interface.
     */
    public ValidateTimeAdvanceGrant timeAdvanceGrantValidator;

    /**
     * The ValidateTimeConstrainedEnabled validation interface.
     */
    public ValidateTimeConstrainedEnabled timeConstrainedEnabledValidator;
```
/**
 * The ValidateTimeRegulationEnabled validation interface.
 */
public ValidateTimeRegulationEnabled timeRegulationEnabledValidator;

/**
 * Default constructor. Creates an instance with no validators in place.
 */
public FedAmbTimeListener()
{
}

/**
 * Alternate constructor. Creates an instance with the specified validators in place.
 * @param requestRetraction_Validator A ValidateRequestRetraction interface
 * @param timeAdvanceGrant_Validator A ValidateTimeAdvanceGrant interface
 * @param timeConstrainedEnabled_Validator A ValidateTimeConstrainedEnabled interface
 * @param timeRegulationEnabled_Validator A ValidateTimeRegulationEnabled interface
 */
public FedAmbTimeListener(
    ValidateRequestRetraction requestRetraction_Validator, 
    ValidateTimeAdvanceGrant timeAdvanceGrant_Validator, 
    ValidateTimeConstrainedEnabled timeConstrainedEnabled_Validator, 
    ValidateTimeRegulationEnabled timeRegulationEnabled_Validator)
{
    this();
    requestRetractionValidator = requestRetraction_Validator;
    timeAdvanceGrantValidator = timeAdvanceGrant_Validator;
    timeConstrainedEnabledValidator = timeConstrainedEnabled_Validator;
    timeRegulationEnabledValidator = timeRegulationEnabled_Validator;
}

//FederateAmbassadorTime implementation

public void timeRegulationEnabled(
    LogicalTime time)
throws InvalidLogicalTime, 
    NoRequestToEnableTimeRegulationWasPending, 
    FederateInternalError
{
    if (null != timeRegulationEnabledValidator)
        timeRegulationEnabledValidator.validate(time);
}
public void timeConstrainedEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
    NoRequestToEnableTimeConstrainedWasPending,
    FederateInternalError
{
    if (null != timeConstrainedEnabledValidator)
        timeConstrainedEnabledValidator.validate(time);
}

public void timeAdvanceGrant(
    LogicalTime theTime)
throws InvalidLogicalTime,
    JoinedFederateIsNotInTimeAdvancingState,
    FederateInternalError
{
    if (null != timeAdvanceGrantValidator)
        timeAdvanceGrantValidator.validate(theTime);
}

public void requestRetraction(
    Message RetractionHandle theHandle)
throws FederateInternalError
{
    if (null != requestRetractionValidator)
        requestRetractionValidator.validate(theHandle);
}

//end FedAmbTimeListener
The FedAmbWrapper class implements the FederateAmbassador interface by serving as a dispatching layer between the callbacks and the event handlers. For each of the FederateAmbassador... sub-interfaces, FedAmbWrapper maintains a hash map of implementing objects, keyed by what is usually an object instance handle. When no handlers are registered for a given callback set, FedAmbWrapper acts as a null implementation.

Handlers can be registered in hierarchical fashion, at the object instance, object class, and default levels. For example, when a reflectAttributeValues callback is received, FedAmbWrapper will first look for a handler registered at the specific object instance handle received; failing that, it will look for a handler registered to the object instance’s class; failing that again, it will look for a default handler (registered under the null key). Sub-interfaces keyed to object class handles (e.g. ObjectDiscovery) start the look-up process at the class level. The Synchronization and NameReservation sub-interfaces use strings (synchronization labels and object instance names, respectively) as keys. The Restore, Save, and Time sub-interfaces, finally, admit a single handler.

Because it incurs an overhead that may be significant, FedAmbWrapper should be used only when flexibility is more important than performance.
/**
 * This FederateAmbassador Wrapper class serves as a dispatcher;
 * before doing an RTIambassador call that expects FederateAmbassador
 * callbacks, you register a listener or responder with the
 * appropriate class of service and the designated key. When the
 * expected callback occurs, the FedAmbWrapper dispatches it to the
 * specified listener/responder. It is the listener/responder's
 * responsibility to validate the arguments it receives (throwing any
 * of the prescribed exceptions as required) and then thread-off any
 * remaining work.
 * <p>
 * Services that are invoked at the Instance level can have
 * listeners/responders registered at that level or at the Class
 * level. As a general rule, FedAmbWrapper will look for a registered
 * listener in the following sequence:
 * <ul>
 * <li> at the Instance level (ObjectInstanceHandle key);
 * <li> at the known Object Class level (ObjectClassHandle key);
 * <li> at the default level listener (null key); or
 * <li> do nothing.
 * </ul>
 * Notable exceptions are the discoveryListener, interactionListener,
 * interactionAdvisoryResponder and registrationResponder services
 * (which stop at the Class level); the nameReservationListener
 * service (which can only be refined at the name level); the
 * federationSynchronizationListener service (which can only be
 * refined at the synchronization label level); and the
 * federationSaveListener, federationRestoreListener and timeListener
 * services (which are undifferentiated).
 * <p>
 * Note that, because of the overloading, you'll need to typecast the
 * null to either ObjectClassHandle or ObjectInstanceHandle for the
 * set* method to be disambiguated. For example:
 * <p>
 * _fedAmbWrapper.setAttributeScopeListener((ObjectClassHandle)null,
 * new MyFederateAmbassadorAttributeScopeAdvisoryClient
 * attributeScopeListener());
 * <p>
 * Under IEEE 1516, the RTI is normally multi-threaded and thus each
 * FederateAmbassador callback occurs from a "Federate Service Thread"
 * belonging to the RTI (one per federate).
 * <p>
public class FedAmbWrapper
    implements FederateAmbassador
{
    /** We need an RTI ambassador reference for some services. */
    private RTIambassador _rti;

    /** FederateAmbassadorAttributeScopeAdvisoryClient HashMap. */
    protected HashMap attributeScopeListeners;

    /** FederateAmbassadorAttributeScopeAdvisoryServer HashMap. */
    protected HashMap attributeScopeResponders;

    /** FederateAmbassadorAttributeUpdateClient HashMap. */
    protected HashMap attributeUpdateListeners;

    /** FederateAmbassadorAttributeUpdateServer HashMap. */
    protected HashMap attributeUpdateResponders;

    /** FederateAmbassadorObjectDiscovery HashMap. */
    protected HashMap discoveryListeners;

    /** FederateAmbassadorObjectRemoval HashMap. */
    protected HashMap removalResponders;

    /** FederateAmbassadorRestore HashMap. */
    protected HashMap federationRestoreListeners;

    /** FederateAmbassadorSave HashMap. */
    protected HashMap federationSaveListeners;

    /** FederateAmbassadorSynchronization HashMap. */
    protected HashMap federationSynchronizationListeners;

    /** FederateAmbassadorInteractionAdvisory HashMap. */
    protected HashMap interactionAdvisoryResponders;

    /** FederateAmbassadorInteractionOccurrence HashMap. */
    protected HashMap interactionListeners;

    /** FederateAmbassadorNameReservation HashMap. */
    protected HashMap nameReservationListeners;
/** FederateAmbassadorAttributeOwnership HashMap. */
protected HashMap
ownershipListeners;

/** FederateAmbassadorObjectRegistrationAdvisory HashMap. */
protected HashMap
registrationResponders;

/** FederateAmbassadorTime HashMap. */
protected HashMap
timeListeners;

/**
 * Constructor.
 * @param rti an RTIambassador to be used by some services
 */
public
FedAmbWrapper(RTIambassador rti)
{
    _rti = rti;
    attributeScopeListeners = new HashMap();
    attributeScopeResponders = new HashMap();
    attributeUpdateResponders = new HashMap();
    attributeUpdateListeners = new HashMap();
    discoveryListeners = new HashMap();
    discoveryListeners = new HashMap();
    removalResponders = new HashMap();
    federationRestoreListeners = new HashMap();
    federationSaveListeners = new HashMap();
    federationSynchronizationListeners = new HashMap();
    interactionAdvisoryResponders = new HashMap();
    interactionListeners = new HashMap();
    nameReservationListeners = new HashMap();
    ownershipListeners = new HashMap();
    registrationResponders = new HashMap();
    timeListeners = new HashMap();
}
/**
 * Associates an attributeScopeListener with a specified ObjectInstanceHandle.
 * The null ObjectInstanceHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeScopeAdvisoryClient unassociates the ObjectInstanceHandle.
 * @param theObject the ObjectInstanceHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param attributeScopeListener the FederateAmbassadorAttributeScopeAdvisoryClient to associate with
 * the specified ObjectInstanceHandle; if null, behaves as a remove();
 * @return the previous attributeScopeListener associated with the ObjectInstanceHandle
 * (null if there was no previous mapping)
 */
public Object
setAttributeScopeListener(
    ObjectInstanceHandle theObject,
    FederateAmbassadorAttributeScopeAdvisoryClient
    attributeScopeListener)
{
    if (null==attributeScopeListener)
    {
        return attributeScopeListeners.remove(theObject);
    }
    else
    {
        return attributeScopeListeners.put(theObject,
            attributeScopeListener);
    }
}
/**
 * Associates an attributeScopeListener with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeScopeAdvisoryClient unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param attributeScopeListener the FederateAmbassadorAttributeScopeAdvisoryClient to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous attributeScopeListener associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */

public Object
setAttributeScopeListener(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorAttributeScopeAdvisoryClient attributeScopeListener)
{
    if (null==attributeScopeListener)
    {
        return attributeScopeListeners.remove(theObjectClass);
    }
    else
    {
        return attributeScopeListeners.put(theObjectClass, attributeScopeListener);
    }
}
/**
 * Associates an attributeScopeResponder with a specified ObjectInstanceHandle. 
 * The null ObjectInstanceHandle designates the default (fall-through) responder. 
 * A null FederateAmbassadorAttributeScopeAdvisoryServer unassociates the ObjectInstanceHandle. 
 * @param theObject the ObjectInstanceHandle with which to associate the responder; 
 * the null value is the default (fall-through) responder 
 * @param attributeScopeResponder the FederateAmbassadorAttributeScopeAdvisoryServer to associate with 
 * the specified ObjectInstanceHandle; if null, behaves as a remove() 
 * @return the previous attributeScopeResponder associated with the ObjectInstanceHandle 
 * (null if there was no previous mapping) 
 */
public Object setAttributeScopeResponder(
    ObjectInstanceHandle theObject, 
    FederateAmbassadorAttributeScopeAdvisoryServer attributeScopeResponder)
{
    if (null==attributeScopeResponder)
    {
        return attributeScopeResponders.remove(theObject);
    }
    else
    {
        return attributeScopeResponders.put(theObject, attributeScopeResponder);
    }
}
/**
 * Associates an attributeScopeResponder with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) responder.
 * A null FederateAmbassadorAttributeScopeAdvisoryServer unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the responder;
 * the null value is the default (fall-through) responder
 * @param attributeScopeResponder the FederateAmbassadorAttributeScopeAdvisoryServer to associate with the specified ObjectClassHandle;
 * if null, behaves as a remove()
 * @return the previous attributeScopeResponder associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */
public Object
setAttributeScopeResponder(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorAttributeScopeAdvisoryServer attributeScopeResponder)
{
    if (null==attributeScopeResponder)
    {
        return attributeScopeResponders.remove(theObjectClass);
    }
    else
    {
        return attributeScopeResponders.put(theObjectClass, attributeScopeResponder);
    }
}
/**
 * Associates an attributeUpdateListener with a specified ObjectInstanceHandle.
 * The null ObjectInstanceHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeUpdateClient unassociates the ObjectInstanceHandle.
 * @param theObject the ObjectInstanceHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param attributeUpdateListener the FederateAmbassadorAttributeUpdateClient to associate with
 * the specified ObjectInstanceHandle; if null, behaves as a remove()
 * @return the previous attributeUpdateListener associated with the ObjectInstanceHandle
 * (null if there was no previous mapping)
 */

public Object setAttributeUpdateListener(
    ObjectInstanceHandle theObject,
    FederateAmbassadorAttributeUpdateClient attributeUpdateListener
)
{
    if (null==attributeUpdateListener)
    {
        return attributeUpdateListeners.remove(theObject);
    }
    else
    {
        return attributeUpdateListeners.put(theObject, attributeUpdateListener);
    }
}
/**
 * Associates an attributeUpdateListener with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeUpdateClient unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the listener;
 * @param attributeUpdateListener the FederateAmbassadorAttributeUpdateClient to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous attributeUpdateListener associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */

public Object setAttributeUpdateListener(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorAttributeUpdateClient attributeUpdateListener)
{
    if (null==attributeUpdateListener)
    {
        return attributeUpdateListeners.remove(theObjectClass);
    }
    else
    {
        return attributeUpdateListeners.put(theObjectClass, attributeUpdateListener);
    }
}
/**
 * Associates an attributeUpdateResponder with a specified ObjectInstanceHandle.
 * The null ObjectInstanceHandle designates the default (fall-through) responder.
 * A null FederateAmbassadorAttributeUpdateServer unassociates the ObjectInstanceHandle.
 * @param theObject the ObjectInstanceHandle with which to associate the responder;
 * @param attributeUpdateResponder the FederateAmbassadorAttributeUpdateServer to associate with
 * the specified ObjectInstanceHandle; if null, behaves as a remove();
 * @return the previous attributeUpdateResponder associated with the ObjectInstanceHandle
 * (null if there was no previous mapping)
 */
public Object
setAttributeUpdateResponder(
   ObjectInstanceHandle theObject,
   FederateAmbassadorAttributeUpdateServer attributeUpdateResponder)
{
    if (null==attributeUpdateResponder)
    {
      return attributeUpdateResponders.remove(theObject);
    }
    else
    {
      return attributeUpdateResponders.put(theObject, attributeUpdateResponder);
    }
}
/**
 * Associates an attributeUpdateResponder with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) responder.
 * A null FederateAmbassadorAttributeUpdateServer unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the responder;
 * the null value is the default (fall-through) responder
 * @param attributeUpdateResponder the FederateAmbassadorAttributeUpdateServer to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous attributeUpdateResponder associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */

public Object setAttributeUpdateResponder(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorAttributeUpdateServer attributeUpdateResponder)
{
    if (null==attributeUpdateResponder)
    {
        return attributeUpdateResponders.remove(theObjectClass);
    }
    else
    {
        return attributeUpdateResponders.put(theObjectClass,
            attributeUpdateResponder);
    }
}
/**
 * Associates a discoveryListener with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through)
 * listener.
 * A null FederateAmbassadorObjectDiscovery unassociates the
 * ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to
 * associate the listener;
 * @param discoveryListener the FederateAmbassadorObjectDiscovery
 * to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous discoveryListener associated with the
 * ObjectClassHandle
 * (null if there was no previous mapping)
 */
public Object
setDiscoveryListener(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorObjectDiscovery discoveryListener)
{
    if (null==discoveryListener)
    {
        return discoveryListeners.remove(theObjectClass);
    }
    else
    {
        return discoveryListeners.put(theObjectClass,
            discoveryListener);
    }
}
/**
 * Associates a removalResponder with a specified ObjectInstanceHandle.
 * The null ObjectInstanceHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorObjectRemoval unassociates the ObjectInstanceHandle.
 * @param theObject the ObjectInstanceHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param removalResponder the FederateAmbassadorObjectRemoval to associate with
 * the specified ObjectInstanceHandle; if null, behaves as a remove()
 * @return the previous removalResponder associated with the ObjectInstanceHandle
 * (null if there was no previous mapping)
 */

public Object setRemovalResponder(
    ObjectInstanceHandle theObject,
    FederateAmbassadorObjectRemoval removalResponder)
{
    if (null==removalResponder)
    {
        return removalResponders.remove(theObject);
    }
    else
    {
        return removalResponders.put(theObject, removalResponder);
    }
}
/**
 * Associates a removalResponder with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorObjectRemoval unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param removalResponder the FederateAmbassadorObjectRemoval to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous removalResponder associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */

public Object setRemovalResponder(
    ObjectClassHandle theObjectClass,
    FederateAmbassadorObjectRemoval removalResponder)
{
    if (null==removalResponder)
    {
        return removalResponders.remove(theObjectClass);
    }
    else
    {
        return removalResponders.put(theObjectClass, removalResponder);
    }
}
/**
 * Sets the federationRestoreListener.
 * Unlike the other callback subsets, no dispatching occurs (there
 * can be only one listener).
 * A null FederateAmbassadorRestore is allowed.
 * @param federationRestoreListener the FederateAmbassadorRestore
to associate;
 * if null, behaves as a remove()
 * @return the previous federationRestoreListener (null if there
 * was no previous mapping)
 */
public Object
setFederationRestoreListener(
    FederateAmbassadorRestore federationRestoreListener)
{
    if (null==federationRestoreListener)
    {
        return federationRestoreListeners.remove(null);
    }
    else
    {
        return federationRestoreListeners.put(null,
            federationRestoreListener);
    }
}

/**
 * Sets the federationSaveListener.
 * Unlike the other callback subsets, no dispatching occurs (there
 * can be only one listener).
 * A null FederateAmbassadorSave is allowed.
 * @param federationSaveListener the FederateAmbassadorSave to
 * associate;
 * if null, behaves as a remove()
 * @return the previous federationSaveListener (null if there
 * was no previous mapping)
 */
public Object
setFederationSaveListener(
    FederateAmbassadorSave federationSaveListener)
{
    if (null==federationSaveListener)
    {
        return federationSaveListeners.remove(null);
    }
    else
    {
        return federationSaveListeners.put(null,
            federationSaveListener);
    }
}
/**
 * Associates a federationSynchronizationListener with a specified synchronization label.
 * The null String (not the empty String) designates the default (fall-through) listener.
 * A null FederateAmbassadorSynchronization unassociates the String.
 * @param synchronizationPointLabel the String with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param federationSynchronizationListener the FederateAmbassadorSynchronization to associate with
 * the specified String; if null, behaves as a remove()
 * @return the previous federationSynchronizationListener associated with the String
 * (null if there was no previous mapping)
 */
 public Object setFederationSynchronizationListener(
     String synchronizationPointLabel,
     FederateAmbassadorSynchronization federationSynchronizationListener)
 {
     if (null==federationSynchronizationListener)
     {
         return federationSynchronizationListeners.remove(synchronizationPointLabel);
     }
     else
     {
         return federationSynchronizationListeners.put(synchronizationPointLabel,
             federationSynchronizationListener);
     }
 }
//FederateAmbassadorInteractionAdvisory

/**
 * Associates an interactionAdvisoryResponder with a specified InteractionClassHandle.
 * The null InteractionClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorInteractionAdvisory unassociates the InteractionClassHandle.
 * @param interactionClass the InteractionClassHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param interactionAdvisoryResponder the FederateAmbassadorInteractionAdvisory to associate with
 * the specified interactionClass; if null, behaves as a remove();
 * @return the previous interactionAdvisoryResponder associated with the interactionClass
 * (null if there was no previous mapping)
 */
public Object setInteractionAdvisoryResponder(
    InteractionClassHandle interactionClass,
    FederateAmbassadorInteractionAdvisory interactionAdvisoryResponder)
{
    if (null==interactionAdvisoryResponder)
    {
        return interactionAdvisoryResponders.remove(interactionClass);
    }
    else
    {
        return interactionAdvisoryResponders.put(interactionClass,
            interactionAdvisoryResponder);
    }
}
/**
 * Associates an interactionListener with a specified InteractionClassHandle.
 * The null InteractionClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorInteractionOccurrence unassociates the InteractionClassHandle.
 * @param interactionClass the InteractionClassHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param interactionListener the FederateAmbassadorInteractionOccurrence to associate with
 * the specified interactionClass; if null, behaves as a remove()
 * @return the previous interactionListener associated with the interactionClass
 * (null if there was no previous mapping)
 */
public Object setInteractionListener(
    InteractionClassHandle interactionClass, 
    FederateAmbassadorInteractionOccurrence interactionListener)
{
    if (null==interactionListener)
    {
        return interactionListeners.remove(interactionClass);
    }
    else
    {
        return interactionListeners.put(interactionClass, 
            interactionListener);
    }
}
/**
 * Associates a nameReservationListener with a specified String.  
 * The null String (not the empty String) designates the default  
 * (fall-through) listener.  
 * A null FederateAmbassadorNameReservation unassociates the  
 * String.  
 * @param objectName the String with which to associate the  
 * listener;  
 * the null value is the default (fall-through) listener  
 * @param nameReservationListener the  
 * FederateAmbassadorNameReservation to associate with  
 * the specified objectName; if null, behaves as a remove()  
 * @return the previous nameReservationListener associated with the  
 * objectName  
 * (null if there was no previous mapping)
 */
public Object
setNameReservationListener(
    String objectName,
    FederateAmbassadorNameReservation nameReservationListener)
{
    if (null==nameReservationListener)
    {
        return nameReservationListeners.remove(objectName);
    }
    else
    {
        return nameReservationListeners.put(objectName,
                                          nameReservationListener);
    }
}
/**
 * Associates an InstanceAttributeOwnershipListener with a specified ObjectInstanceHandle.
 * The null ObjectInstanceHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeOwnership unassociates the ObjectInstanceHandle.
 * @param theObject the ObjectInstanceHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param ownershipListener the FederateAmbassadorAttributeOwnership to associate with
 * the specified ObjectInstanceHandle; if null, behaves as a remove()
 * @return the previous ownershipListener associated with the theObject
 * (null if there was no previous mapping)
 */
public Object setOwnershipListener(
    ObjectInstanceHandle theObject,
    FederateAmbassadorAttributeOwnership ownershipListener)
{
    if (null==ownershipListener)
    {
        return ownershipListeners.remove(theObject);
    }
    else
    {
        return ownershipListeners.put(theObject, ownershipListener);
    }
}
/**
 * Associates an InstanceAttributeOwnershipListener with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) listener.
 * A null FederateAmbassadorAttributeOwnership unassociates the ObjectClassHandle.
 * @param theObjectClass the ObjectClassHandle with which to associate the listener;
 * the null value is the default (fall-through) listener
 * @param ownershipListener the FederateAmbassadorAttributeOwnership to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous ownershipListener associated with the ObjectClassHandle
 * (null if there was no previous mapping)
 */

public Object
setOwnershipListener(
   ObjectClassHandle theObjectClass,
   FederateAmbassadorAttributeOwnership ownershipListener)
{
   if (null==ownershipListener)
   {
      return ownershipListeners.remove(theObjectClass);
   }
   else
   {
      return ownershipListeners.put(theObjectClass,
                                     ownershipListener);
   }
}
/**
 * Associates a RegistrationAdvisoryResponder with a specified ObjectClassHandle.
 * The null ObjectClassHandle designates the default (fall-through) responder.
 * A null FederateAmbassadorObjectRegistrationAdvisory unassociates the ObjectClassHandle.
 * @param theClass the ObjectClassHandle with which to associate the responder;
 * the null value is the default (fall-through) responder
 * @param registrationResponder the FederateAmbassadorObjectRegistrationAdvisory to associate with
 * the specified ObjectClassHandle; if null, behaves as a remove()
 * @return the previous registrationResponder associated with the theClass
 * (null if there was no previous mapping)
 */

public Object setRegistrationResponder(
    ObjectClassHandle theClass,
    FederateAmbassadorObjectRegistrationAdvisory registrationResponder)
{
    if (null==registrationResponder)
    {
        return registrationResponders.remove(theClass);
    }
    else
    {
        return registrationResponders.put(theClass, registrationResponder);
    }
}

/**
 * Sets the timeListener.
 * Unlike the other callback subsets, no dispatching occurs (there can be only one listener).
 * A null FederateAmbassadorTime is allowed.
 * @param timeListener the FederateAmbassadorTime to associate; if null, behaves as a remove()
 * @return the previous federationTimeListener (null if there was no previous mapping)
 */

public Object setTimeListener(
    FederateAmbassadorTime timeListener)
{
    if (null==timeListener)
    {
        return timeListeners.remove(null);
    }
    else
    {
        return timeListeners.put(null, timeListener);
    }
}
private FederateAmbassadorAttributeOwnership
getOwnershipListener(
    ObjectInstanceHandle theObject)
{
    Object listener = ownershipListeners.get(theObject);
    // Fall-through to Class level?
    try
    {
        if (null==listener) listener = ownershipListeners.get(_rtl.getKnownObjectClassHandle(theObject));
        // Fall-through to default handler?
        if (null==listener) listener = ownershipListeners.get(null);
        // Otherwise, give up
        if (null==listener) return null;
        return (FederateAmbassadorAttributeOwnership)listener;
    }
    //ObjectInstanceNotKnown, etc. can’t happen
    catch (RTIexception ignored) { return null; }
}

public void
requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeNotPublished,
        FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).requestAttributeOwnershipAssumption(theObject, offeredAttributes, userSuppliedTag);
    }
    catch (NullPointerException ignored) {}
    catch (NullPointerException ignored) {}
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNot Owned,
        AttributeDivestitureWasNotRequested,
        FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).requestDivestitureConfirmation(theObject, offeredAttributes);
    }
    catch (NullPointerException ignored) {}
}

public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        Attribute AcquisitionWasNotRequested,
        AttributeAlreadyOwned,
        AttributeNotPublished,
        FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).attributeOwnershipAcquisitionNotification(theObject, securedAttributes, userSuppliedTag);
    }
    catch (NullPointerException ignored) {}
}
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).attributeOwnershipUnavailable(theObject, theAttributes);
    } catch (NullPointerException ignored) {} 
}

public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).requestAttributeOwnershipRelease(theObject, candidateAttributes, userSuppliedTag);
    } catch (NullPointerException ignored) {} 
}

public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).confirmAttributeOwnershipAcquisitionCancellation(theObject, theAttributes);
    } catch (NullPointerException ignored) {} 
}
public void informAttributeOwnership(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute,
    FederateHandle theOwner)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).informAttributeOwnership(theObject,
            theAttribute, theOwner);
        catch (NullPointerException ignored) {}
    }
}

public void attributeIsNotOwned(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).attributeIsNotOwned(theObject,
            theAttribute);
        catch (NullPointerException ignored) {}
    }
}

public void attributeIsOwnedByRTI(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError
{
    try
    {
        getOwnershipListener(theObject).attributeIsOwnedByRTI(theObject,
            theAttribute);
        catch (NullPointerException ignored) {}
    }
}
private FederateAmbassadorAttributeScopeAdvisoryClient getAttributeScopeListener(
    ObjectInstanceHandle theObject)
{
    Object listener = attributeScopeListeners.get(theObject);
    // Fall-through to Class level?
    try
    {
        if (null == listener) listener =
            attributeScopeListeners.get(RTI.getKnownObjectClassHandle(theObject));
        // Fall-through to default handler?
        if (null == listener) listener =
            attributeScopeListeners.get(null);
        // Otherwise, give up
        if (null == listener) return null;
        return (FederateAmbassadorAttributeScopeAdvisoryClient) listener;
    }
    catch (RTIException ignored) { return null; }
}

public void attributesInScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    try
    {
        getAttributeScopeListener(theObject).attributesInScope(theObject, 
            theAttributes);
    }
    catch (NullPointerException ignored) {}
public void attributesOutOfScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    try {
        getAttributeScopeListener(theObject).attributesOutOfScope(theObject,
            theAttributes);
        catch (NullPointerException ignored) {} 
    }

    //FederateAmbassadorAttributeScopeAdvisoryServer part

    private FederateAmbassadorAttributeScopeAdvisoryServer
    getAttributeScopeResponder(
        ObjectInstanceHandle theObject)
    {
        Object listener = attributeScopeResponders.get(theObject);
        //Fall-through to class level?
        try {
            if (null==listener) listener =
                attributeScopeResponders.get(_rti.getKnownObjectClassHandle(theObject));
            //Fall-through to default handler?
            if (null==listener) listener =
                attributeScopeResponders.get(null);
            //Otherwise, give up
            if (null==listener) return null;
            return
                (FederateAmbassadorAttributeScopeAdvisoryServer) listener;
        }
        //ObjectInstanceNotKnown, etc. can’t happen
        catch (RTIexception ignored) { return null; }
    }
public void turnUpdatesOnForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    try
    {
        getAttributeScopeResponder(theObject).turnUpdatesOnForObjectInstance(t
    theObject, theAttributes);
        catch (NullPointerException ignored) {}
    }
}

public void turnUpdatesOffForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
 throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    try
    {
        getAttributeScopeResponder(theObject).turnUpdatesOffForObjectInstance(t
    theObject, theAttributes);
        catch (NullPointerException ignored) {}
    }
// FederateAmbassadorAttributeUpdateClient part

private FederateAmbassadorAttributeUpdateClient
getAttributeUpdateListener(
    ObjectInstanceHandle theObject)
{
    Object listener = attributeUpdateListeners.get(theObject);
    // Fall-through to Class level?
    try {
        if (null == listener) listener = attributeUpdateListeners.get(_rti.getKnownObjectClassHandle(theObject));
        // Fall-through to default handler?
        if (null == listener) listener = attributeUpdateListeners.get(null);
        // Otherwise, give up
        if (null == listener) return null;
        return (FederateAmbassadorAttributeUpdateClient)listener;
    }
    // ObjectInstanceNotKnown, etc. can't happen
    catch (RTIException ignored) { return null; }
}

public void
reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    try {
        getAttributeUpdateListener(theObject).reflectAttributeValues(theObject,
            theAttributes, userSuppliedTag, sentOrdering, theTransport);
    } catch (NullPointerException ignored) {}
public void reflectAttributeValue(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    try
    {
        getAttributeUpdateListener(theObject).reflectAttributeValue(theObject,
            theAttributes, userSuppliedTag, sentOrdering, theTransport,
            sentRegions);
        {}
    }
    }
}

public void reflectAttributeValue(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        FederateInternalError
{
    try
    {
        getAttributeUpdateListener(theObject).reflectAttributeValue(theObject,
            theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime,
            receivedOrdering);
        }
    }
    catch (NullPointerException ignored) {}
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValuePairMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotSubscribed,
                FederateInternalError
{
    try
    {
        getAttributeUpdateListener(theObject).reflectAttributeValues(theObject,
                                theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime,
                                receivedOrdering, sentRegions);
    }
    catch (NullPointerException ignored) {}}
}

public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValuePairMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
    throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotSubscribed,
                InvalidLogicalTime,
                FederateInternalError
{
    try
    {
        getAttributeUpdateListener(theObject).reflectAttributeValues(theObject,
                                theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime,
                                receivedOrdering, retractionHandle);
    }
    catch (NullPointerException ignored) {}}
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown,
AttributeNotRecognized,
AttributeNotSubscribed,
InvalidLogicalTime,
FederateInternalError
{
    try {
        getAttributeUpdateListener(theObject).reflectAttributeValues(theObject,
            theAttributes, userSuppliedTag, sentOrdering, theTransport, theTime,
            receivedOrdering, retractionHandle, sentRegions);
    } catch (NullPointerException ignored) {} 
}

//FederateAmbassadorAttributeUpdateServer part
private FederateAmbassadorAttributeUpdateServer
getAttributeUpdateResponder(
    ObjectInstanceHandle theObject)
{
    Object listener = attributeUpdateResponders.get(theObject);
    //Fall-through to Class level?
    try {
        if (null==listener) listener = attributeUpdateResponders.get(_rti.getKnownObjectClassHandle(theObject));
        //Fall-through to default handler?
        if (null==listener) listener = attributeUpdateResponders.get(null);
        //Otherwise, give up
        if (null==listener) return null;
        return (FederateAmbassadorAttributeUpdateServer)listener;
    } catch (RTIexception ignored) { return null; }
}
public void provideAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    try {
    getAttributeUpdateResponder(theObject) .provideAttributeValueUpdate(theObject, theAttributes, userSuppliedTag);
    } catch (NullPointerException ignored) {} //FederateAmbassadorObjectDiscovery part
}
private FederateAmbassadorObjectDiscovery getDiscoveryListener( 
    ObjectClassHandle theObjectClass)
{
    Object listener = discoveryListeners.get(theObjectClass); //Fall-through to default handler?
    if (null==listener) listener = discoveryListeners.get(null); //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorObjectDiscovery) listener;
}
public void discoverObjectInstance( 
    ObjectInstanceHandle theObject,
    ObjectClassHandle theObjectClass,
    String objectName)
throws CouldNotDiscover,
    ObjectClassNotRecognized,
    FederateInternalError
{
    try {
    getDiscoveryListener(theObjectClass).discoverObjectInstance(theObject, theObjectClass, objectName);
    } catch (NullPointerException ignored) {} //FederateAmbassadorObjectDiscovery part
}
private FederateAmbassadorObjectRemoval
getRemovalResponder(
    ObjectInstanceHandle theObject)
{
    Object listener = removalResponders.get(theObject);
    //Fall-through to class level?
    try
    {
        if (null==listener) listener = 
            removalResponders.get(_rti.getKnownObjectClassHandle(theObject));
        //Fall-through to default handler?
        if (null==listener) listener = removalResponders.get(null);
        //Otherwise, give up
        if (null==listener) return null;
        return (FederateAmbassadorObjectRemoval)listener;
    }
    //ObjectInstanceNotKnown, etc. can't happen
    catch (RTIexception ignored) { return null; }
}

public void
removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering)
throws ObjectInstanceNotKnown,
FederateInternalError
{
    try
    {
        getRemovalResponder(theObject).removeObjectInstance(theObject,
            userSuppliedTag, sentOrdering);
    }
    catch (NullPointerException ignored) {}
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
    InvalidLogicalTime,
    FederateInternalError
{
    try
    {
        getRemovalResponder(theObject) .removeObjectInstance(theObject,
            userSuppliedTag, sentOrdering, theTime, receivedOrdering,
            retractionHandle);
        }
    }
    catch (NullPointerException ignored) {} }

//FederateAmbassadorRestore part

private FederateAmbassadorRestore
getFederationRestoreListener()
{
    Object listener = federationRestoreListeners.get(null);
    //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorRestore)listener;
}

public void
requestFederationRestoreSucceeded(String label)
throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().requestFederationRestoreSucceeded(label
            );
    }
    catch (NullPointerException ignored) {} }
}
public void requestFederationRestoreFailed
    (String label)
    throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().requestFederationRestoreFailed(label);
    }
    catch (NullPointerException ignored) {}
}

public void federationRestoreBegun()
    throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().federationRestoreBegun();
    }
    catch (NullPointerException ignored) {}
}

public void initiateFederateRestore
    (String label, FederateHandle federateHandle)
    throws SpecifiedSaveLabelDoesNotExist, 
        CouldNotInitiateRestore, 
        FederateInternalError
{
    try
    {
        getFederationRestoreListener().initiateFederateRestore(label, federateHandle);
    }
    catch (NullPointerException ignored) {}
}

public void federationRestored()
    throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().federationRestored();
    }
    catch (NullPointerException ignored) {}
}
public void federationNotRestored(
    RestoreFailureReason reason)
throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().federationNotRestored(reason);
        throw (NullPointerException ignored)
    }
}

public void federationRestoreStatusResponse(
    FederateHandleRestoreStatusPair[] response)
throws FederateInternalError
{
    try
    {
        getFederationRestoreListener().federationRestoreStatusResponse(response);
        throw (NullPointerException ignored)
    }
}

//FederateAmbassadorSave part

private FederateAmbassadorSave getFederationSaveListener()
{
    Object listener = federationSaveListeners.get(null);
    //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorSave)listener;
}

public void initiateFederateSave(
    String label)
throws UnableToPerformSave,
    FederateInternalError
{
    try
    {
        getFederationSaveListener().initiateFederateSave(label);
    }
    catch (NullPointerException ignored) {}
}
public void initiateFederateSave(String label, LogicalTime time) throws InvalidLogicalTime, UnableToPerformSave, FederateInternalError {
    try {
        getFederationSaveListener().initiateFederateSave(label, time);
    } catch (NullPointerException ignored) { }
}

public void federationSaved() throws FederateInternalError {
    try {
        getFederationSaveListener().federationSaved();
    } catch (NullPointerException ignored) { }
}

public void federationNotSaved(SaveFailureReason reason) throws FederateInternalError {
    try {
        getFederationSaveListener().federationNotSaved(reason);
    } catch (NullPointerException ignored) { }
}

public void federationSaveStatusResponse(FederateHandleSaveStatusPair[] response) throws FederateInternalError {
    try {
        getFederationSaveListener().federationSaveStatusResponse(response);
    } catch (NullPointerException ignored) { }
}
// FederateAmbassadorSynchronization part

private FederateAmbassadorSynchronization getFederationSynchronizationListener(String synchronizationPointLabel) {
    Object listener = federationSynchronizationListeners.get(synchronizationPointLabel);
    // Fall-through to default handler?
    if (null == listener) listener = federationSynchronizationListeners.get(null);
    // Otherwise, give up
    if (null == listener) return null;
    return (FederateAmbassadorSynchronization) listener;
}

public void synchronizationPointRegistrationSucceeded(String synchronizationPointLabel) throws FederateInternalError {
    try {
        getFederationSynchronizationListener(synchronizationPointLabel).synchronizationPointRegistrationSucceeded(synchronizationPointLabel);
    }
    catch (NullPointerException ignored) {}
}

public void synchronizationPointRegistrationFailed(String synchronizationPointLabel, SynchronizationPointFailureReason reason) throws FederateInternalError {
    try {
        getFederationSynchronizationListener(synchronizationPointLabel).synchronizationPointRegistrationFailed(synchronizationPointLabel, reason);
    }
    catch (NullPointerException ignored) {}
}
public void announceSynchronizationPoint(String synchronizationPointLabel, byte[] userSuppliedTag)
    throws FederateInternalError
{
    try
    {
        getFederationSynchronizationListener(synchronizationPointLabel).announceSynchronizationPoint(synchronizationPointLabel, userSuppliedTag);
    }
    catch (NullPointerException ignored) {} //DR DC Valcartier TR 2007-412
}

public void federationSynchronized(String synchronizationPointLabel)
    throws FederateInternalError
{
    try
    {
        getFederationSynchronizationListener(synchronizationPointLabel).federationSynchronized(synchronizationPointLabel);
    }
    catch (NullPointerException ignored) {} //DR DC Valcartier TR 2007-412
}

private FederateAmbassadorInteractionAdvisory getInteractionAdvisoryResponder(InteractionClassHandle interactionClass)
{
    Object listener = interactionAdvisoryResponders.get(interactionClass);
    //Fall-through to default handler?
    if (null==listener) listener = interactionAdvisoryResponders.get(null);
    //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorInteractionAdvisory)listener;
}
public void turnInteractionsOn(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError
{
    try
    {
        getInteractionAdvisoryResponder(theHandle).turnInteractionsOn(theHandle);
        catch (NullPointerException ignored) {}
    }

    public void turnInteractionsOff(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError
    try
    {
        getInteractionAdvisoryResponder(theHandle).turnInteractionsOff(theHandle);
        catch (NullPointerException ignored) {}
    }

    //FederateAmbassadorInteractionOccurrence part
    private FederateAmbassadorInteractionOccurrence getInteractionListener(
    InteractionClassHandle interactionClass)
    {
        Object listener = interactionListeners.get(interactionClass);
        //Fall-through to default handler?
        if (null==listener) listener = interactionListeners.get(null);
        //Otherwise, give up
        if (null==listener) return null;
        return (FederateAmbassadorInteractionOccurrence)listener;
    }
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport);
        catch (NullPointerException ignored) {}
    }
}

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport, sentRegions);
        catch (NullPointerException ignored) {}
    }
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering
) throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport, theTime, receivedOrdering);
    }
    catch (NullPointerException ignored) {}
}

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions
) throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport, theTime, receivedOrdering, sentRegions);
    }
    catch (NullPointerException ignored) {}
}
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(
            interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport,
            theTime, receivedOrdering, messageRetractionHandle);
    }
    catch (NullPointerException ignored) {}
}

public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError
{
    try
    {
        getInteractionListener(interactionClass).receiveInteraction(
            interactionClass, theParameters, userSuppliedTag, sentOrdering, theTransport,
            theTime, receivedOrdering, messageRetractionHandle, sentRegions);
    }
    catch (NullPointerException ignored) {}
}
private FederateAmbassadorNameReservation
getNameReservationListener(
    String objectName)
{
    Object listener = nameReservationListeners.get(objectName);
    // Fall-through to default handler?
    if (null==listener) listener = nameReservationListeners.get(null);
    // Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorNameReservation)listener;
}

public void
objectInstanceNameReservationSucceeded(
    String objectName)
throws UnknownName,
    FederateInternalError
{
    try
    {
        getNameReservationListener(objectName).objectInstanceNameReservationSucceeded(objectName);
        
    } catch (NullPointerException ignored) {}
}

public void
objectInstanceNameReservationFailed(
    String objectName)
throws UnknownName,
    FederateInternalError
{
    try
    {
        getNameReservationListener(objectName).objectInstanceNameReservationFailed(objectName);
        
    } catch (NullPointerException ignored) {}
}
private FederateAmbassadorObjectRegistrationAdvisory getRegistrationResponder(
    ObjectClassHandle theClass)
{
    Object listener = registrationResponders.get(theClass);
    //Fall-through to default handler?
    if (null==listener) listener = registrationResponders.get(null);
    //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorObjectRegistrationAdvisory) listener;
}

public void startRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished, FederateInternalError
{
    try
    {
        getRegistrationResponder(theClass).startRegistrationForObjectClass(theClass);
        catch (NullPointerException ignored) {}
    }
}

public void stopRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished, FederateInternalError
{
    try
    {
        getRegistrationResponder(theClass).stopRegistrationForObjectClass(theClass);
        catch (NullPointerException ignored) {}
    }
}

private FederateAmbassadorTime getTimeListener()
{
    Object listener = timeListeners.get(null);
    //Otherwise, give up
    if (null==listener) return null;
    return (FederateAmbassadorTime) listener;
}
public void
timeRegulationEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
    NoRequestToEnableTimeRegulationWasPending,
    FederateInternalError
{
    try
    {
        getTimeListener().timeRegulationEnabled(time);
    }
    catch (NullPointerException ignored) {} 
}

public void
timeConstrainedEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
    NoRequestToEnableTimeConstrainedWasPending,
    FederateInternalError
{
    try
    {
        getTimeListener().timeConstrainedEnabled(time);
    }
    catch (NullPointerException ignored) {} 
}

public void
timeAdvanceGrant(
    LogicalTime theTime)
throws InvalidLogicalTime,
    JoinedFederateIsNotInTimeAdvancingState,
    FederateInternalError
{
    try
    {
        getTimeListener().timeAdvanceGrant(theTime);
    }
    catch (NullPointerException ignored) {} 
}

public void
requestRetraction(
    MessageRetractionHandle theHandle)
throws FederateInternalError
{
    try
    {
        getTimeListener().requestRetraction(theHandle);
    }
    catch (NullPointerException ignored) {} 
}

//end FedAmbWrapper
Annex E – The Java Chat Client

The Java Chat Client is a graphical user-interface (GUI) application. A single window is used, with various context-sensitive controls.

Figure 30. The Java Chat client in the disconnected state. The Log In button is enabled only when the text box beside is not empty. The remaining controls are disabled.

Figure 31. The Java Chat client in the joined state. The text box beside the Log Out button is disabled. The Send button is enabled only when the text box beside is not empty.
The `ChatRoomRegistryEntries` class implements the HLA ChatRoomRegistry’s list attribute object datatype (an `HLAvariableArrayType` of `ChatRoomRegistryEntry`).

```java
// File: ChatRoomRegistryEntries.java

package chat;

import hla.rti1516.CouldNotDecode;
import ca.gc.drdc_rddc.hla.rti1516.omt.*;
import java.util.Collection;

/**<p>A type-safe `HLAvariableArrayType` used by the ChatRoomRegistry object. *It uses `<code>ChatRoomRegistryEntry</code>` elements. *<code>@author</code> [mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault] ([@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier]) *<code>@version 1.0</code> */
public class ChatRoomRegistryEntries extends HLAobjectArray {
    /** Octet boundary of this class. */
    // Although not necessary, declaring this is more efficient
    public final static int octetBoundary = HLAinteger32BE.octetBoundary;
    // ChatRoomRegistryEntry.octetBoundary;
    // Since ChatRoomRegistryEntry consists of
    // - an HLAunicodeString (a variable array type, of octetBoundary 4 for the count and 2 for the HLAunicodeChar) and
    // - an HLAinteger16BE (octetBoundary 2)

    /** * Constructs an empty `<code>ChatRoomRegistryEntries</code>.
    */
    public ChatRoomRegistryEntries() {
        super();
    }
```
/**
 * Constructs <code>this</code> from the specified Collection.
 * An exception occurs if the Collection doesn't return a series of
 * element-compatible objects.
 * @param c a Collection specifying <code>this</code>' value
 */
public ChatRoomRegistryEntries(Collection c)
{
    super(c);
}

/**
 * Constructs <code>this</code> from the specified Object (a
 * single-element array).
 * @param o an element-compatible Object specifying
 * <code>this</code>' first value
 */
public ChatRoomRegistryEntries(Object o)
{
    super(o);
}

/**
 * Creates an <code>ChatRoomRegistryEntries</code> from the network
 * representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code>
 * representation of the <code>ChatRoomRegistryEntries</code>
 * @throws CouldNotDecode if the <code>byte[]</code> could not be decoded
 */
public ChatRoomRegistryEntries(byte[] buffer)
    throws CouldNotDecode
{
    super(buffer);
}
/**
 * Creates an <code>ChatRoomRegistryEntries</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>ChatRoomRegistryEntries</code>
 * @param offset where in the <code>buffer</code> the <code>ChatRoomRegistryEntries</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */

public ChatRoomRegistryEntries(byte[] buffer, int offset)
throws CouldNotDecode
{
    super(buffer, offset);
}

/**
 * Creates an <code>ChatRoomRegistryEntries</code> from the supplied <code>ByteWrapper</code>.
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>ChatRoomRegistryEntries</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */

public ChatRoomRegistryEntries(ByteWrapper byteWrapper)
throws CouldNotDecode
{
    super(byteWrapper);
}

//HLAarraydatatype interface implementation

/**
 * Returns the Class of the array's elements.
 * @return the Class of the array's elements
 */

public Class getElementClass()
{
    return ChatRoomRegistryEntry.class;
}

//end ChatRoomRegistryEntries
The ChatRoomRegistryEntry class implements the HLA ChatRoomRegistryEntry fixed record datatype.

// File: ChatRoomRegistryEntry.java
package chat;

import hla.rti1516.CouldNotDecode;
import ca.gc.drdc.rddl.hla.rti1516.ont.*;

/*
 * A type-safe HLAfixedRecordType used by the ChatRoomRegistryEntries type.
 * It consists of two fields, an HLAunicodeString name and an HLAinteger16BE slot.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca Daniel U. Thibault} ({@link http://www.valcartier.drdc-rddc.gc.ca DRDC Valcartier})
 * @version 1.0
 */
public class ChatRoomRegistryEntry
    extends HLAfixedRecordType
{
    // The first field's count; the HLAunicodeString elements (HLAunicodeChar)
    // and the HLAinteger16BE field have an octetBoundary of 2 each
    public static final int octetBoundary = HLAinteger32BE.octetBoundary;
    // The fields that make up our record:
    public HLAunicodeString name;
    public HLAinteger16BE slot;

    /**
     * Constructs a <code>ChatRoomRegistryEntry</code> of default values.
     */
    public ChatRoomRegistryEntry()
        throws CouldNotDecode
    {
        try
        {
            initializeFields();
        }
        catch (InstantiationException e)
        {
            CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
            throw (CouldNotDecode)cnd.initCause(e);
        }
    }
}
**

* Creates a `<code>ChatRoomRegistryEntry</code>` from the specified field values.
* @param aName an HLAunicodeString to copy into the `<code>name</code>` field
* @param aSlot an HLAinteger16BE to copy into the `<code>slot</code>` field
* @throws CouldNotDecode if something goes wrong
*/
public
ChatRoomRegistryEntry(HLAunicodeString aName,
HLAinteger16BE aSlot)
throws CouldNotDecode
{
    this();
    name.setValue(aName.getValue());
    slot.setValue(aSlot.getValue());
}

/**
 * Creates a `<code>ChatRoomRegistryEntry</code>` from the specified field values.
 * @param aName a String to copy into the `<code>name</code>` field
 * @param aSlot an HLAinteger16BE to copy into the `<code>slot</code>` field
 * @throws CouldNotDecode if something goes wrong
 */
public
ChatRoomRegistryEntry(String aName,
HLAinteger16BE aSlot)
throws CouldNotDecode
{
    this();
    name.setValue(aName);
    slot.setValue(aSlot.getValue());
}

/**
 * Creates a `<code>ChatRoomRegistryEntry</code>` from the network representation in the provided `<code>byte[]</code>`.
 * @param buffer the network-provided `<code>byte[]</code>` representation of the `<code>ChatRoomRegistryEntry</code>`
 * @throws CouldNotDecode if the `<code>buffer</code>` could not be decoded
 */
public
ChatRoomRegistryEntry(byte[] buffer)
throws CouldNotDecode
{
    this(buffer, 0);
}
/**
 * Creates a <code>ChatRoomRegistryEntry</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.  
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>ChatRoomRegistryEntry</code>  
 * @param offset where in the <code>buffer</code> the <code>ChatRoomRegistryEntry</code> representation begins  
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded  
 */
public ChatRoomRegistryEntry(byte[] buffer, int offset) throws CouldNotDecode
    this(new ByteWrapper(buffer, offset));
}

/**
 * Creates a <code>ChatRoomRegistryEntry</code> from the supplied <code>ByteWrapper</code>.  
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>ChatRoomRegistryEntry</code> begins  
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded  
 */
public ChatRoomRegistryEntry(ByteWrapper byteWrapper) throws CouldNotDecode
    this();
    decode(byteWrapper);
/**
 * Creates a <code>ChatRoomRegistryEntry</code> from the provided <code>Map</code>. 
 * The <code>Map</code> is specified as mapping String keys (representing the field names) to their values <code>HLAdatatype</code> implementations). 
 * The map <code>size()</code> will be unchanging, since the record will always have the same fields in the same order. 
 * Values may not be <code>null</code> since they all must be HLA data types (i.e. all references will exist). 
 * @param theMap the <code>Map</code> representation of the <code>ChatRoomRegistryEntry</code> 
 * @throws CouldNotDecode if the <code>Map</code> could not be decoded 
 */

public ChatRoomRegistryEntry(java.util.Map theMap) throws CouldNotDecode {
    this();
    try {
        putAll(theMap);
    } //Wrap all exceptions as CouldNotDecode 
    catch (Exception e) {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}

//end ChatRoomRegistryEntry
The Participant class implements the HLA Participant fixed record datatype.

// File: Participant.java

package chat;

import hla.rti1516.CouldNotDecode;
import ca.gc.drdc.rddc.hla.rti1516.ont.*;

/**
 * A type-safe HLAfixedRecordType used to represent a Participant object.
 * It consists of three fields, an HLAboolean logged_in, an HLAinteger32BE user_handle
 * and an HLAinteger16BE chat_room_slot.
 * @author {mailto:Daniel.Thibault@DRDC-RDDC.gc.ca} Daniel U. Thibault
 * @version 1.0
 */

public class Participant extends HLAfixedRecordType {
    public static final int octetBoundary = HLAinteger32BE.octetBoundary; //4

    // The fields that make up our record:
    public HLAboolean logged_in;
    public HLAinteger32BE user_handle;
    public HLAinteger16BE chat_room_slot;

    /**
     * Constructs a <code>Participant</code> of default values.
     */
    public Participant() throws CouldNotDecode {
        try {
            initializeFields();
        } catch (InstantiationException e) {
            CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
            throw (CouldNotDecode)cnd.initCause(e);
        }
    }
}
/**
 * Creates a <code>Participant</code> from the specified field values.
 * @param aLoggedIn an HLAboolean to copy into the <code>logged in</code> field
 * @param aUserHandle an HLAIinteger16BE to copy into the <code>user handle</code> field
 * @param aChatRoomSlot an HLAIinteger16BE to copy into the <code>chat room slot</code> field
 * @throws CouldNotDecode if somethings goes wrong
 */
public Participant (HLAboolean aLoggedIn, HLAIinteger32BE aUserHandle, HLAIinteger16BE aChatRoomSlot) throws CouldNotDecode {
    this();
    logged_in.setValue(aLoggedIn.getValue());
    user_handle.setValue(aUserHandle.getValue());
    chat_room_slot.setValue(aChatRoomSlot.getValue());
}

/**
 * Creates a <code>Participant</code> from the network representation in the provided <code>byte[]</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>Participant</code>
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public Participant(byte[] buffer) throws CouldNotDecode {
    this(buffer, 0);
}

/**
 * Creates a <code>Participant</code> from the network representation in the provided <code>byte[]</code> at the indicated <code>offset</code>.
 * @param buffer the network-provided <code>byte[]</code> representation of the <code>Participant</code>
 * @param offset where in the <code>buffer</code> the <code>Participant</code> representation begins
 * @throws CouldNotDecode if the <code>buffer</code> could not be decoded
 */
public Participant(byte[] buffer, int offset) throws CouldNotDecode {
    this(new ByteWrapper(buffer, offset));
}
/**
 * Creates a <code>Participant</code> from the supplied <code>ByteWrapper</code>. 
 * @param byteWrapper the <code>ByteWrapper</code> at whose current <code>pos()</code> the representation of the <code>Participant</code> begins
 * @throws CouldNotDecode if the <code>ByteWrapper</code> could not be decoded
 */
public Participant(ByteWrapper byteWrapper)
    throws CouldNotDecode
{
    this();
    decode(byteWrapper);
}

/**
 * Creates a <code>Participant</code> from the provided <code>Map</code>. 
 * The <code>Map</code> is specified as mapping String keys (representing the field names) to their values <code>HLA datatype</code> implementations). 
 * The map <code>size()</code> will be unchanging, since the record will always have the same fields in the same order. 
 * Values may not be <code>null</code> since they all must be HLA data types (i.e. all references will exist). 
 * @param theMap the <code>Map</code> representation of the <code>Participant</code>
 * @throws CouldNotDecode if the <code>Map</code> could not be decoded
 */
public Participant(java.util.Map theMap)
    throws CouldNotDecode
{
    this();
    try
    {
        putAll(theMap);
    }
    //Wrap all exceptions as CouldNotDecode
    catch (Exception e)
    {
        CouldNotDecode cnd = new CouldNotDecode(e.getMessage());
        throw (CouldNotDecode)cnd.initCause(e);
    }
}
//end Participant
The `MyChat` class implements the Java HLA Chat Client. This is a large class in part because it contains a multitude of inner classes. It was mostly developed using the (free) NetBeans 3 integrated development environment (IDE), which makes navigating the code in object view fairly easy and efficient. The comments which NetBeans inserts to recognise its own protected (user read-only) blocks have been removed in order to make the source more legible.

```
// File: MyChat.java

package chat;

import java.io.File;
import java.util.HashMap;
//Needed to iterate through the Interaction key set:
import java.util.Iterator;
//Event-handling thread utilities
import javax.swing.SwingUtilities;
//HLA interfaces and normative classes
import hla.rti1516.*;
//The RTI implemented class
import se.pitch.pti1516.RTI;
//Our encoder facilities
import ca.gc.drdc_rddc.hla.rti1516.omt.*;
//Our FedAmbWrapper
import ca.gc.drdc_rddc.hla.rti1516.FedAmb.*;

/**
 * A Chat client that uses the HLA IEEE 1516 RTI as its backbone.
 * Demonstrates Interactions, Objects, and Data Distribution Management.
 * <p>
 * Although the ConcurrentAccess exception from 1.3 is no more,
 * IEEE 1516 will throw "RTIInternalError: Concurrent access
 * attempted to <method name>" if the RTI ambassador is invoked
 * from the Federate service thread -- but only for certain
 * services (unlike 1.3).
 * getObjectName and
 * getAttributeHandleValueMapFactory won't throw the exception,
 * for example.
 * <p>
 * The Chat client joins the federation right away, sets up
 * listeners and responders, and publishes and subscribes to
 * objects and interactions.
 * <p>
 * At first, it was thought to have the federate refrain from
 * subscribing until it logs on, but it became clear that, far
 * from diminishing message traffic, the added complexity of
 * managing known/unknown objects, up-to-date and out-dated
 * reflections, ownership negotiations and so on could in fact
 * increase traffic under certain conditions, and certainly made
 * the multi-threaded program hellishly difficult to design and
 * debug.
 * <p>
```
* The federation has one unique object, named *ChatRoomRegistry*, of the ChatRoomRegistry class. This object has one attribute, "list", which is an HLAVariableArray of ChatRoomRegistryEntry objects, themselves of type HLAFixedRecord. These have two fields, "name" (HLAUnicodeString) and "slot" (HLAInteger16BE). The ChatRoomRegistry's purpose is to assign a unique slot to each ChatRoom object (see below) when it is created. The list keeps track of all extant ChatRooms, and can be used by a ChatRoom creator to ensure the uniqueness of the slot.

* In circumstances where certain operations are preferred to be performed by only one federate at a time, the ChatRoomRegistry could be used as a "bâton", ownership being passed between federates as a token of their right in performing the operation in question. This was not used here.

* The ChatRoom objects are created and deleted as needed by the federates. They have two attributes, "slot" (HLAInteger16BE) and "name" (HLAUnicodeString). There are three special ChatRooms:

  * `<li>`The "nowhere" ChatRoom is never instantiated and has reserved slot 0. No federate ever subscribes to this region. It is used (through AssociateRegionForUpdates) to temporarily withdraw a Participant object from visibility and thus force advisories to trigger. Participants are constantly associated with the nowhere region. When a Participant switches ChatRooms, it is first unassociated from its current ChatRoom then re-associated with the new one. In between, it remains associated with the nowhere region just to prevent it from reverting to the default region. Note that I'd rather have used slot -1, but dimensions are specified as running from 0 upward, and I did not want to use normalization...

  * `<li>`The "_waiting_room" is slot 1 and is never deleted. This is where Participants are parked whilst logged out. All federates constantly subscribe to the "_waiting_room".

  * `<li>`The "_<General>_" ChatRoom, finally, has slot 2. It is the ChatRoom every Participant initially logs into.

When a Participant switches ChatRooms or logs out, if the current list of other Participants in the ChatRoom (filtered using the slot) is empty, then the ChatRoom must be deleted. We could delay the inevitable by transferring ownership to some other Participant who happens to be elsewhere; nevertheless, since there'll eventually be a time when there is only one Participant logged into the federation, we might as well delete ChatRooms at that time. This means that the <General> ChatRoom may be deleted and recreated repeatedly (imagine the lone Participant keeps switching ChatRooms). For this reason, we do *not* use the HLA name reservation service with ChatRooms (except for the "_waiting_room", which is a fixture); we'll rely on the ChatRoomRegistry for this.
User-created ChatRooms have names prefixed with "c", so a user could create another "<General>" ("c<General>").

Participant objects, finally, have three attributes:
- "logged_in" (HLAboolean), "user_handle" (HLAinteger32BE) and "chat_room_slot" (HLAinteger16BE). Logged_in is redundant, since a user is logged-in if his Participant token is in a chat_room_slot other than 0 or 1 (a federate may own several Participant objects, but only one of them can be elsewhere than slots 0 or 1). We use the name reservation service to ensure the uniqueness of user names. The user_handles are associated with a dimension for DDM purposes, allowing private messages between users.

When a user requests to log in as a certain name, we attempt reservation. If it succeeds, the name is new and the correspondingly-named Participant object is created. If it fails, the Participant object is either already in use (in which case the log-in fails) or is pre-extant but unused (in which case we need only acquire it to complete the log-in). The Participant objects cannot be deleted once created -- the name reservation mechanism entails that deletion is irrevocable.

Therefore, ownership of logged-out Participant objects is transferred between federates as needed. A federate that shuts down must divest any Participants it owns (which will perforce all be in the "_waiting_room"). The last federate to leave the federation deletes everything.

DDM is used in two ways. The basic way uses the ChatRoomSlots dimension, and tracks which Participants are in the same ChatRoom as each federate. Normal messaging is within the ChatRoom only. The other way is using the UserHandle dimension, and allows private messaging between two users. Because the interface only shows those Participants in the same ChatRoom, private messaging is thus a sub-case, but in theory you could send private messages to anyone else in the federation.

A federate is in one of three states: logged out, logged in or logging in (a transitional state). This is represented by the booleans _me_logged_in and _me_logging_in (F, F = logged out; T, F = logged in; F, T = logging in; T, T = not possible).

@author Daniel U. Thibault <Daniel.Thibault@DRDC-RDDC.GC.Ca>
public class MyChat
extends javax.swing.JFrame
implements java.awt.event.ActionListener,
javax.swing.event.DocumentListener,
java.awt.event.WindowListener,
java.awt.event.ItemListener{

private static final int CRC_PORT = 8989;
private static final String _name_ChatRoomRegistry = "ChatRoomRegistry";
private static final String _name_nowhere_ChatRoom = "nowhere";
private static final String _name_waiting_room_ChatRoom = "waiting_room";
private static final String _name_general_ChatRoom = "<General>";
private static final short _slot_nowhere_ChatRoom = 0;
private static final short _slot_waiting_room_ChatRoom = 1;
private static final short _slot_general_ChatRoom = 2;
private static final short _slot_FirstFreeChatRoomSlot = 3;

private String _args[];
private String rtiHost = "localhost"; //args[0]
private String fdd = "D:\Documents and Settings\dthibault\Mes Documents\Java Projects\MyChat.xml"; //args[1]
private String federate = "MyChatRoom"; //args[2]
private String fedname = "MyChatter"; //args[3]
private FederateHandle _federateHandle;
private RTIambassador _rtiAmbassador;
private FedAmbWrapper _fedAmbassador;

private InteractionClassHandle _ich_Communication;
private ParameterHandle _iph_Communication_message;
private ParameterHandle _iph_Communication_sender;

private ObjectClassHandle _och_ChatRoomRegistry;
private AttributeHandleSet _oahs_ChatRoomRegistry;
private AttributeHandleSet _oahs_ChatRoomRegistry_forUpdate;
private AttributeHandle _oah_ChatRoomRegistry_DeletePrivilege;
private AttributeHandle _oah_ChatRoomRegistry_list;
private AttributeHandleValueMap _oahvm_ChatRoomRegistry;

private ObjectClassHandle _och_ChatRoom;
private AttributeHandleSet _oahs_ChatRoom;
private AttributeHandleSet _oahs_ChatRoom_forUpdate;
private AttributeHandle _oah_ChatRoom_DeletePrivilege;
private AttributeHandle _oah_ChatRoom_slot;
private AttributeHandle _oah_ChatRoom_name;
private ObjectClassHandle _och_Participant;
private AttributeHandleSet _oabs_Participant;
private AttributeHandleSet _oabs_Participant_forUpdate;
private AttributeHandle _oah_Participant_DeletePrivilege;
private AttributeHandle _oah_Participant_logged_in;
private AttributeHandle _oah_Participant_user_handle;
private AttributeHandle _oah_Participant_chat_room_slot;

private DimensionHandle _dh_UserHandleSlots;
private DimensionHandleSet _dhs_UserHandleSlotsSet;
private DimensionHandle _dh_ChatRoomSlots;
private DimensionHandleSet _dhs_ChatRoomSlotsSet;

// The handle of the Region matched to "nowhere" (slot 0)
private RegionHandle _rh_nowhere_ChatRoom;
private RegionHandleSet _rhs_nowhere_ChatRoom;

// The handle of the Region matched to the "waiting_room" ChatRoom (slot 1)
private RegionHandle _rh_waiting_room_ChatRoom;
private RegionHandleSet _rhs_waiting_room_ChatRoom;

// The handle of the Region matched to the "<General>" ChatRoom (slot 2)
private RegionHandle _rh_general_ChatRoom;
private RegionHandleSet _rhs_general_ChatRoom;

// The handle of the Region matched to our current ChatRoom, if any
private RegionHandle _rh_current_ChatRoom;
private RegionHandleSet _rhs_current_ChatRoom;

// The list (size 2) of AttributeSet-RegionSet pairs used for Participant publish/subscribe
private AttributeSetRegionSetPairList _asrspl_Participant_nowhere;
private AttributeSetRegionSetPairList _asrspl_Participant_waiting_room;

// The AttributeSet will always be _oabs_Participant_forUpdate;
// the RegionSet will always be one of _rh_nowhere_ChatRoom,
// _rh_waiting_room_ChatRoom or _rh_current_ChatRoom

private Semaphore _sem_reservation = new Semaphore();
private Semaphore _sem_discovery = new Semaphore();
private Semaphore _sem_acquisition = new Semaphore();
private Semaphore _sem_divestiture = new Semaphore();

private boolean _me_logged_in = false;
private boolean _me_logging_in = false;
private boolean _alone = true;
private boolean _me_shutting_down = false;

// To allow user names and chat room names to be anything, the latter will be prefixed
// by "p" and "c", respectively, whilst our reserved names will be prefixed by "_"
private class aChatRoomRegistry
{
    // Whether we own the object or not
    public boolean owned = false;
    // Whether we subscribe to the object class or not
    // (and therefore whether the value is up to date or not)
    public boolean subscribed = false;
    // Whether we are divesting the object or not (makes sense only if owned)
    public boolean divesting = false;
    // The ChatRoomRegistry object's name
    final public String name = _name_ChatRoomRegistry;
    // The ChatRoomRegistry object's handle
    public ObjectInstanceHandle handle = null;
    // The ChatRoomRegistry object's list field
    public ChatRoomRegistryEntries list;

    /**
     * Default constructor; the name is unique and the list pre-loaded with the "waiting_room" and <General> ChatRooms.
     */
    public aChatRoomRegistry()
        throws CouldNotDecode
    {
        list = new ChatRoomRegistryEntries();
        list.add(list.size(), new ChatRoomRegistryEntry(new HLAUnicodeString(_name_waiting_room_ChatRoom), new HLAInteger16BE(_slot_waiting_room_ChatRoom)));
        list.add(list.size(), new ChatRoomRegistryEntry(new HLAUnicodeString(_name_general_ChatRoom), new HLAInteger16BE(_slot_general_ChatRoom)));
    }

    private aChatRoomRegistry _ChatRoomRegistry = null;
/**
 * Constructs a chat room of the specified name and slot.
 * @param aName a String specifying the ChatRoom's name field
 * @param aSlot a Short specifying the ChatRoom's slot field
 */

public aChatRoom(String aName, short aSlot) {
    slot = new HLAinteger16BE(aSlot);
    name = new HLAunicodeString(aName);
}

//The nowhere ChatRoom is never instantiated
//The waiting room ChatRoom
final private aChatRoom _waiting_room_ChatRoom = new aChatRoom(_name_waiting_room_ChatRoom, _slot_waiting_room_ChatRoom);

//The general ChatRoom
private aChatRoom _general_ChatRoom; // = new aChatRoom(_name_general_ChatRoom, _slot_general_ChatRoom);

//The ChatRoom we're in
private aChatRoom _myChatRoom;

//The set of ChatRooms (keys are ObjectInstanceHandle, values are aChatRoom)
private HashMap _theChatRooms = new HashMap();
// Our Participant object
private class Participant
{
    // Whether we own the object or not
    public boolean owned = false;
    // Whether we subscribe to the object class or not
    // (and therefore whether the value is up to date or not)
    public boolean subscribed = false;
    // Whether we are divesting the object or not (makes sense only if owned)
    public boolean divesting = false;
    // Whether the object is in scope or not (relevant only if owned is false)
    // (we're forced to do this instead of localDelete)
    public boolean inscope = false;
    // The object's handle
    public ObjectInstanceHandle handle = null;
    // The object's name
    public HLAunicodeString name = new HLAunicodeString();
    // The logged-in field
    private HLAboolean logged_in;
    // The user handle field
    private HLAinteger32BE user_handle;
    // The current chat room slot field
    private HLAinteger16BE chat_room_slot;

    /**
     * Default constructor; the name is empty, logged_in is HLAfalse,
     * user_handle is 0 and chat_room_slot is 0 (nowhere).
     */
    public Participant()
    {
        logged_in = new HLAboolean(false);
        user_handle = new HLAinteger32BE(0);
        chat_room_slot = new HLAinteger16BE(_slot_nowhere.ChatRoom);
    }

    private aParticipant _me = new aParticipant();
    // List of known Participants (including ourselves); keys are ObjectInstanceHandles, values are aParticipant objects
    private HashMap _theParticipants = new HashMap();
/** Creates new form MyChat */
public MyChat(String args[])
{
    // This runs in the main thread ("main")
    _args = args;
    initComponents();
    txtUsername.getDocument().addDocumentListener(this);
    txtMessage.getDocument().addDocumentListener(this);
}

/**
* This method is called from within the constructor to 
* initialize the form.
* WARNING: Do NOT modify this code. The content of this method is 
* always regenerated by the Form Editor.
*/
private void initComponents()
{
    jPanel1 = new javax.swing.JPanel();
    jPanel3 = new javax.swing.JPanel();
    btnLogon = new javax.swing.JButton();
    txtUsername = new javax.swing.JTextField();
    jPanel4 = new javax.swing.JPanel();
    btnSendMessage = new javax.swing.JButton();
    txtMessage = new javax.swing.JTextField();
    jPanel5 = new javax.swing.JPanel();
    jPanel6 = new javax.swing.JPanel();
    lblChatRoom = new javax.swing.JLabel();
    lstChatRooms = new javax.swing.JComboBox();
    btnNewChatRoom = new javax.swing.JButton();
    jPanel7 = new javax.swing.JPanel();
    lblSendTo = new javax.swing.JLabel();
    lstSendTo = new javax.swing.JComboBox();
    txtArea = new javax.swing.JTextArea();
    jPanel2 = new javax.swing.JPanel();
    lblStatus = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("MyChat");
    addWindowListener(this);

    jPanel1.setLayout(new java.awt.BorderLayout());
    jPanel3.setLayout(new java.awt.BorderLayout());
    btnLogon.setText("Log In");
    btnLogon.setEnabled(false);
    btnLogon.addActionListener(this);
    btnLogon.addWindowListener(this);
    jPanel3.add(btnLogon, java.awt.BorderLayout.WEST);
    txtUsername.setEnabled(false)
    jPanel3.add(txtUsername, java.awt.BorderLayout.CENTER);
    jPanel1.add(jPanel3, java.awt.BorderLayout.NORTH);
jPanel4.setLayout(new java.awt.BorderLayout());
btnSendMessage.setText("Send");
btnSendMessage.setEnabled(false);
btnSendMessage.addActionListener(this);

jPanel4.add(btnSendMessage, java.awt.BorderLayout.WEST);
txtMessage.setEnabled(false);

jPanel4.add(txtMessage, java.awt.BorderLayout.CENTER);

jPanel1.add(jPanel4, java.awt.BorderLayout.CENTER);

jPanel5.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.LEFT));


lblChatRoom.setText("Chat Room:");
lblChatRoom.setEnabled(false);

jPanel6.add(lblChatRoom);

lstChatRooms.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "<General>" }));

lstChatRooms.setEnabled(false);
lstChatRooms.addItemListener(this);

jPanel6.add(lstChatRooms);

btnNewChatRoom.setText("New");
btnNewChatRoom.setActionCommand("NewChatRoom");
btnNewChatRoom.setEnabled(false);
btnNewChatRoom.addActionListener(this);

jPanel6.add(btnNewChatRoom);

jPanel5.add(jPanel6);


lblSendTo.setText("Send To:");
lblSendTo.setEnabled(false);

jPanel7.add(lblSendTo);

lstSendTo.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "<All>" }));

lstSendTo.setEnabled(false);

jPanel7.add(lstSendTo);

jPanel5.add(jPanel7);

jPanel1.add(jPanel5, java.awt.BorderLayout.SOUTH);

getContentPane().add(jPanel1, java.awt.BorderLayout.NORTH);
Code for dispatching events from components to event handlers.

```java
public void actionPerformed(java.awt.event.ActionEvent evt)
{
    if (evt.getSource() == btnLogon) {
        MyChat.this.btnLogonActionPerformed(evt);
    } else if (evt.getSource() == btnSendMessage) {
        MyChat.this.btnSendMessageActionPerformed(evt);
    } else if (evt.getSource() == btnNewChatRoom) {
        MyChat.this.btnNewChatRoomActionPerformed(evt);
    }
}

public void itemStateChanged(java.awt.event.ItemEvent evt)
{
    if (evt.getSource() == lstChatRooms) {
        MyChat.this.lstChatRoomsItemStateChanged(evt);
    }
}

public void windowActivated(java.awt.event.WindowEvent evt)
{
}

public void windowClosed(java.awt.event.WindowEvent evt)
{
}

public void windowClosing(java.awt.event.WindowEvent evt)
{
    if (evt.getSource() == MyChat.this) {
        MyChat.this.exitForm(evt);
    }
}
public void windowDeactivated(java.awt.event.WindowEvent evt) {
}

public void windowDeiconified(java.awt.event.WindowEvent evt) {
}

public void windowIconified(java.awt.event.WindowEvent evt) {
}

public void windowOpened(java.awt.event.WindowEvent evt) {
    if (evt.getSource() == MyChat.this) {
        MyChat.this.openForm(evt);
    }
}

private void btnNewChatRoomActionPerformed(java.awt.event.ActionEvent evt) {
    // To allow user names and chat room names to be anything, the latter will be prefixed
    // by "p" and "c", respectively, whilst our reserved names will be prefixed by "_
    String newChatRoomName = javax.swing.JOptionPane.showInputDialog("Enter the new chat room's
    name: ");
    if (newChatRoomName.equals("")) return;
    synchronized (AcquireChatRoomRegistry()) {
        lstChatRooms.addItem(newChatRoomName);
    }
    lstChatRooms.setSelectedItem(newChatRoomName);
}
private void lstChatRoomsItemStateChanged(java.awt.event.ItemEvent evt) {
    //getItemSelectable //the originator of the event, a
    javax.swing.JComboBox
    //getItem //the item affected by the event, a java.lang.String
    (or whatever was added to the list)
    if (! me.logged_in || (evt.getStateChange() != evt.SELECTED))
        return;
    //Moving to the evt.getItem().toString() Chat Room
    try {
        String newChatRoomName = evt.getItem().toString(); // General ChatRoom
        if ( ((javax.swing.JComboBox) evt.getItemSelectable() ) .getSelectedIndex() == 0) {
            newChatRoomName = _name_general_chatroom; // "* +
            evt.getItem().toString(); //General ChatRoom
        } else {
            newChatRoomName = "c" + evt.getItem().toString();
        } //if
        //The change of ChatRooms cannot be from nowhere or
        waiting_room; it has
        //to be from an active ChatRoom, including the <General>
        ChatRoom.
        _rtiAmbassador.unassociateRegionsForUpdates( me.handle,
        _asrsp1Participant_current);
        //At this point, _me.handle is only associated with
        _asrsp1Participant_nowhere
        //Unsubscribe from the current ChatRoom
        _rtiAmbassador.unsubscribeInteractionClassWithRegions(_ich_communicati
        on, _rhs_current_Chattroom);
        //Look at the ChatRoom we're leaving to decide whether to
        delete it or not
        int count = 0;
        synchronized(_theParticipants) {
            aParticipant _participant;
            for (Iterator i = _theParticipants.values().iterator();
            i.hasNext();)
                iParticipant = (aParticipant)i.next();
                //Out of scope Participants have unreliable attributes;
            they cannot be in our ChatRoom in any case
            //The only exceptions are owned Participants: any owned
            orphans can only be in the waiting_room
            //me isn't counted
            if (! _me.handle.equals(_iParticipant.handle)) &&
                _iParticipant.inscope) &&
                (_me.chat_room_slot.equals(_iParticipant.chat_room_slot)) ) count++;
                //for
            } //synchronized(_theParticipants)
//This'll cause the Participants in our ChatRoom to go out of scope

_rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(_och_Participant, _asrsp1_Participant_current);

//What if someone logs into the ChatRoom after we've counted its Participants but before we actually delete it?

aChatRoom _previousChatRoom = _myChatRoom;
if (_newChatRoomName.equals(_name_general_ChatRoom))
{
    //Does the General ChatRoom exist?
    _general_ChatRoom = SeekChatRoomBySlot(_slot_general_ChatRoom);
    //General ChatRoom may not be in _theChatRooms, since it is never explicitly recreated
    if (null == _general_ChatRoom)
    {
        synchronized(AcquireChatRoomRegistry())
        {
            _general_ChatRoom = new aChatRoom(_name_general_ChatRoom, _slot_general_ChatRoom);
            _general_ChatRoom.owned = _general_ChatRoom.subscriptions = true;
            //
            _general_ChatRoom.divesting = false;
            synchronized(_theChatRooms)
            {
                _theChatRooms.put(_general_ChatRoom.handle = _rtiAmbassador.registerObjectInstance(_och_ChatRoom),
                                         _general_ChatRoom);
                //No need to add the General ChatRoom to the _ChatRoomRegistry, as it is already listed
                } //synchronized(_theChatRooms)
        } //synchronized(AcquireChatRoomRegistry())
    } //if
    _myChatRoom = _general_ChatRoom;
} else {
    _myChatRoom = SeekChatRoomByName(_newChatRoomName);
    //Cannot fail
} //if
// Modify asrspl and delete old region, unless it was the General ChatRoom
RegionHandle _rh_previous(ChatRoom = _rh_current(ChatRoom);
// = (RegionHandle) rhs_current(ChatRoom.toArray()[0];
    _asrspl_Participant(current).remove(0);
    _rhs_current(ChatRoom).remove(_rh_previous(ChatRoom);
    if (!_rh_previous(ChatRoom).equals(_rh_general(ChatRoom))
    |
        _rtiAmbassador.deleteRegion(_rh_previous(ChatRoom);
    } // if
    // New region
    if (_newChatRoomName.equals(_name_general(ChatRoom))
        _rh_current(ChatRoom) = _rh_general(ChatRoom;
    else
        _rh_current(ChatRoom) =
            _rtiAmbassador.createRegion(_dhs(ChatRoomSlotsSet);
            _rtiAmbassador.setRangeBounds(_rh_current(ChatRoom,
                _dch(ChatRoomSlots), new RangeBounds(_myChatRoom.slot.getValue(), 1 +
                    _myChatRoom.slot.getValue()));
            } // if
    } // else
    _rhs_current(ChatRoom).add(_rh_current(ChatRoom);
    _rtiAmbassador.commitRegionModifications(_rhs_current(ChatRoom);
    // Won't complain if there are no mods to commit
    // Regenerate asrspl
    _asrspl_Participant(current).add(new
        AttributeRegionAssociation(_oahs_Participant_forUpdate,
            _rhs_current(ChatRoom));
        _me.chat_room_slot.setValue(_myChatRoom.slot.getValue());
        // Go back in.
        // This'll reveal the participants in the new ChatRoom
        _rtiAmbassador.subscribeObjectClassAttributesWithRegions(_och_Particip
            _ant, _asrspl_Participant_current);
            // Re-open the communication channel for the new ChatRoom
            _rtiAmbassador.subscribeInteractionClassWithRegions(_ich_Communication
                , _rhs_current(ChatRoom);
                // Reveal ourselves to the new ChatRoom
                _rtiAmbassador.associateRegionsForUpdates(_me.handle,
                    _asrspl_Participant_current);
//Delete old ChatRoom?
if (count <= 0) //Should be just ==0 but you never know
(because of inscope, _me isn't counted)
{
    //We were the only Participant left in the old ChatRoom;
delete it as we leave (we must be the owner, obviously)
synchronized(AcquireChatRoomRegistry())
{
    //Because we're the owner, we won't get a
RemoveObjectInstance notification
//The remove returns the removed value, which is
_previousChatRoom
synchronized(_theChatRooms)
{
    _rtiAmbassador.deleteObjectInstance(((aChatRoom)_theChatRooms.remove(_
previousChatRoom.handle)).handle, null);
} //synchronized(_theChatRooms)
RemoveChatRoomAndUpdateRegistry(_previousChatRoom.slot.getValue());
if ( !_previousChatRoom.name.toString().equals(_name_general_chatRoom))
{
    lstChatRooms.removeItem( _previousChatRoom.name.toString().substring(1) 
});
} //if
else if ( _previousChatRoom.owned)
{
    //There are other Participants but we were the owner:
transfer ownership
_previousChatRoom.divesting = true;

    _rtiAmbassador.negotiatedAttributeOwnershipDivestiture(_previousChatRo
om.handle, _oahs_chatRoom, null);
} //if
} catch (Exception e) {
    e.printStackTrace();
} finally {
    //try
}

//Start of DocumentListener implementation and dispatching code
/**
 * The NetBeans IDE unfortunately does not expose/integrate this
critical bit,
 * so we have to hook it up manually.
 */
public void changedUpdate(javax.swing.event.DocumentEvent evt)
{
public void insertUpdate(javax.swing.event.DocumentEvent evt) {
    //This runs in the event-handling thread ("AWT-EventQueue-0")
    if (evt.getDocument().equals(txtUsername.getDocument()))
    {
        MyChat.this.txtUsernameDocumentChanged(evt);
    } else if (evt.getDocument().equals(txtMessage.getDocument()))
    {
        MyChat.this.txtMessageDocumentChanged(evt);
    }
}

public void removeUpdate(javax.swing.event.DocumentEvent evt) {
    if (evt.getDocument().equals(txtUsername.getDocument()))
    {
        MyChat.this.txtUsernameDocumentChanged(evt);
    } else if (evt.getDocument().equals(txtMessage.getDocument()))
    {
        MyChat.this.txtMessageDocumentChanged(evt);
    }
}

//User-specified handlers
/**
 * This handler watches the txtUsername's text; the btnLogon is enabled
 * only when that text is other than the empty string.
 */
public void
txtUsernameDocumentChanged(javax.swing.event.DocumentEvent evt) {
    //This runs in the event-handling thread ("AWT-EventQueue-0")
    // A user name must be specified before btnLogon may be clicked
    if (txtUsername.getText().length() > 0)
    {
        btnLogon.setEnabled(true);
        btnLogon.getRootPane().setDefaultButton(btnLogon);
    } else {
        btnLogon.setEnabled(false);
    }
}
/**
 * This handler watches the txtMessage's text; the btnSendMessage
 * is enabled only when that text is other than the empty string.
 */

public void txtMessageDocumentChanged(javax.swing.event.DocumentEvent evt) {
    // This runs in the event-handling thread ("AWT-EventQueue-0")
    // A message must be specified before btnSendMessage may be clicked
    if (txtMessage.getText().length() > 0) {
        btnSendMessage.setEnabled(true);
        btnSendMessage.getRootPane().setDefaultButton(btnSendMessage);
    } else {
        btnSendMessage.setEnabled(false);
    }
}

// End of DocumentListener implementation and dispatching code
/**
 * Occurs when the window is first opened.
 */
private void openForm(java.awt.event.WindowEvent evt)
{
    //This runs in the event-handling thread ("AWT-EventQueue-0")
    lblStatus.setText("MyChat initialising - RTIambassador obtained");
    new Thread() { public void run()
    {
        //Join federation
        if (!joinFederation()) return;
        try {
            ChatRoomRegistry = new aChatRoomRegistry(); //We couldn't do this in the field declarations because of the escaping Exception
            } catch (CouldNotDecode ignored) {
                } //try
        //The ChatRoomRegistry is created with the "waiting_room" and
        "<General>" slots already filled in
        if (!getHandles()) return;
        if (!setupChatRoomRegistry()) return;
        if (!setupChatRooms()) return;
        if (!setupParticipants()) return;
        SwingUtilities.invokeLater(new Runnable() { public void run()
        {
            //Remaining initialisation
            try {
                txtUsername.setEnabled(true); //This'll allow btnLogon to become enabled
                txtUsername.requestFocus();
                lblStatus.setText("Welcome to MyChat - Please log in");
            } catch (Exception e) {
                e.printStackTrace();
            } //try
            } }); //Runnable
        } }); //Thread
}
private boolean joinFederation()
{
    // Runs on a separate thread -- not the event dispatch thread
    // Remaining initialisation
    try {
        // Process the command-line arguments
        if (_args.length > 0) rtiHost = _args[0];
        // NetBeans runs this in its root directory, e.g. "C:\Program Files\J2SDK\NetBeans3.5"
        if (_args.length > 1) fdd = _args[1];
        if (_args.length > 2) fedex = _args[2];
        if (_args.length > 3) fedname = _args[3];
        // Get the RTIambassador
        try {
            _rtiAmbassador = RTI.getRTIambassador(rtiHost, CRC_PORT);
        } catch (Exception e) {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run()
                {
                    lblStatus.setText("Unable to connect to CRC on " + rtiHost + ":" + CRC_PORT);
                }
            }); // Runnable
            return false;
        } // try
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run()
            {
                lblStatus.setText("MyChat initialising - RTIambassador obtained");
            }
        }); // Runnable
        // Get the FederateAmbassador
        // FedAmbWrapper provides a dispatching layer around the FederateAmbassador itself
        try {
            _fedAmbassador = new FedAmbWrapper(_rtiAmbassador);
        } catch (Exception e) {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run()
                {
                    lblStatus.setText("Unable to create new FedAmbWrapper");
                }
            }); // Runnable
            return false;
        } // try
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run()
            {
                lblStatus.setText("MyChat initialising - FedAmbassador obtained");
            }
        }); // Runnable
    }
}
// Destroy any lingering empty federation execution
try {
    _rtiAmbassador.destroyFederationExecution(fedex);
    SwingUtilities.invokeLater(new Runnable() { public void run() {
        lblStatus.setText("MyChat initialising - Federation destroyed");
    } }); //Runnable
} catch (FederatesCurrentlyJoined ignored) {
} catch (FederationExecutionDoesNotExist ignored) {
} //try

// Create the federation execution
final File fddFile = new File(fdd);
try {
    rtiAmbassador.createFederationExecution(fedex, fddFile.toURL());
    SwingUtilities.invokeLater(new Runnable() { public void run() {
        lblStatus.setText("MyChat initialising - Federation created");
    } }); //Runnable
} catch (FederationExecutionAlreadyExists ignored) {
} catch (CouldNotOpenFDD cnof) {
    SwingUtilities.invokeLater(new Runnable() { public void run() {
        lblStatus.setText("Could not open FDD " + fddFile.getAbsolutePath().toString() + "");
    } }); //Runnable
    return false;
} catch (ErrorReadingFDD erf) {
    SwingUtilities.invokeLater(new Runnable() { public void run() {
        lblStatus.setText("Corrupt FDD " + fddFile.getAbsolutePath().toString() + "");
    } }); //Runnable
    return false;
} //try

// Join the federation execution
_federateHandle = _rtiAmbassador.joinFederationExecution(fedname, fedex, _fedAmbassador, null);
SwingUtilities.invokeLater(new Runnable() { public void run() {
    lblStatus.setText("MyChat initialising - Joined as " + _federateHandle);
} }); //Runnable
    return true;
} catch (Exception e) {
    e.printStackTrace();
    return false;
} //try
private boolean
getHandles()
{
    // Runs on a separate thread -- not the event dispatch thread
    try {
        // Obtain object/interaction and parameter/attribute handles
        _ich_Communication = _rtiAmbassador.getInteractionClassHandle("Communication");
        _iph_Communication_message = _rtiAmbassador.getParameterHandle(_ich_Communication, "message");
        _iph_Communication_sender = _rtiAmbassador.getParameterHandle(_ich_Communication, "sender");

        _och_ChatRoomRegistry =
            _rtiAmbassador.getObjectClassHandle("ChatRoomRegistry");
        _oah_ChatRoomRegistry_DeletePrivilege =
            _rtiAmbassador.getAttributeHandle(_och_ChatRoomRegistry, RTI.PrivilegeToDeleteObjectName); // "HLAprivilegeToDeleteObject"
        _oahs_ChatRoomRegistry_list =
            _rtiAmbassador.getAttributeHandle(_och_ChatRoomRegistry, "list");
        // Build the attribute handle sets
        _oahs_ChatRoomRegistry =
            _rtiAmbassador.getAttributeHandleSetFactory().create();
        _oahs_ChatRoomRegistry.add(_oah_ChatRoomRegistry_DeletePrivilege);
        _oahs_ChatRoomRegistry_forUpdate =
            _rtiAmbassador.getAttributeHandleSetFactory().create();
        _oahs_ChatRoomRegistry_forUpdate.add(_oah_ChatRoomRegistry_list);

        _oahs_ChatRoom =
            _rtiAmbassador.getObjectClassHandle("ChatRoom");
        _oah_ChatRoom_DeletePrivilege =
            _rtiAmbassador.getAttributeHandle(_oah_ChatRoom, RTI.PrivilegeToDeleteObjectName);
        _oah_ChatRoom_name =
            _rtiAmbassador.getAttributeHandle(_oah_ChatRoom, "name");
        _oah_ChatRoom_slot =
            _rtiAmbassador.getAttributeHandle(_oah_ChatRoom, "slot");
        // Build the attribute handle sets
        _oahs_ChatRoom =
            _rtiAmbassador.getAttributeHandleSetFactory().create();
        _oahs_ChatRoom.add(_oah_ChatRoom_DeletePrivilege);
        _oahs_ChatRoom.add(_oah_ChatRoom_name);
        _oahs_ChatRoom.add(_oah_ChatRoom_slot);
        _oahs_ChatRoom_forUpdate =
            _rtiAmbassador.getAttributeHandleSetFactory().create();
        _oahs_ChatRoom_forUpdate.add(_oah_ChatRoom_name);
        _oahs_ChatRoom_forUpdate.add(_oah_ChatRoom_slot);
_och_Participant = _rtiAmbassador.createObjectClassHandle("Participant");
_oah_Participant_DeletePrivilege = _rtiAmbassador.getAttributeHandle(_och_Participant,
RTI.PrivilegeToDeleteObjectName);
_oah_Participant_logged_in = _rtiAmbassador.getAttributeHandle(_och_Participant, "logged_in");
_oah_Participant_user_handle = _rtiAmbassador.getAttributeHandle(_och_Participant, "user_handle");
_oah_Participant_chat_room_slot = _rtiAmbassador.getAttributeHandle(_och_Participant, "chat_room_slot");
//Build the attribute handle sets
_oahs_Participant = _rtiAmbassador.getAttributeHandleSetFactory().create();
_oahs_Participant.add(_oah_Participant_DeletePrivilege);
_oahs_Participant.add(_oah_Participant_logged_in);
_oahs_Participant.add(_oah_Participant_user_handle);
_oahs_Participant.add(_oah_Participant_chat_room_slot);
_oahs_Participant_forUpdate = _rtiAmbassador.getAttributeHandleSetFactory().create();
_oahs_Participant_forUpdate.add(_oah_Participant_logged_in);
_oahs_Participant_forUpdate.add(_oah_Participant_user_handle);
_oahs_Participant_forUpdate.add(_oah_Participant_chat_room_slot);

_dh_UserHandleSlots = _rtiAmbassador.getDimensionHandle("UserHandleSlots");
_dh_ChatRoomSlots = _rtiAmbassador.getDimensionHandle("ChatRoomSlots");
_dhs_UserHandleSlotsSet = _rtiAmbassador.getDimensionHandleSetFactory().create();
_dhs_UserHandleSlotsSet.add(_dh_UserHandleSlots);
_dhs_ChatRoomSlotsSet = _rtiAmbassador.getDimensionHandleSetFactory().create();
_dhs_ChatRoomSlotsSet.add(_dh_ChatRoomSlots);

_rhs_nowhere_ChatRoom = _rtiAmbassador.getRegionHandleSetFactory().create();
_rh_nowhere_ChatRoom = _rtiAmbassador.createRegion(_dhs_ChatRoomSlotsSet);
// nowhere_ChatRoom is slot 0
_rtiAmbassador.setRangeBounds(_rh_nowhere_ChatRoom,
_dh_ChatRoomSlots, new RangeBounds(_slot_nowhere_ChatRoom, 1 +
_slot_nowhere_ChatRoom));
_rhs_nowhere_ChatRoom.add(_rh_nowhere_ChatRoom);
_rtiAmbassador.commitRegionModifications(_rhs_nowhere_ChatRoom);
_rhs_waiting_room_chatRoom =
_rtiAmbassador.getRegionHandleSetFactory().create();
_rh_waiting_room_chatRoom =
_rtiAmbassador.createRegion(_dhs_chatRoomSlotsSet);
    // waiting_room_chatRoom is slot 1
    _rtiAmbassador.setRangeBounds(_rh_waiting_room_chatRoom,
    _dh_chatRoomSlots, new RangeBounds(_slot_waiting_room_chatRoom, 1 +
    _slot_waiting_room_chatRoom));
    _rhs_waiting_room_chatRoom.add(_rh_waiting_room_chatRoom);

_rtiAmbassador.commitRegionModifications(_rhs_waiting_room_chatRoom);

    // The General chatRoom
_rhs_general_chatRoom =
_rtiAmbassador.getRegionHandleSetFactory().create();
_rh_general_chatRoom =
_rtiAmbassador.createRegion(_dhs_chatRoomSlotsSet);
    // General chatRoom is slot 2
    _rtiAmbassador.setRangeBounds(_rh_general_chatRoom,
    _dh_chatRoomSlots, new RangeBounds(_slot_general_chatRoom, 1 +
    _slot_general_chatRoom));
    _rhs_general_chatRoom.add(_rh_general_chatRoom);

_rtiAmbassador.commitRegionModifications(_rhs_general_chatRoom);
// _rh_current_ChatRoom = null;
// _rh_myParticipantRegion = null;
// // For chat_room_slot filtering, we associate all "forUpdate"
// attributes to _rh_ChatRoomSlots regions
// // We'd get InvalidRegionContext when subscribing if we
// included the DeletePrivilege
    _asrspl_Participant_nowhere = _rtiAmbassador.getAttributeSetRegionSetPairListFactory().create(1);
    _asrspl_Participant_nowhere.add(new AttributeRegionAssociation(_oahs_Participant_forUpdate,
        _rhs_nowhere_ChatRoom));
    _asrspl_Participant_waiting_room = _rtiAmbassador.getAttributeSetRegionSetPairListFactory().create(1);
    _asrspl_Participant_waiting_room.add(new AttributeRegionAssociation(_oahs_Participant_forUpdate,
        _rhs_waiting_room_ChatRoom));
    // There'll be no Communication traffic through the
    // waiting_room; this is used only to find out
    // (through the Interaction Advisories) whether there are
    // other federates or not.
    _fedAmbassador.setInteractionAdvisoryResponder(_ich_Communication, new MyCommunicationAdvisoryResponder());
    _fedAmbassador.setInteractionListener(_ich_Communication, new MyCommunicationInteractionListener());
    _rtiAmbassador.publishInteractionClass(_ich_Communication);
    _rtiAmbassador.subscribeInteractionClassWithRegions(_ich_Communication,
        _rhs_waiting_room_ChatRoom);
    return true;
} catch (Exception e) {
    e.printStackTrace();
    return false;
} //try

private boolean setupChatRoomRegistry()
{
    try {
        // With all objects, we cannot allow orphans (objects which have no instances owned by any federate) because they become undiscoverable. Therefore we can expect ownership of the "common objects" (i.e. the ChatRoomRegistry, the ChatRoom objects and, to a certain extent, the Participant objects) to flow between the federates as they join and leave the federation as well as when they require ownership for modification purposes.
        // Thus, for each class we must set up the following services: discovery, removal, attribute update (listen // and respond) and ownership. Whichever federate owns (temporarily) an object will keep an image of // that object, using the aChatRoomRegistry, aChatRoom and aParticipant objects.

        _fedAmbassador.setAttributeUpdateResponder(_och_ChatRoomRegistry, new MyChatRoomRegistryInstanceAttributeResponder());
        _rtiAmbassador.publishObjectClassAttributes(_och_ChatRoomRegistry, _oahs_ChatRoomRegistry_forUpdate);
        _fedAmbassador.setAttributeUpdateListener(_och_ChatRoomRegistry, new MyChatRoomRegistryInstanceAttributeListener());
        _fedAmbassador.setOwnershipListener(_och_ChatRoomRegistry, new MyChatRoomRegistryOwnershipListener(null));

        // Get the ChatRoomRegistry, create it if necessary
        // Reserve name
        synchronized(_sem_reservation)
        {
            _sem_reservation.value = false;

            _fedAmbassador.setNameReservationListener(_name_ChatRoomRegistry, new MyNameReservationListener(_sem_reservation));
            _rtiAmbassador.reserveObjectInstanceName(_name_ChatRoomRegistry);
            // Wait for reservation succeeded/failed
            waitFor(_sem_reservation);
            _fedAmbassador.setNameReservationListener(_name_ChatRoomRegistry, null);
            reservation_succeeded = _sem_reservation.value;
        } // synchronized(_sem_reservation)
if (reservation_succeeded)
{
    //Having reserved the name, we know we're the first
    federate to reach this point,
    //so we must create the ChatRoomRegistry.
    synchronized(_ChatRoomRegistry)
    {
        _ChatRoomRegistry.owned = _ChatRoomRegistry.subscribed
        = true;
        //
        _ChatRoomRegistry.divesting = false;
        _ChatRoomRegistry.handle =
        rtiAmbassador.registerObjectInstance(_och_ChatRoomRegistry,
        _name_ChatRoomRegistry);
    }
    //Update responder is already in place
    rtiAmbassador.subscribeObjectClassAttributes(_och_ChatRoomRegistry,
    _oahs_ChatRoomRegistry_forUpdate);
    //Update listener is already in place
    } //synchronized(_ChatRoomRegistry)
else {
    //Name reservation failed, which means there is an already
    extant instance
    synchronized(_sem_discovery)
    {
        _sem_discovery.value = false;
        _fedAmbassador.setDiscoveryListener(_och_ChatRoomRegistry, new
        MyChatRoomRegistryDiscoveryListener(_sem_discovery));
        //No removal responder required since the object is
        never deleted
        _ChatRoomRegistry.subscribed = true;
        rtiAmbassador.subscribeObjectClassAttributes(_och_ChatRoomRegistry,
        _oahs_ChatRoomRegistry_forUpdate);
        //Update listener is already in place
        waitFor(_sem_discovery);
        _fedAmbassador.setDiscoveryListener(_och_ChatRoomRegistry, null);
    } //if
    return true;
} catch (Exception e) {
    e.printStackTrace();
    return false;
} finally {
    } //try
}
private boolean setupChatRooms()
{
    try {
        _fedAmbassador.setDiscoveryListener(_och_ChatRoom, new
            MyChatRoomDiscoveryListener(null));
        _fedAmbassador.setRemovalResponder(_och_ChatRoom, new
            MyChatRoomRemovalResponder());
        _fedAmbassador.setOwnershipListener(_och_ChatRoom, new
            MyChatRoomOwnershipListener(null, null));
        _fedAmbassador.setAttributeUpdateListener(_och_ChatRoom, new
            MyChatRoomInstanceAttributeListener());
        // Note that if we wanted an instance-specific responder, we
        // could not put it in place before getting
        // _oh_myChatRoom back from registerObjectInstance or
        // getObjectInstanceHandle, so we could
        // conceivably miss a provideAttributeValueUpdate request
        // issued by another federate between the
        // registration and the setting up of the responder --unless
        // both statements are put in a single synchronized block.
        _fedAmbassador.setAttributeUpdateResponder(_och_ChatRoom, new
            MyChatRoomInstanceAttributeResponder());
        _rtiAmbassador.publishObjectClassAttributes(_och_ChatRoom,
            _oahs_ChatRoom_forUpdate);
    }
    catch (RTIException e)
    {
        e.printStackTrace();
    }

    // Reserve name
    boolean reservation_succeeded;
    synchronized(_sem_reservation)
    {
        _sem_reservation.value = false;
        _fedAmbassador.setNameReservationListener(_name_waiting_room_ChatRoom,
            new MyNameReservationListener(_sem_reservation));
        _rtiAmbassador.reserveObjectInstanceName(_name_waiting_room_ChatRoom);
        // Wait for reservation succeeded/failed
        waitFor(_sem_reservation);
        _fedAmbassador.setNameReservationListener(_name_waiting_room_ChatRoom,
            null);
        reservation_succeeded = _sem_reservation.value;
    } //synchronized(_sem_reservation)
if (reservation_succeeded)
{
    // Having reserved the name, we know we're the first
    // federate to reach this point,
    // so we must create the ChatRoom
    synchronized (AcquireChatRoomRegistry())
    {
        synchronized (_theChatRooms)
        {
            _waiting_room_ChatRoom.owned =
                _waiting_room_ChatRoom.subscribed = true;
            //
            _waiting_room_ChatRoom.divesting = false;
            // Update responder is already in place
            _theChatRooms.put (_waiting_room_ChatRoom.handle -
                _rtiAmbassador.registerObjectInstance (_och_ChatRoom,
                _waiting_room_ChatRoom.name.toString()), _waiting_room_ChatRoom);
            _rtiAmbassador.subscribeClassAttributes (_och_ChatRoom,
                _oahs_ChatRoom_forUpdate);
            // The waiting_room ChatRoom is already in the
            _ChatRoomRegistry, so no need to update it
        } // synchronized (_theChatRooms)
        } // synchronized (AcquireChatRoomRegistry())
    } else {
        // Name reservation failed, which means there is an already
        // extant instance
        synchronized (_sem_discovery)
        { 
            _sem_discovery.value = false;
            _fedAmbassador.setDiscoveryListener (_och_ChatRoom, new
            MyChatRoomDiscoveryListener (_sem_discovery));
            // This is the only pre-existing aChatRoom, so we must
            set its subscribed property here
            _waiting_room_ChatRoom.subscribed = true;
            _rtiAmbassador.subscribeClassAttributes (_och_ChatRoom,
                _oahs_ChatRoom_forUpdate);
            // Update listener is already in place
            waitFor (_sem_discovery);
            _fedAmbassador.setDiscoveryListener (_och_ChatRoom, new
            MyChatRoomDiscoveryListener (null));
        } // synchronized (_sem_discovery)
    } // if
    // The General ChatRoom may or may not exist initially
    return true;
} catch (Exception e) { 
    e.printStackTrace();
    return false;
} finally {
    } // try
private boolean setupParticipants()
{
    try {
        fedAmbassador.setDiscoveryListener(_och_Participant, new
             MyParticipantDiscoveryListener(null, null));
             // These objects are never deleted, so removal is impossible
             // Instead, they go in/out of scope
        _fedAmbassador.setAttributeScopeListener(_och_Participant, new
             MyParticipantAttributeScopeListener());
        _fedAmbassador.setOwnershipListener(_och_Participant, new
             MyParticipantOwnershipListener(null, null));
        _fedAmbassador.setAttributeUpdateListener(_och_Participant, new
             MyParticipantInstanceAttributeListener());
             // Note that if we wanted an instance-specific responder, we
             // could not put it in place before getting
             // _ohh_myParticipant back from registerObjectInstance or
             // getObjectInstanceHandle, so we could
             // conceivably miss a provideAttributeValueUpdate request
             // issued by another federate between the
             // registration and the setting up of the responder --unless
             // both statements are put in a single synchronized block.
        _fedAmbassador.setAttributeUpdateResponder(_och_Participant, new
             MyParticipantInstanceAttributeResponder());
        _rtiAmbassador.publishObjectClassAttributes(_och_Participant,
             _oahs_Participant_forUpdate);
             // Note that associateRegionsForUpdates can be done only on a
             // per-instance basis
             // We publish federation-wide but will update and subscribe
             // through DDM
        _rtiAmbassador.subscribeObjectClassAttributesWithRegions(_och_Participant,
             _asrspl_Participant_waiting_room);
             // Update listener is already in place

        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    } finally {
        // try
    }
}
/**  
* Used by various methods to acquire ownership of the  
ChatRoomRegistry.  
* It is expected to be called by a statement like:  
* synchronized(AcquireChatRoomRegistry()) {}  
*/  
private aChatRoomRegistry  
AcquireChatRoomRegistry()  
{
    //Can't synchronize on _ChatRoomRegistry because the waitFor  
    //argument, and the AcquisitionNotification will need to  
    //The Java wait method only frees up the monitor of the object  
    //"wait" for a *set* of monitors.  
    synchronized(_ChatRoomRegistry)  
    {  
        if (_ChatRoomRegistry.owned) return _ChatRoomRegistry;  
        synchronized(_sem_acquisition)  
        {
            try {
                FederateAmbassadorAttributeOwnership previous =  
                    (FederateAmbassadorAttributeOwnership)_fedAmbassador.setOwnershipListe  
                    _och_ChatRoomRegistry, new  
                    MyChatRoomRegistryOwnershipListener(_sem_acquisition));  
                _sem_acquisition.value = false;  
                _rtiAmbassador.attributeOwnershipAcquisition(_ChatRoomRegistry.handle,  
                    _oahs_ChatRoomRegistry, null);  
                waitFor(_sem_acquisition);  
            } catch (Exception ignored) {}  
            //try  
            } //synchronized(_sem_acquisition)  
            //Failure to acquire is not an option  
            return null;  
        // } //synchronized(_ChatRoomRegistry)  
    }
/**
  * Used by various methods to remove a ChatRoom from the registry and update the latter.
  * Remember that the ChatRoomRegistry only maps ChatRoom names to slots; it does not link
  * to ChatRoom object instances at all.
  *
  * @param slot a short specifying the ChatRoom entry to remove
  */
  
  private void
  RemoveChatRoomAndUpdateRegistry(short slot)
  {
      //Presumes _ChatRoomRegistry has been synchronized on
      try {
          boolean updateNeeded = false;
          if (slot <= _slot_general.ChatRoom) return; //waiting_room and General are fixtures
          for (int i = 0; i < _ChatRoomRegistry.list.size(); i++)
          {
              if (slot == ((ChatRoomRegistryEntry)_ChatRoomRegistry.list.get(i)).slot.getValue())
              {
                  _ChatRoomRegistry.list.remove(i);
                  updateNeeded = true;
                  break;
              } //if
          } //for
          if (!updateNeeded) return;
          AttributeHandleValueMap _ahvm.ChatRoomRegistry =
          _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs.ChatRoomRegistry_forUpdate.size());
          _ahvm.ChatRoomRegistry.put(_oah.ChatRoomRegistry_list,
          _ChatRoomRegistry.list.toByteArray());
          _rtiAmbassador.updateAttributeValues(_ChatRoomRegistry.handle,
          _ahvm.ChatRoomRegistry, null);
          catch (Exception ignored) {
              finally {
                  //try
                  }
          } //catch
      } catch (Exception ignored) {
      } finally {
  } //try
  }
/**
 * Used by various methods to add a ChatRoom to the registry and update the latter.
 * The ChatRoom is also added to _theChatRooms
 * @param name a String specifying the ChatRoom name to give to the new entry (it will be prefixed by the method)
 * @return the newly created aChatRoom object (null in case of failure)
 */
private aChatRoom AddChatRoomAndUpdateRegistry(String name)
{
    // Must be called from the event-handling thread
    // Presumes _ChatRoomRegistry has been synchronized on
    try {
        // To allow user names and chat room names to be anything, the latter will be prefixed
        // by "p" and "c", respectively, whilst our reserved names will be prefixed by "*
        String newChatRoomName = "c" + name;
        aChatRoom _ChatRoom = SeekChatRoomByName(newChatRoomName);
        if (_ChatRoom != null)
        {
            javax.swing.JOptionPane.showMessageDialog(null, "The chat room name " + name + " already exists. Sorry!",
                javax.swing.JOptionPane.INFORMATION_MESSAGE);
            return null;
        }
        // At this point we know the name is OK; now find a slot
        short candidate = _slot_FirstFreeChatRoomSlot;
        // Each SeekChatRoom method synchronizes on _theChatRooms
        for (; null != SeekChatRoomBySlot(candidate); candidate++)
        {
            // We'll use candidate for the slot value
            _ChatRoomRegistry.list.add(_ChatRoomRegistry.list.size(), new
                HLAinteger16BE(candidate));
            AttributeHandleValueMap _ahvm_ChatRoomRegistry =
                _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs_ChatRoomRegistry_forUpdate.size());
            _ahvm_ChatRoomRegistry.put(_oah_ChatRoomRegistry_list, _ChatRoomRegistry.list.toByteArray());
            _rtiAmbassador.updateAttributeValues(_ChatRoomRegistry.handle, _ahvm_ChatRoomRegistry, null);
        }
    }
}
ChatRoom = new aChatRoom(newChatRoomName, candidate);
_chatRoom.subscribed = _chatRoom.owned = true;
synchronized (_theChatRooms)
{
    _theChatRooms.put (_chatRoom.handle =
    _rtiAmbassador.registerObjectInstance (_och _chatRoom), _chatRoom);
} //synchronized (_theChatRooms)
return _chatRoom;
} catch (Exception ignored) {
    finally {
        try
        return null;
    }
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt)
{
    try {
        _me_shutting_down = true;
        //This runs in the event-handling thread ("AWT-EventQueue-0")
        if (_rtiAmbassador!=null)
        {
            _lblStatus.setText("MyChat shutting down...");
            if (!_logged_in) doLogout();
            // if ("Log Out".equals(btnLogon.getText())) doLogout();
            doUnsubscribe();
            if (_alone) doForceDivest();
            doUnpublish();
            try {
              //Any remaining owned objects will be deleted (there
              should be none when !_alone, really) (well, maybe some Region
              objects...)

              _rtiAmbassador.resignFederationExecution(ResignAction.DELETE_OBJECTS_T
              HEN_DIVEST);
              //OwnershipAcquisitionPending, FederateOwnsAttributes,
FederateNotExecutionMember, RTIinternalError
              _lblStatus.setText("MyChat shutting down - Resigning
              from federation");
              _rtiAmbassador.destroyFederationExecution(fedex);
              //FederatesCurrentlyJoined,
FederationExecutionDoesNotExist, RTIinternalError
              _lblStatus.setText("MyChat shutting down - Federation *
+ fedex + " destroyed");
            } catch (RTIexception ignored) {
                try
            } //if
        } finally {
            System.exit (0);
        } //try
    }
}
private void doUnsubscribe()
{
    try {
        _rtiAmbassador.unsubscribeInteractionClassWithRegions(_ich_Communication, _rhs_waiting_room_ChatRoom);

        _rtiAmbassador.unsubscribeObjectClassAttributes(_och_ChatRoom, _oahs_ChatRoom_forUpdate);

        _rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(_och_Participant, _asrsubj_Participant_waiting_room);

        _rtiAmbassador.unsubscribeObjectClassAttributes(_och_ChatRoomRegistry, _oahs_ChatRoomRegistry_forUpdate);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
    } //try
}

private void doUnpublish()
{
    try {
        _rtiAmbassador.unpublishInteractionClass(_ich_Communication);

        _rtiAmbassador.unpublishObjectClassAttributes(_och_ChatRoom, _oahs_ChatRoom_forUpdate);
        //unassociateRegionsForUpdates can only be done on a per-instance basis

        _rtiAmbassador.unpublishObjectClassAttributes(_och_Participant, _oahs_Participant_forUpdate);

        _rtiAmbassador.unpublishObjectClassAttributes(_och_ChatRoomRegistry, _oahs_ChatRoomRegistry_forUpdate);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
    } //try
}
private void
divestChatRoomRegistry()
{
// This method is invoked when the federate wants to shut down
but still owns some instance attributes
// This usually runs in the event-handling thread ("AWT-
EventQueue-0") but can be called from other threads
try {
    if (!ChatRoomRegistry.owned) return;
    Semaphore _sem_divest = new Semaphore();
    synchronized (_sem_divest)
    {
        _sem_divest.value = false;
        java.lang.Object _oldListener =
        _fedAmbassador.setOwnershipListener(_och_ChatRoomRegistry, new
        MyChat.MyChatRoomRegistryOwnershipListener(_sem_divest));
        _ChatRoomRegistry.divesting = true;
        _rtiAmbassador.negotiatedAttributeOwnershipDivestiture(_ChatRoomRegis-
try.handle, _oahs_ChatRoomRegistry, null);
        // Other federates will receive
        requestAttributeOwnershipAssumption and
        // respond with attributeOwnershipAcquisitionIfAvailable.
        // We'll get requestDivestitureConfirmation, to which we'll
        confirmDivestiture
        // Once that is done, we no longer own _chatRoomRegistry
        and can resign normally.
        waitFor (_sem_divest);
        // _fedAmbassador.setOwnershipListener(_och_ChatRoomRegistry,
        null);
        _fedAmbassador.setOwnershipListener(_och_ChatRoomRegistry,
        (FederateAmbassadorAttributeOwnership)_oldListener);
        } // synchronized (_sem_divest)
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
    // try
    }
private void divestChatRooms()
{
    // This method is invoked when the federate wants to shut down
    // but still owns some instance attributes
    // This usually runs in the event-handling thread ("AWT-
    // EventQueue-0") but can run from other threads
    try {
        // ChatRoom objects
        aChatRoom _ChatRoom;
        Semaphore _sem_divest = new Semaphore();
        synchronized(_sem_divest) {
            java.lang.Object _oldListener =
            _fedAmbassador.setOwnershipListener(_och_ChatRoom, new
            MyChat.MyChatRoomOwnershipListener(_sem_divest, null));
            // Any federate callbacks requesting ownership assumption
            will be turned down right away;
            // however, ownership acquisition notifications could
            conceivably squeak through
            // (if another federate started shutting down just before
            this one does) and would lock
            // up on the synchronized(_theChatRooms), so we take a
            snapshot and release the monitor right away.
            aChatRoom[] _ChatRooms;
            synchronized(_theChatRooms) {
                _ChatRooms = (aChatRoom[]) _theChatRooms.values().toArray(new aChatRoom[0]);
                // synchronized(_theChatRooms)
                for (Iterator i = _theChatRooms.entrySet().iterator();
                     i.hasNext();)
                for (int i = 0; i < _ChatRooms.length; i++) {
                    _ChatRoom =
                    (aChatRoom)((java.util.Map.Entry)i.next()).getValue();
                    _ChatRoom = _ChatRooms[i];
                    slot: " + _ChatRoom.slot.toString() + "
                    handle: " + _ChatRoom.handle.toString();
                    if (_ChatRoom.owned) {
                        _sem_divest.increaseCount();
                        _ChatRoom.divesting = true;
                        _rtiAmbassador.negotiatedAttributeOwnershipDivestiture(_ChatRoom.handle,
                        _oahs_ChatRoom, null); // Other federates will receive
                        requestAttributeOwnershipAssumption and
                        // respond with attributeOwnershipAcquisitionIfAvailable.
                        // We'll get requestDivestitureConfirmation, to which
                        we'll confirmDivestiture
                        // Once that is done, we no longer own any ChatRooms
                        and can resign normally.
                    } // if
                } // for
if (!_sem_divest.zero())
{
    waitFor(_sem_divest);
} //if

_fedAmbassador.setOwnershipListener(_och_ChatRoom, new
MyChat.MyChatRoomOwnershipListener(null, null));
_fedAmbassador.setOwnershipListener(_och_ChatRoom,
{FederateAmbassadorAttributeOwnership} _oldListener);  
} //synchronized(_sem_divest)
} catch (Exception e) {
    e.printStackTrace();
} finally {
    //try

private void
divestParticipants()
{
    //This method is invoked when the federate wants to shut down
    but still owns some instance attributes
    //This usually runs in the event-handling thread ("AWT-
    EventQueue-0") but could be called from other threads

    try {
        //Participant objects
        aParticipant _participant;
        Semaphore _sem_divest = new Semaphore();
        synchronized(_sem_divest)
        {
            java.lang.Object _oldListener =
_fedAmbassador.setOwnershipListener(_och_Participant, new
MyChat.MyParticipantOwnershipListener(_sem_divest, null));
            //Any federate callbacks requesting ownership assumption
            will be turned down right away;
            //however, ownership acquisition notifications could
            conceivably squeak through
            //(if another federate started shutting down just before
            this one does) and would lock
            //up on the synchronized(_theParticipants), so we take a
            snapshot and release the monitor right away.
            aParticipant[] _Participants;
            synchronized(_theParticipants)
            {
                _Participants = (aParticipant[])
_theParticipants.values().toArray(new aParticipant[0]);
            } //synchronized(_theParticipants)
for (Iterator i = _theParticipants.entrySet().iterator();
i.hasNext();)
    [ //Participant =
        (java.util.Map.Entry)i.next().getValue();
        _Participant = _Participants[i];
    ]

    slot: " + _Participant.chat_room_slot.toString();
    handle: " + _Participant.user_handle.toString();
    if (_Participant.owned)
        [ _sem_divest.increaseCount();
          _Participant.divesting = true;
          _rtiAmbassador.negotiatedAttributeOwnershipDivestiture(_Participant.handle, _oahs_Participant, null);
          //Other federates will receive requestAttributeOwnershipAssumption and respond with attributeOwnershipAcquisitionIfAvailable.
          //We'll get requestDivestitureConfirmation, to which we'll confirmDivestitureOnce that is done, we no longer own any Participants and can resign normally.
        ]
    if (!_sem_divest.zero();)
        [ waitFor(_sem_divest);
          //fedAmbassador.setOwnershipListener(_ooh_Participant, new MyChat.MyParticipantOwnershipListener(null, null));
          _fedAmbassador.setOwnershipListener(_ooh_Participant,
              (FederateAmbassadorAttributeOwnership)_oldListener);
          synchronized(_sem_divest) ]
          catch (Exception e) {
              e.printStackTrace();
          } finally {
              //try
          }
      ]
private void
doForceDivest() {

    // This method is invoked when the federate wants to shut down
    // but still owns some instance attributes
    // This runs in the event-handling thread ("AWT-EventQueue-0")
    try {
        // ChatRoomRegistry object
        divestChatRoomRegistry();
        // ChatRoom objects
        divestChatRooms();
        // Participant objects
        divestParticipants();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // try
    }

    /**
     * The btnLogon is used to Log On and to Log Out.
     */
    private void btnLogonActionPerformed(java.awt.event.ActionEvent evt) {

        // This runs in the event-handling thread ("AWT-EventQueue-0")
        if (!me_logged_in) // ("Log Out".equals(btnLogon.getText()))
        {
            // Log out procedure
            doLogout();
        } else {
            // Log In procedure
            doLogin();
        } // if
    }
private void doLogin()
{
    try {
        // This runs in the event-handling thread ("AWT-EventQueue-0")
        _me_logging_in = true;
        if (!registerParticipant())
        {
            _me_logging_in = false;
            return;
        } // if
        _me_logged_in = true;

        _rtiAmbassador.subscribeInteractionClassWithRegions(_ich_Communication
            , _rhs_current_ChatRoom);

        _rtiAmbassador.subscribeInteractionClassWithRegions(_ich_Communication
            , _rhs_myParticipantRegion);
        _me_logging_in = false;
        txtUsername.setEnabled(false);
        btnLogon.setText("Log Out");
        //
        btnSendMessage.setEnabled(true);
        txtMessage.setEnabled(true);
        lblChatRoom.setEnabled(true);
        lstChatRooms.setEnabled(true);
        btnNewChatRoom.setEnabled(true);
        lblSendTo.setEnabled(true);
        lstSendTo.setEnabled(true);
        txtMessage.requestFocus();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
    } //try
}
private boolean registerParticipant()
{
    java.util.Map.Entry _m;
    ObjectInstanceHandle _h;
    aParticipant _p;
    aParticipant _participant;
    // This runs in the event-handling thread ("AWT-EventQueue-0")
    because it is called from it

    // This method's purpose is to validate the user name;
    // if it succeeds, the Participant object with the
    // requested user name is now owned by this federate.
    // We know it won't be an empty String because the button is
    disabled when the txtUsername is empty.
    _me.name.setValue("p" + txtUsername.getText());
    // _me.handle = null; // ObjectInstanceHandle of the Participant
    object
    // _me.user_handle.setValue(0); // Same value, as an HLAint32 for
    DDM purposes
    try {
        // Reserve name
        boolean reservation_succeeded;
        synchronized(_sem_reservation)
        {
            _sem_reservation.value = false;

            _fedAmbassador.setNameReservationListener(_me.name.toString(),
new
MyNameReservationListener(_sem_reservation));

            _rtiAmbassador.reserveObjectInstanceName(_me.name.toString());
            // Wait for reservation succeeded/failed
            waitFor(_sem_reservation);

            _fedAmbassador.setNameReservationListener(_me.name.toString(),
null);
            reservation_succeeded = _sem_reservation.value;
        } // synchronized(_sem_reservation)
//Prepare the <General> ChatRoom
_rhs_current_ChatRoom = _rtiAmbassador.getRegionHandleSetFactory().create();

//Arrival <General> ChatRoom is slot 2
_rhs_current_ChatRoom = _rh_general_ChatRoom;
_rhs_current_ChatRoom.add(_rh_current_ChatRoom);

_rtiAmbassador.commitRegionModifications(_rhs_current_ChatRoom);

//Won't complain if there are no mods to commit
_asrspl_Participant_current = _rtiAmbassador.getAttributeSetRegionSetPairListFactory().create(1);
_asrspl_Participant_current.add(new AttributeRegionAssociation(_oahs_Participant_forUpdate,
_rhs_current_ChatRoom));

//Note that the prepared association has not been applied yet
if (reservation succeeded)
{
  //For chat_room_slot filtering, we associate all
"forUpdate* attributes to _dh_ChatRoomSlots regions,
//We'd get InvalidRegionContext when subscribing if we
included the DeletePrivilege
  synchronized[_me]
  {
    _me.logged_in.setBoolean(_me.owned = _me.subscribed =
!(me.inscope = _me.divesting = false));
    _me.chat_room_slot.setValue(_slot_nowhere_ChatRoom);
    //If there were any federates subscribed to the
creation region (nowhere),
    //there would be Discovery and Auto-Update would
trigger our InstanceAttributeResponder and all that;
    //Hence the atomic register-and-put; the
  synchronized(_me) would delay a bit until the _me.user_handle is set
    //But since no-one subscribes to nowhere, there is no
  rush
  synchronized(_theParticipants)
  {
    _theParticipants.put(_me.handle =
_rtiAmbassador.registerObjectInstanceWithRegions(_och_Participant,
_asrspl_Participant_nowhere, _me.name.toString()), _me);
  } //synchronized(_theParticipants)
  //_me.user_handle.setValue(ObjectInstanceHandleToInt(_me.handle));
    _me.user_handle.setValue(_me.handle.hashCode());
  } //synchronized(_me)
At this point, we know the desired Participant object exists -- we just don't know where it is.

// If it is logged-out, it'll be in the waiting_room and we will discover it there and can then acquire it.
// It is logged-in by someone else, it'll be in some other ChatRoom (2+) and we won't discover it
// and cannot use it. There is no "fail-fast" in that situation, however.
// It is reasonably safe to assume that if we don't know of the sought Participant, it must be logged in by someone else.

// Is the object known?
try {
  try {
    Participant = SeekParticipant(_rtiAmbassador.getObjectInstanceHandle(_me.name.toString()));
  } catch (ObjectInstanceNotKnown ex) {
    // The Participant was unknown; it cannot be in the waiting_room
    _lblStatus.setText(_me.name.toString().substring(1) + " is already logged-in. Sorry!");
    _me.name.setValue("*");
    return false;
  }
} finally {
  // At this point, the sought Participant is known

  // If freshly discovered, we could wait for its user_handle to be updated by reflectAttributeValue
  // (this is the only field guaranteed to change) so that we can then safely look up its logged_in or chat_room_slot values.
  // Instead, we'll request ownership right away; a logged-in Participant will fail.

  // It could be already owned (if it was an unused Participant we just happened to have the custody of)
  if (!Participant.owned) {
    // For a Participant to be eligible for acquisition, it must be in the
    // _waiting_room and therefore in scope
    boolean acquisition_succeeded = false;

if (_Participant.inscope) {
    // If the _Participant goes out of scope before we
    // get to the attributeOwnershipAcquisition call,
    // we'll be stuck at the semaphore. This may happen
    if there is a race for the same Participant.
    // About the only way around this risk I can see may
    // be to have the acquiring federates first
    // grab the ChatRoomRegistry (before testing
    // inscope), releasing it once the Participant
    // acquisition is completed.
    // This is a form of queueing, where the federates
    // are passing between themselves a token
    // (the ChatRoomRegistry) which gives them the
    "right" to attempt acquisition. The
    // disadvantage is that it creates an artificial
    bottleneck (and extra traffic).
    synchronized(_sem_acquisition) {
        _sem_acquisition.value = false;
        _fedAmbassador.setOwnershipListener(_och_Participant, new
        MyParticipantOwnershipListener(_sem_acquisition,
        _Participant.handle));
        // Before acquiring, set up update regions so they
do not lapse during ownership transfer
        _rtiAmbassador.associateRegionsForUpdates(_Participant.handle,
        _asrsp1_Participant_nowhere);
        _rtiAmbassador.associateRegionsForUpdates(_Participant.handle,
        _asrsp1_Participant_waiting_room);
        _rtiAmbassador.attributeOwnershipAcquisition(_Participant.handle,
        _oahs_Participant, null);
        waitFor(_sem_acquisition);
        _fedAmbassador.setOwnershipListener(_och_Participant, new
        MyParticipantOwnershipListener(null, null));
        acquisition_succeeded = _sem_acquisition.value;
    } //synchronized(_sem_acquisition)
    // Acquisition failed: someone else is logged in
    under that Participant
    lblStatus.setText(_me.name.toString().substring(1) +
    " is already logged-in. Sorry!");
    _me.name.setValue(""");
    return false;
} //if
// Acquisition succeeded: that Participant was available
} //if
synchronized(_me)
{
    _me = _Participant;
    // _me.divesting = false; //could be still true; will be
    fixed later
    _me.logged_in.setBoolean(_me.owned = true);
    //
    _me.user_handle.setValue(ObjectInstanceHandleToInt(_me.handle));
    _me.user_handle.setValue(_me.handle.hashCode());
    //Removing _me from the waiting_room will trip
    Attribute Scope Advisories
    _rtiAmbassador.unassociateRegionsForUpdates(_me.handle,
    _asrsl_Participant_waiting_room);
    //At this point, _me.handle is only associated with
    _asrsl_Participant_nowhere
    _me.chat_room_slot.setValue(_slot_nowhere_chatRoom);
    } //synchronized(_me)
    } //if
    //Does the General ChatRoom exist?
    synchronized(_theChatRooms)
    {
        _general_ChatRoom =
        SeekChatRoomBySlot(_slot_general_chatRoom);
        if (null == _general_ChatRoom)
        {
            synchronized(AcquireChatRoomRegistry())
            {
                _general_ChatRoom = new
                aChatRoom(_name_general_chatRoom, _slot_general_chatRoom);
                _general_ChatRoom.owned =
                _general_ChatRoom.subscribed = true;
                // _general_ChatRoom.divesting = false;
                _theChatRooms.put(_general_ChatRoom.handle =
                _rtiAmbassador.registerObjectInstance(_och_ChatRoom),
                _general_ChatRoom);
                //No need to add the General ChatRoom to the
                _ChatRoomRegistry, as it is already listed
                } //synchronized(AcquireChatRoomRegistry())
        } else {
        } //if
    } //synchronized(_theChatRooms)
It is just barely possible, given thread sequencing and RTI transmission delays, that two (or more) copies of the general chat room (or any other chat room except for the waiting room) may end up being created. This does not matter for Participant regions since the slots won't be affected. (the ChatRoomRegistry controls slot assignment, and its ownership is passed from federate to federate).

We could try to cut down on such situations by checking upon Discovery -- if one discovers a ChatRoom duplicate (same name and therefore slot), and one owns one of the two, acquire the Registry (this being a race condition, the owner of the duplicate could be trying to do the same, so this acquisition acts as a tie-breaker) then, if the condition still holds, acquire aggressively the duplicate. Once that is achieved, delete the duplicate. Removal responders would need to check for duplicates in their lists of ChatRooms, so if one is advised of the removal of _general_ChatRoom or _myChatRoom (the only two that matter), effect the substitution rather than setting the member to null. Substitution would also inhibit deletion of the entry in lstChatRooms.

When shutting down a chat room, we should also delete repeatedly until the name does not appear in the list any more.

The converse is also just barely possible: one federate could delete a ChatRoom just as another is in the process of joining it. Again, this is a matter for the Removal responder. If one is advised of the Removal of a ChatRoom that matters, and there is no duplicate extant, then one must recreate the ChatRoom object.

Another possible solution to these kinds of problems would be to make the federates time-constrained and time-regulating. Time-stamping the deletions and creations of chat rooms could be used to ensure that all federate callbacks are processed before any action is taken.

For expediency's sake, this level of coding has not been implemented here.
// This association will trigger Attribute Scope Advisories,  
Discovery:

_myChatRoom = _general_ChatRoom;
_me.chat_room_slot.setValue(_slot_general_ChatRoom);
_me_logged_in = !_(!me_logging_in == false);

// If you associate first and then subscribe, a previously
known object won't trip the scope advisories (a)

_rtiAmbassador.subscribeObjectClassAttributesWithRegions(_och_Participant, _arsrpl_Participant_current);

// Add the _general ChatRoom association to the pre-existing
_nowhere ChatRoom association
_rtiAmbassador.associateRegionsForUpdates(_me.handle,
_rsrpl_Participant_current);

// Subscription will refresh subscribed Participants;
// since we use Auto-Provide, an explicit Update request is
not needed

lblStatus.setText("Welcome, " +
_me.name.toString().substring(1));

// Subscribe to the interaction
// We'll be listening to interactions that come in on our
user-handle channel in addition to the chat-room-slot channel
// It is the interaction sender's responsibility to pick the
channels to use

_rh_myParticipantRegion =
_rtiAmbassador.createRegion(_dhs_UserHandleSlotsSet);
// setRangeBounds must be invoked at least once for each
dimension specified.
// Only then can commitRegionModifications be invoked to turn
the region template into a region specification.
_rtiAmbassador.setRangeBounds(_rh_myParticipantRegion,
_dh_UserHandleSlots, new RangeBounds(_me.user_handle.getValue(),
_me.user_handle.getValue() + 1));

_rhs_myParticipantRegion =
_rtiAmbassador.getRegionHandleSetFactory().create();
_rhs_myParticipantRegion.add(_rh_myParticipantRegion);

_rtiAmbassador.commitRegionModifications(_rhs_myParticipantRegion);

// subscribe InteractionClassWithRegions done by doLogin

return true;
} catch (RTIException e) {
  e.printStackTrace();
  return false;
} finally {
  // try
}
private void
dologout()
{
    try {
        // This runs in the event-handling thread ("AWT-EventQueue-0")
        // Log out procedure
        _me_logged_in = false;
        // Move from "current" ChatRoom to "waiting_room" ChatRoom
        // Unsubscribe from the current ChatRoom

        _rtiAmbassador.unsubscribeInteractionClassWithRegions(_ich_Communicati
            on, _rhs_current_ChatRoom);
        // Unsubscribe from the user handle channel

        _rtiAmbassador.unsubscribeInteractionClassWithRegions(_ich_Communicati
            on, _rhs_myParticipantRegion);

        // Look at the ChatRoom we're leaving to decide whether to
delete it or not
        int count = 0;
        synchronized(_theParticipants)
        {
            aParticipant _iParticipant;
            for (Iterator i = _theParticipants.values().iterator();
                i.hasNext();)
            {
                _iParticipant = (aParticipant)i.next();
                // Out of scope Participants have unreliable attributes;
                they cannot be in our ChatRoom in any case
                // The only exceptions are owned Participants: any owned
                orphans can only be in the waiting_room
                // _me isn't counted
                if (((!_me.handle.equals(_iParticipant.handle)) &&
                    (_iParticipant.inscope) &&
                    (_me.chat_room_slot.equals(_iParticipant.chat_room_slot))) count++;
            } // for
            // synchronized(_theParticipants)

            // This'll cause the Participants in our ChatRoom to go out of
            scope

            _rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(_och_Partic
                ipant, _asrspl_Participant_current);

            // What if someone logs into the ChatRoom after we've counted
            its Participants but before we actually delete it?
// Doesn't seem worth synchronizing on _myChatRoom
if (count <= 0) // Should be just == 0 but you never know
(because of inscope, _me isn't counted)
{
  // We were the only Participant left in the ChatRoom;
delete it as we leave (we must be the owner, obviously)
synchronized(AcquireChatRoomRegistry())
{
  // theChatRooms.remove(_myChatRoom.handle); // Because
  // we're the owner, we won't get a RemoveObjectInstance notification
  // _rtiAmbassador.deleteObjectInstance(_myChatRoom.handle,
  // null);
  // Because we're the owner, we won't get a
  RemoveObjectInstance notification
  // The remove returns the removed value, which is
  _myChatRoom

  _rtiAmbassador.deleteObjectInstance(((aChatRoom)_theChatRooms.remove(_
  myChatRoom.handle)).handle, null);

  RemoveChatRoomAndUpdateRegistry(_myChatRoom.slot.getValue());
  if
  {(_myChatRoom.name.toString().equals(_name_general_chatRoom))
   
   lstChatRooms.removeItem(_myChatRoom.name.toString().substring(1));
   } //if
  } //synchronized(AcquireChatRoomRegistry())
} else if (_myChatRoom.owned)
{
  // There are other Participants but we were the owner:
  transfer ownership
  _myChatRoom.divesting = true;

  _rtiAmbassador.negotiatedAttributeOwnershipDivestiture(_myChatRoom.han
dle, _oahs_chatRoom, null);
} //if

  _myChatRoom = null; // Should be OK; if being divested, it
  still has a reference from theChatRooms and therefore won't be
  garbaged
The _me Participant is the only one that can be elsewhere than the waiting room and be owned; by definition, an owned Participant stands in for the federate in whichever ChatRoom it is. Move it from current to waiting room

```
_rtiAmbassador.unassociateRegionsForUpdates(_me.handle,
_asrspl_Participant_current);

synchronized(_me) {

    _me.logged_in.setBoolean(false);
    _me.chat_room_slot.setValue(_slot_waiting_room.ChatRoom);
    _rtiAmbassador.associateRegionsForUpdates(_me.handle,
    _asrspl_Participant_waiting_room);

    //AutoProvide will fetch our attribute updates (since all federates subscribe to the waiting_room ChatRoom at all times)
}

synchronized(_me) {

    _me = new aParticipant();

    //Delete the now unused regions
    _rhs_myParticipantRegion.remove(_rh_myParticipantRegion);
    _rtiAmbassador.deleteRegion(_rh_myParticipantRegion);
    _rh_myParticipantRegion = null;
    _rh_myParticipantRegion = null;
    if (_rh_current(ChatRoom).equals(_rh_generalChatRoom))
    
        _rtiAmbassador.deleteRegion(_rh_currentChatRoom);
        _rh_currentChatRoom = null;
    }

    //The lstSendTo will be emptied by the AttributeScopeListener
    for (int i = 1; i < lstSendTo.getItemCount(); i++)
        lstSendTo.removeItemAt(i);

    btnSendMessage.setEnabled(false);
    txtMessage.setEnabled(false);
    btnCancelChatRoom.setEnabled(false);
    lstChatRooms.setSendTo(0); //So that we log back into the _<General> chat room
    lstChatRooms.setEnabled(false);
    btnCancelChatRoom.setEnabled(false);
    lstSendTo.setEnabled(false);
    btnLogon.setText("Log In");
    txtUsername.setEnabled(true);
    txtUsername.requestFocus();
    btnLogon.getRootPane().setDefaultButton(btnLogon);
    _lblStatus.setText("Welcome to MyChat - Please log in");
} catch (Exception e) {
    e.printStackTrace();
} finally {
}
```
/*
 * The btnSendMessage is used to broadcast a chat message line.
 */

private void
btnSendMessageActionPerformed(java.awt.event.ActionEvent evt)
{
    //This runs in the event-handling thread ("AWT-EventQueue-0")
    try {
        //We know the message won't be empty because the button is
disabled when that is the case
        HLAunicodeString us_cmdline = new
        HLAunicodeString(txtMessage.getText());
        //Note that there is no such thing as a ParameterHandleSet
        ParameterHandleValueMap _phvm_parameters =
        _rtiAmbassador.getParameterHandleValueMapFactory().create(2);
        _phvm_parameters.put(_iph_Communication_message,
        us_cmdline.toByteArray());
        _phvm_parameters.put(_iph_Communication_sender,
        _me.name.toByteArray());
        //The lstSendTo can be used to send a private message (to a
single user) instead of all users in the current chat room
        if (lstSendTo.getSelectedIndex() == 0)
        {
            //This'll become WithRegions to handle ChatRooms...
            _rtiAmbassador.sendInteraction(_ich_Communication,
            _phvm_parameters, null);
            _rtiAmbassador.sendInteractionWithRegions(_ich_Communication,
            _phvm_parameters, _rhs_current_ChatRoom, null);
            txtArea.append(_me.name.toString().substring(1) + ": " +
            us_cmdline + "\n");
            lblStatus.setText("Message sent");
        }
        else {
            //We'll use DDM to send the message to a single user
            (within the same chat room)
            //The Participant object has three attributes, all three
of which are bound
            //to the ChatRoomSlots dimension; in addition, user_handle
is bound to UserHandleSlots.

            //To allow user names and chat room names to be anything,
the latter will be prefixed
            //by "p" and "c", respectively, whilst our reserved names
will be prefixed by "*
            HLAunicodeString _us_TargetName = new HLAunicodeString("p" +
            lstSendTo.getSelectedItem().toString());
            //Although the lstSendTo cannot change on us (because
we're in the event-handling thread too),
            //the Participants could, so we must synchronize
synchronized(_theParticipants) {
    // us_TargetName is the name of a Participant object,
    // us_TargetName.toString()
    ObjectInstanceHandle key = _rtiAmbassador.getObjectInstanceHandle(us_TargetName.toString());
    aParticipant _Participant = SeekParticipantByName(us_TargetName.toString());
    if (_Participant == null) {
        javax.swing.JOptionPane.showMessageDialog(null, __TargetName.toString().substring(1) + " unexpectedly logged out. Message not sent.", federateHandle.toString(), javax.swing.JOptionPane.PLAIN_MESSAGE);
        return;
    }
    ObjectInstanceHandle key = _Participant.handle;
    long key_long = (long) key.hashCode();
    DimensionHandleSet dimensions = _rtiAmbassador.getAvailableDimensionsForInteractionClass(_ich_Communication);
    RegionHandle _rh_aParticipant = _rtiAmbassador.createRegion(_dsUserHandleSlotsSet);
    _rtiAmbassador.setRangeBounds(_rh_aParticipant, _dh_UserHandleSlots, new RangeBounds(key_long, key_long + 1));
    _rtiAmbassador.setRangeBounds(_rh_aParticipant, _dh_UserHandleSlots, new RangeBounds(_Participant.user_handle.getValue(), _Participant.user_handle.getValue() + 1));
    RegionHandleSet rhs_aParticipant = _rtiAmbassador.getRegionHandleSetFactory().create();
    rhs_aParticipant.add(_rh_aParticipant);
    _rtiAmbassador.commitRegionModifications(rhs_aParticipant);
    _rtiAmbassador.sendInteractionWithRegions(_ich_Communication, _phvm_parameters, rhs_aParticipant, null);
    rhs_aParticipant.remove(_rh_aParticipant);
    _rtiAmbassador.deleteRegion(_rh_aParticipant);
    synchronized(_theParticipants) {
        txtArea.append(__name.toString().substring(1) + " (to " + lstSendTo.getSelectedItem().toString() + ")": __cmdline + "\n"; lblStatus.setText("Private message sent");
    }
    catch (RTIexception e) {
        lblStatus.setText(e.getLocalizedMessage());
    }
    finally {
        //try{
        }catch (RTIexception e) {
            lblStatus.setText(e.getLocalizedMessage());
        }
    }
}
private aParticipant
SeekParticipant(ObjectInstanceHandle theObject)
{
  synchronized(_theParticipants)
  {
    return (aParticipant)_theParticipants.get(theObject);
    //synchronized(_theParticipants)
  }
}

private aChatRoom
SeekChatRoom(ObjectInstanceHandle theObject)
{
  synchronized(_theChatRooms)
  {
    return (aChatRoom)_theChatRooms.get(theObject);
    //synchronized(_theChatRooms)
  }
}

private aChatRoom
SeekChatRoomByName(String name)
{
  aChatRoom _ChatRoom;
  synchronized(_theChatRooms)
  {
    for (Iterator i = _theChatRooms.entrySet().iterator();
    i.hasNext());
    
    _ChatRoom =
    (aChatRoom)((java.util.Map.Entry)i.next()).getValue();
    if (_ChatRoom.name.toString().equals(name)) return
    _ChatRoom;
    } //for
    return null;
    } //synchronized(_theChatRooms)
}
/**
 * This utility method finds the ChatRoom object in the hashmap by its slot.
 * @param slot a short specifying the ChatRoom.slot sought
 * @return the sought ChatRoom object, or null if not present
 */
private aChatRoom
SeekChatRoomBySlot(short slot)
{
    aChatRoom _ChatRoom;
    synchronized(_theChatRooms)
    {
        for (Iterator i = _theChatRooms.entrySet().iterator(); i.hasNext();)
        {
            ChatRoom (aChatRoom) ( (java.util.Map.Entry) i.next() ) .getValue();
            if (_ChatRoom.slot.getValue() == slot) return _ChatRoom;
        } //for
        return null;
    } //synchronized(_theChatRooms)
}

/**
 * This utility method finds the Participant object in the hashmap by its name.
 * @param name a String specifying the Participant.name sought
 * @return the sought Participant object, or null if not present
 */
private aParticipant
SeekParticipantByName(String name)
{
    aParticipant _Participant;
    synchronized(_theparticipants)
    {
        for (Iterator i = _theparticipants.entrySet().iterator(); i.hasNext();)
        {
            Participant (aParticipant) ( (java.util.Map.Entry) i.next() ) .getValue();
            if (_Participant.name.toString().equals(name)) return _Participant;
        } //for
        return null;
    } //synchronized(_theparticipants)
}
/**
 * This utility method finds the Participant object in the hashmap by its user_handle.
 * @param user_handle an int specifying the Participant.user_handle sought
 * @return the sought Participant object, or null if not present
 */
private aParticipant
SeekParticipantByUserHandle(int user_handle)
{
    aParticipant _participant;
synchronized(_theParticipants)
    {
        for (Iterator i = _theParticipants.entrySet().iterator();
i.hasNext();)
        {
            _participant =
            (aParticipant)((java.util.Map.Entry)i.next()).getValue();
            if (_participant.user_handle.getValue() == user_handle)
            return _participant;
        } //for
        return null;
    } //synchronized(_theParticipants)
}

/**
 * This utility method provides the missing toByteArray method of ObjectInstanceHandle.
 * @param o the ObjectInstanceHandle to encode
 * @return the byte[] representation of the ObjectInstanceHandle
 * @throws CouldNotDecode if something goes wrong
 */
private byte[]
ObjectInstanceHandleToByteArray(ObjectInstanceHandle o)
throws CouldNotDecode
{
    byte[] b = new byte[o.encodedLength()];
o.encode(b, 0);
    return b;
}
/**
 * This utility method converts an ObjectInstanceHandle into an
 * integer (HLAInteger32BE).
 * This could break with later implementations of the RTI, should
 * it switch to 64-bit handles.
 * @param o the ObjectInstanceHandle to convert
 * @return the int representation of the ObjectInstanceHandle
 * @throws CouldNotDecode if something goes wrong
 */

/**
  * @param o the ObjectInstanceHandle to convert
  * @return the int representation of the ObjectInstanceHandle
  * @throws CouldNotDecode if something goes wrong
  */

private int
ObjectInstanceHandleToInt(ObjectInstanceHandle o)
throws CouldNotDecode
{
  //ObjectInstanceHandle has an encodedLength of 4
  final byte[] bytes = new byte[o(encodedLength());
  o.encode(bytes, 0);
  return new HLAinteger32BE(bytes).getValue();
}

/**
  * A simple class whose only purpose is to act as a semaphore
  * between Threads and carry a boolean payload (initially false).
  */

public class
Semaphore
// extends Object
{
  public boolean
  value = false;

private int
  count;
/**
 * Default constructor
 */

public
Semaphore()
{
  this(0);
}
/**
 * Constructs a Semaphore of the specified initial count.
 * @param count an int specifying the Semaphore's initial count
 */
public
Semaphore(int theCount)
{
    count = theCount;
}

/**
 * Whether the count has reached zero or not.
 * @return a boolean indicating whether the count has reached zero or not
 */
public boolean
zero()
{
    return (count == 0);
}

/**
 * Decreases the Semaphore's count by one.
 * The count cannot decrease below zero.
 */
public void
decreaseCount()
{
    decreaseCount(1);
}

/**
 * Decreases the Semaphore's count by the specified amount.
 * The count cannot decrease below zero.
 * @param amount an int to decrease the count by
 */
public void
decreaseCount(int amount)
{
    if (count > amount)
    {
        count -= amount;
    } else {
        count = 0;
    } //if
}

/**
 * Increases the Semaphore's count by one.
 */
public void
increaseCount()
{
    increaseCount(1);
}
/**
 * Increases the Semaphore's count by the specified amount.
 * @param amount an int to increase the count by
 */
public void
increaseCount(int amount)
{
    count += amount;
}

STM
II
FederateAmbassador implementation bits //
II
STM

// These are all called from the Federate Service Thread ("Federate
service thread")

// Name Reservation

/**
 * Used with ChatRoomRegistry and Participant.
 * Note that we do not admit a null Semaphore.
 */
public class
MyNameReservationListener
extends FedAmbNameReservationListener
{
    private Semaphore whichSemaphore;

    // Constructor
    public
    MyNameReservationListener(Semaphore aSemaphore)
    {
        whichSemaphore = aSemaphore;
    }

    protected void
doReservation(
        String objectName,
        boolean succeeded)
    throws UnknownName,
        FederateInternalError
    {
        synchronized(whichSemaphore)
        {
            whichSemaphore.value = succeeded;
            whichSemaphore.notify();
        }
    }

    // class MyNameReservationListener
// Interactions

public class MyCommunicationInteractionListener
    extends FedAmbInteractionListener
{
    protected void doInteraction(
        InteractionClassHandle interactionClass,
        ParameterHandleValueMap theParameters,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport,
        LogicalTime theTime,
        OrderType receivedOrdering,
        MessageRetractionHandle messageRetractionHandle,
        RegionHandleSet sentRegions)
        throws InteractionClassNotRecognized,
                InteractionParameterNotRecognized,
                InteractionClassNotSubscribed,
                InvalidLogicalTime,
                FederateInternalError
    {
        try {
            final HLAunicodeString _message = new HLAunicodeString();
            final HLAunicodeString _sender = new HLAunicodeString();
            for (Iterator i = theParameters.keySet().iterator();
                 i.hasNext(); )
            {
                ParameterHandle _parameterHandle =
                (ParameterHandle)i.next();
                if (_parameterHandle.equals(_iph_communication_message))
                {
                    _message.decode((byte[])theParameters.get(_parameterHandle));
                    // String((byte[])theParameters.get(_parameterHandle));
                } else if
                (_parameterHandle.equals(_iph_communication_sender))
                {
                    _sender.decode((byte[])theParameters.get(_parameterHandle));
                    // String((byte[])theParameters.get(_parameterHandle));
                } for
                SwingUtilities.invokeLater(new Runnable() { public void run()
                {
                    txtArea.append(_sender.toString().substring(1) + 
                    _message + "\n"); //Runnable
                } }); //Runnable
            } catch (CouldNotDecode e) {
                e.printStackTrace();
            } finally {
                //try
            } //method doInteraction
        } //class MyCommunicationInteractionListener
public class MyChatRoomRegistryDiscoveryListener
    extends FedAmbDiscoveryListener
{
    private Semaphore whichSemaphore;

    //Constructor
    public MyChatRoomRegistryDiscoveryListener(Semaphore theSemaphore)
    {
        whichSemaphore = theSemaphore;
    }

    public void discoverObjectInstance(
        ObjectInstanceHandle theObject,
        ObjectClassHandle theObjectClass,
        String objectName)
        throws CouldNotDiscover,
            ObjectClassNotRecognized,
            FederateInternalError
    {
        try {
            if (!theObjectClass.equals(_och_ChatRoomRegistry)) throw
                new ObjectClassNotRecognized("Unexpected object class");
            if (_ChatRoomRegistry.handle != null) throw new
                CouldNotDiscover("ChatRoomRegistry already discovered");
            if (!_ChatRoomRegistry.subscribed) throw new
                FederateInternalError("ChatRoomRegistry not subscribed");
            //Assuming AutoProvide, this'll be immediately followed by
            //an Update, so we cannot thread off
            _ChatRoomRegistry.handle = theObject;
            if (whichSemaphore == null) return;
            synchronized (whichSemaphore)
            {
                whichSemaphore.value = true;
                whichSemaphore.notify();
            } //synchronized (whichSemaphore)
            catch (Exception intercepted) {
                //try
                } //method discoverObjectInstance
        } //class MyChatRoomRegistryDiscoveryListener
public class MyChatRoomDiscoveryListener
extends FedAmbDiscoveryListener
{
    private Semaphore whichSemaphore;

    // Constructor
    public MyChatRoomDiscoveryListener(Semaphore theSemaphore)
    {
        whichSemaphore = theSemaphore;
    }

    public void discoverObjectInstance(
            ObjectInstanceHandle theObject,
            ObjectClassHandle theObjectClass,
            String objectName)
    throws CouldNotDiscover,
            ObjectClassNotRecognized,
            FederateInternalError
    {
        try {
            if (!theObjectClass.equals(_och_Room)) throw new ObjectClassNotRecognized("Unexpected object class");
            // Assuming AutoProvide, this'll be immediately followed by an Update, so we cannot thread off
            synchronized(_theChatRooms)
            {
                aChatRoom _ChatRoom;
                if (objectName.equals(_name_waiting_room_Room))
                {
                    _ChatRoom = _waiting_room_Room;
                }
                else {
                    _ChatRoom = new aChatRoom("unknown", (short)-1);
                }
                //if
                _ChatRoom.handle = theObject;
                _ChatRoom.subscribed = true;
                // ChatRoom name and slot will come from Update, we must wait until then before updating the GUI
                theChatRooms.put(theObject, _ChatRoom);
                if (whichSemaphore == null) return;
            } //synchronized(_theChatRooms)
            synchronized(whichSemaphore)
            {
                whichSemaphore.value = true;
                whichSemaphore.notify();
            } //synchronized(whichSemaphore)
        } catch (Exception intercepted) {
            finally {
                //try
                } //method discoverObjectInstance
        } //class MyChatRoomDiscoveryListener
public class MyChatRoomRemovalResponder
    extends FedAmbRemovalResponder
{

    protected void doRemove(
        ObjectInstanceHandle theObject,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        LogicalTime theTime,
        OrderType receivedOrdering,
        MessageRetractionHandle retractionHandle)
    throws ObjectInstanceNotKnown,
            InvalidLogicalTime,
            FederateInternalError
    {
        try {
            final aChatRoom _ChatRoom =
        (aChatRoom) theChatRooms.remove(theObject);
            if (_ChatRoom == null) throw new
        ObjectInstanceNotKnown("Unexpected object instance");
            if (_ChatRoom.name.toString().equals( name general ChatRoom)) return;
        SwingUtilities.invokeLater(new Runnable() { public void 
run() {
            //Strip leading character
            String theChatRoomName =
        _ChatRoom.name.toString().substring(1);
            lstChatRooms.removeItem(theChatRoomName);
        } }); //Runnable 
        } catch (Exception intercepted) {
        } finally {
        } //try 
    } //method doRemove
} //class MyChatRoomRemovalResponder

public class MyParticipantDiscoveryListener
    extends FedAmbDiscoveryListener
{

    private Semaphore
        whichSemaphore;
    private String
        whichName;

    //Constructor
    public
    MyParticipantDiscoveryListener(Semaphore aSemaphore, String
        anObjectName)
    { 
        whichSemaphore = aSemaphore;
        whichName = anObjectName;
    }
public void
discoverObjectInstance(
    ObjectInstanceHandle theObject,
    ObjectClassHandle theObjectClass,
    String objectName)
throws CouldNotDiscover,
    ObjectClassNotRecognized,
    FederateInternalError
{
    try {
        if (!theObjectClass.equals(_och_Participant)) throw new
            ObjectClassNotRecognized("Unexpected object class");

        // Because the discovery is immediately followed by the
        // firing of an InScope advisory, we cannot thread off
        // Maintain list of Participants (within the current
        ChatRoom)
        try {

synchronized(_theParticipants)
{
    aParticipant _Participant = new aParticipant();
    _Participant.subscribed = true;
    _Participant.inscope = false;  //Default, the InScope advisory will follow discovery in any case
    _Participant.handle = theObject;
    //
    _Participant.name.setValue(_rtiAmbassador.getObjectInstanceName(theObject));
    _Participant.name.setValue(objectName);
    if (_me_shutting_down) return;
    //We use the Auto-Provide switch, so an explicit Update request is not needed
    //We could add the discovered Participant's name to lstSendTo right away,
    //but we need to know in which ChatRoom slot it resides, so we must wait for the InScope advisory
    //because we subscribe to two regions at any given time, we can discover instances in either one)
    //Unfortunately, even with ConveyRegionDesignators set to true, this does not happen with the Discovery call
    if (whichSemaphore == null) return;
    //toString required because whichName is String but _Participant.name is HLAunicodeString
    if (!whichName.equals(_Participant.name.toString()))
        return;
    } //synchronized(_theParticipants)
} finally {
    //try
    synchronized(whichSemaphore)
    {
        whichSemaphore.value = true;
        whichSemaphore.notify();
    } //synchronized(whichSemaphore)
    catch (Exception intercepted) {
        finally {
            //try
            method discoverObjectInstance
        } //class MyParticipantDiscoveryListener
    } //Ownership

public class MyChatRoomRegistryOwnershipListener extends FedAmbOwnershipListener
{
    private Semaphore whichSemaphore;

    //Constructor
    public MyChatRoomRegistryOwnershipListener(Semaphore aSemaphore)
    {
        whichSemaphore = aSemaphore;
    }
public void requestAttributeOwnershipAssumption(
        ObjectInstanceHandle theObject,
        AttributeHandleSet offeredAttributes,
        byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeNotPublished,
        FederateInternalError
    {
        try {
            // Decline if we're in the process of shutting down (and
            // thus divesting)
            if (me.shutting_down()) return;
            // If the object is ChatRoomRegistry
            if (theObject.equals(ChatRoomRegistry.handle)) throw new
                ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
            if (offeredAttributes.containsAll(ChatRoomRegistry)) throw new
                AttributeNotRecognized("Unexpected attribute set offered");
            if (ChatRoomRegistry.owned) throw new
                AttributeAlreadyOwned("ChatRoomRegistry already owned");
            // Acquiesce if able
            // This condition is similar to !me.logged_in but better
            if (!ChatRoomRegistry.subscribed) return; // If not
                subscribed, probably not up to date -- decline
            // If not threaded off, this throws RTIInternalError:
            Concurrent access attempted to <method name>.
            new Thread() { public void run() {
                try {
                    // Note that the divesting federate will have offered
                    ownership to all federates, and that even if they all acquiesce,
                    // only one will receive an
                    ownershipAcquisitionNotification; if we use
                    attributeOwnershipAcquisition, there won't be any
                    // negative feedback if we fail to get ownership --
                    and we'll have an outstanding acquisition request.
                    // This is why it is preferable to use
                    attributeOwnershipAcquisitionIfAvailable
                    _rtiAmbassador.attributeOwnershipAcquisitionIfAvailable(ChatRoomRegis-
                        try.handle, oahs(ChatRoomRegistry));
                    catch(Exception ignored) {
                        finally { }
                        try {
                        }.start(); // Thread
                    } catch (Exception intercepted) {
                        finally { }
                        try {
                    } // Method requestAttributeOwnershipAssumption
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    FederateInternalError
{
    try {
        if (!theObject.equals(_ChatRoomRegistry.handle))
            throw new ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
        if (!offeredAttributes.containsAll(_oahs_chatRoomRegistry))
            throw new AttributeNotRecognized("Unexpected attribute set being divested");
        if (!_ChatRoomRegistry.owned)
            throw new AttributeNotOwned("ChatRoomRegistry not owned");

        // If not threaded off, this throws RTIinternalError:
        Concurrent access attempted to <method name>
        new Thread() { public void run() {
            synchronized(_ChatRoomRegistry) {

                _rtiAmbassador.confirmDivestiture(_ChatRoomRegistry.handle,
                _oahs_chatRoomRegistry, null);
                _ChatRoomRegistry.owned =
                _ChatRoomRegistry.divesting = false;
                } //synchronized(_ChatRoomRegistry)
                if (whichSemaphore == null) return;
                synchronized(whichSemaphore) {
                    whichSemaphore.value = true; // that would be
                    _sem_divestiture
                    whichSemaphore.notify();
                } //synchronized(whichSemaphore)
                } catch(Exception ignored) {
                    finally {
                        } //try
                        } //Thread
            } catch (Exception intercepted) {
                finally {
                    } //try
                } //method requestDivestitureConfirmation
public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAcquisitionWasNotRequested,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError
{
    try {
        if (!theObject.equals(_ChatRoomRegistry.handle)) throw new
            ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
        if (!securedAttributes.containsAll(_oahs(ChatRoomRegistry)) throw new
            AttributeNotRecognized("Unexpected attribute set secured");
        if (_ChatRoomRegistry.owned) throw new
            AttributeAlreadyOwned("ChatRoomRegistry already owned");
        if (_ChatRoomRegistry.subscribed) throw new
            AttributeAcquisitionWasNotRequested("ChatRoomRegistry not
            subscribed");
        //This completes the ownership transfer process
        synchronized(_ChatRoomRegistry)
        {
            _ChatRoomRegistry.owned = true ||
            (_ChatRoomRegistry.divesting = _me_shutting_down);
        } //synchronized(_ChatRoomRegistry)
        if (whichSemaphore != null)
        {
            synchronized(whichSemaphore)
            {
                whichSemaphore.value = true; //that would be
                _sem_acquisition
                whichSemaphore.notify();
            } //synchronized(whichSemaphore)
        } //if
        //Just in case we somehow managed to acquire whilst
        switching to shutting down mode
        if (_me_shutting_down)
        {
            new Thread() { public void run() {
                try {
                    _sem_acquisition
                    divestChatRoomRegistry();
                } catch (Exception ignored) {
                    } finally {
                    } //try
                } }.start(); //Thread
            } //if
        } catch (Exception intercepted) {
            } finally {
            } //try
    } ///method attributeOwnershipAcquisitionNotification
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateInternalError
{
    try {
        if (!theObject.equals(_ChatRoomRegistry.handle)) throw new
            ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
        if (!theAttributes.containsAll(_oahs_ChatRoomRegistry))
            throw new AttributeNotRecognized("Unexpected attribute set turned
down");
        if (_ChatRoomRegistry.owned) throw new
            AttributeAlreadyOwned("ChatRoomRegistry already owned");
        if (!_ChatRoomRegistry.subscribed) throw new
            AttributeAcquisitionWasNotRequested("ChatRoomRegistry not
subscribed");

        if (whichSemaphore == null) return;
        synchronized (whichSemaphore) {
            whichSemaphore.value = false; // that would be
            _sem_acquisition
            whichSemaphore.notify();
            } // synchronized (whichSemaphore)
        } catch (Exception intercepted) {
    } finally {
    } // try
} // method attributeOwnershipUnavailable

public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    try {
        if (!theObject.equals(_ChatRoomRegistry.handle)) throw new
            ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
        if (!candidateAttributes.containsAll(_oahs_ChatRoomRegistry))
            throw new AttributeNotRecognized("Unexpected attribute set requested");
        if (!_ChatRoomRegistry.owned) throw new
            AttributeNotOwned("ChatRoomRegistry not owned");
        // if (!_ChatRoomRegistry.subscribed) throw new
            FederateInternalError("ChatRoomRegistry not subscribed");
        // Acquiesce
// If not threaded off, this throws RTIInternalError:
Concurrent access attempted to <method name>
new Thread() { public void run() {
  try {
    // If we're currently busy with the ChatRoomRegistry,
    // we'll be synchronized on it and this thread will
    // have to wait
    synchronized(_ChatRoomRegistry)
    {
      if (!oahs_ChatRoomRegistry.containsAll(_rtiAmbassador.attributeOwnership
Di vestitureIfWanted(_ChatRoomRegistry.handle,
_oahs_ChatRoomRegistry))) return;
      _ChatRoomRegistry.owned =
      _ChatRoomRegistry.divesting = false;
      } //synchronized(_ChatRoomRegistry)
      if (whichSemaphore == null) return;
      synchronized(whichSemaphore)
      {
        whichSemaphore.value = true; // that would be
        _sem_divestiture whichSemaphore.notify();
      } //synchronized(whichSemaphore)
    } catch (Exception ignored) {
    } finally {
    } //try
  } }.start(); // Thread
} catch (Exception intercepted) {
} finally {
} // try
} // method requestAttributeOwnershipRelease
public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeAlreadyOwned,
            AttributeAcquisitionWasNotCanceled,
            FederateInternalError
{
    if (_ChatRoomRegistry.handle != theObject) throw new ObjectInstanceNotKnown("Unknown ChatRoomRegistry offered");
    if (!_theAttributes.containsAll(_oahs_ChatRoomRegistry)) throw new AttributeNotRecognized("Unexpected attribute set cancelled");
    if (_ChatRoomRegistry.owned) throw new AttributeAlreadyOwned("ChatRoomRegistry already owned");
    throw new AttributeAcquisitionWasNotCanceled("I don't think so...");
} //method confirmAttributeOwnershipAcquisitionCancellation

/**
 * public void
doInformOwnership(
    final ObjectInstanceHandle theObject,
    final AttributeHandle theAttribute,
    final boolean isUnowned,
    final boolean isOwnedByRTI,
    final FederateHandle theOwner)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            FederateInternalError
{
} //method doInformOwnership
*/
} //class MyChatRoomRegistryOwnershipListener

public class MyChatRoomRegistryOwnershipListener
    extends FedAmbOwnershipListener
{
    private Semaphore whichSemaphore;
    private ObjectInstanceHandle whichObject;
    //The Semaphore and ObjectInstanceHandle serve two purposes:
    // - In Acquisition mode, Semaphore.value will be set once an
      acquisitionNotification or Unavailable callback is received for the
    // - In Divestiture mode, each requestDivestitureConfirmation
      and requestAttributeOwnershipRelease results in a Semaphore.decreaseCount; once Semaphore.zero() is true, it is notified.
// Constructor
public
MyChatRoomOwnershipListener(Semaphore aSemaphore,
ObjectInstanceHandle anObject)
{
    whichSemaphore = aSemaphore;
    whichObject   = anObject;
}
public void requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeNotPublished,
        FederateInternalError
{
    try {
        //Decline if we're in the process of shutting down (and
        //thus divesting)
        if (_me_shutting_down) return;
        final aChatRoom _ChatRoom = SeekChatRoom(theObject);
        if (_ChatRoom == null) throw new
            ObjectInstanceNotKnown("Unknown ChatRoom offered");
        if (!offeredAttributes.containsAll(_oahs_ChatRoom)) throw
            new AttributeNotRecognized("Unexpected attribute set offered");
        if (_ChatRoom.owned) throw new
            AttributeAlreadyOwned("ChatRoom already owned");
        //Also decline if:
        //We're not subscribed, thus the values are probably not
        //up to date; or
        //We're logged out and the chatRoom slot is other than the
        //waiting_room
        if (!_ChatRoom.subscribed) || (!_me_logged_in) &&
            (_ChatRoom.slot.getValue() != _slot_waiting_room_ChatRoom)) return;
        //Either we're logged in,const the chat room is the
        //waiting room;
        //If not the waiting room, accept only if its our current
        //chat room
        if (_ChatRoom.slot.getValue() !=
            _slot_waiting_room_ChatRoom) &&
            (!_ChatRoom.slot.equals(_myChatRoom.slot))) return;
        //If not threaded off, this throws RTIinternalError:
        Concurrent access attempted to <method name>
        new Thread() { public void run() { try {
            //Accept only if subscribed (and presumably up to
            //date)
            _rtiAmbassador.attributeOwnershipAcquisitionAvailable(_ChatRoom.hand
            le, _oahs_ChatRoom); //or theObject
            } catch (Exception ignored) { 
                finally {
                } //try
            } }.start();  //Thread
            } catch (Exception intercepted) { 
                finally {
                } //try
            } //method requestAttributeOwnershipAssumption
}
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        AttributeDivestitureWasNotRequested,
        FederateInternalError {
    try {
        final aChatRoom ChatRoom = SeekChatRoom(theObject);
        if (ChatRoom == null) throw new ObjectInstanceNotKnown("Unknown ChatRoom offered");
        if (!offeredAttributes.containsAll(_oahs_ChatRoom)) throw new AttributeNotRecognized("Unexpected attribute set requested");
        if (!_ChatRoom.owned) throw new AttributeNotOwned("ChatRoom not owned");

        // If not threaded off, this throws RTIinternalError: Concurrent access attempted to <method name>
        new Thread() {
            public void run() {
                // Confirm
                synchronized(_ChatRoom) {
                    _rtiAmbassador.confirmDivestiture(_ChatRoom.handle, _oahs_ChatRoom, null);
                    _ChatRoom.owned = _ChatRoom.divesting = false;
                }
                if (whichSemaphore == null) return;
                synchronized(whichSemaphore) {
                    whichSemaphore.decreaseCount();
                    if (whichSemaphore.zero()) {
                        whichSemaphore.notify();
                    }
                } // synchronized(whichSemaphore)
                catch (Exception ignored) {
                    finally {
                        // try
                    }
                }.start(); // Thread
            } catch (Exception intercepted) {
                finally {
                    // try
                }
            } // method requestDivestitureConfirmation
public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAcquisitionWasNotRequested,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError
{
    try {
        final aChatRoom _ChatRoom = SeekChatRoom(theObject);
        if (_ChatRoom == null) throw new ObjectInstanceNotKnown("Unknown ChatRoom offered");
        if (!securedAttributes.containsAll(_oahs_ChatRoom)) throw new AttributeNotRecognized("Unexpected attribute set acquired");
        if (!_ChatRoom.owned) throw new AttributeAlreadyOwned("ChatRoom already owned");
        synchronized(_ChatRoom) {
            _ChatRoom.owned = !_ChatRoom.divesting;
            if (whichSemaphore != null) {
                synchronized(whichSemaphore) {
                    whichSemaphore.value = (_ChatRoom.handle.equals(whichObject));
                    if (whichSemaphore.value) whichSemaphore.notify();
                } //synchronized(whichSemaphore)
            } //if
            //Just in case we somehow managed to acquire whilst switching to shutting down mode
            if (!_me_shutting_down) {
                new Thread() { public void run() { try {
                    divestChatRooms();
                } catch (Exception ignored) { } finally { }
                } //try
                }.start(); //Thread
            } //if
        }
    } catch (Exception intercepted) { }
} finally { }
//method attributeOwnershipAcquisitionNotification
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeAlreadyOwned,
            AttributeAcquisitionWasNotRequested,
            FederateInternalError
{
    try {
        final aChatRoom _chatRoom = SeekChatRoom(theObject);
        if (_chatRoom == null) throw new ObjectInstanceNotKnown("Unknown ChatRoom offered");
        if (!theAttributes.containsAll(_chatRoom)) throw new AttributeNotRecognized("Unexpected attribute set denied");
        if (_chatRoom.owned) throw new AttributeAlreadyOwned("ChatRoom already owned");
        if (whichSemaphore == null) return;
        synchronized (whichSemaphore) {
            whichSemaphore.value = false;
            if (_chatRoom.handle.equals(whichObject))
                whichSemaphore.notify();
        } //synchronized (whichSemaphore)
    } catch (Exception intercepted) {
    } finally {
    } //try
} //method attributeOwnershipUnavailable
public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederatelnternalError
{
    try {
        final aChatRoom _ChatRoom = SeekChatRoom(theObject);
        if (_ChatRoom == null) throw new
ObjectInstanceNotKnown("Unknown ChatRoom offered");
        if (!candidateAttributes.containsAll(_oahs_ChatRoom))
throw new AttributeNotRecognized("Unexpected attribute set
requested");
        if (!_ChatRoom.owned) throw new
AttributeNotOwned("ChatRoom not owned");
        // Acquiesce

        // If not threaded off, this throws RTIinternalError:
        ConcurrentHashMap ConcurrentHashMap = ConcurrentHashMap
        new Thread() { public void run() {
            try {
                if (_oahs_ChatRoom.containsAll(_rtiAmbassador.attributeOwnershipDivestitureIfWanted(_ChatRoom.handle, _oahs_ChatRoom))) return;
                // Success:
                synchronized(_ChatRoom)
                { _ChatRoom.owned = _ChatRoom.divesting = false;
                } // synchronized(_ChatRoom)
                if (whichSemaphore == null) return;
                synchronized(whichSemaphore)
                { whichSemaphore.decreaseCount();
                    if (whichSemaphore.zero()) {
                        whichSemaphore.notify();
                    }
                } // synchronized(whichSemaphore)
            } catch (RTIexception ignored) {
            } finally {
            } // try
        }.start(); // Thread
    } catch (Exception intercepted) {
    } finally {
    } // try
} // method requestAttributeOwnershipRelease
public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError
[
    final aChatRoom _ChatRoom = SeekChatRoom(theObject);
    if (_ChatRoom == null) throw new ObjectInstanceNotKnown("Unknown ChatRoom offered");
    if (!theAttributes.containsAll(_oahs_ChatRoom)) throw new AttributeNotRecognized("Unexpected attribute set cancelled");
    if (_ChatRoom.owned) throw new AttributeAlreadyOwned("ChatRoom already owned");
    throw new AttributeAcquisitionWasNotCanceled("I don't think so...");
} //method confirmAttributeOwnershipAcquisitionCancellation
/**
public void doInformOwnership(
    final ObjectInstanceHandle theObject,
    final AttributeHandle theAttribute,
    final boolean isUnowned,
    final boolean isOwnedByRTI,
    final FederateHandle theOwner)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    FederateInternalError
[
    } //method doInformOwnership
*/
] //class MyChatRoomOwnershipListener
public class MyParticipantOwnershipListener extends FedAmbOwnershipListener {
    private Semaphore whichSemaphore;
    private ObjectInstanceHandle whichObject;
    // The Semaphore and ObjectInstanceHandle serve two purposes:
    // - In Acquisition mode, Semaphore.value will be set once an
      acquisitionNotification or
      Unavailable callback is received for the
      ObjectInstanceHandle, and then the Semaphore is notified.
    // - In Divestiture mode, each requestDivestitureConfirmation
      and requestAttributeOwnershipRelease
      results in a Semaphore.decreaseCount; once Semaphore.zero() is true, it is notified.

    // Constructor
    public MyParticipantOwnershipListener(Semaphore aSemaphore, ObjectInstanceHandle anObject) {
        whichSemaphore = aSemaphore;
        whichObject = anObject;
    } // method MyParticipantOwnershipListener
public void requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError
{
    try {
        //Decline if we're in the process of shutting down (and
        //thus divesting)
        if (me.shutting_down) return;
        final aParticipant _Participant = SeekParticipant(theObject);
        if (_Participant == null) throw new ObjectInstanceNotKnown("Unknown Participant offered");
        if (!offeredAttributes.containsAll(_oahs_Participant))
            throw new AttributeNotRecognized("Unexpected attribute set offered");
        if (_Participant.owned) throw new AttributeAlreadyOwned("Participant already owned");
        //Accept only if subscribed (and presumably up to date)
        //The inscope attribute is similar, but subscribed will do
        just fine
        if (!_Participant.subscribed) return; //If not subscribed, probably not up to date --decline

        //If not threaded off, this throws RTIInternalError:
        Concurrent access attempted to <method name>
        new Thread() { public void run() {
            try {
                //Before acquiring, set up update regions so they do
                //not lapse during ownership transfer
                synchronized(_Participant) {
                    _rtiAmbassador.associateRegionsForUpdates(_Participant.handle,
                        _asrspl_Participant_nowhere);
                    _rtiAmbassador.associateRegionsForUpdates(_Participant.handle,
                        _asrspl_Participant_waiting_room);
                    //synchronized(_Participant)
                    _rtiAmbassador.attributeOwnershipAcquisitionIfAvailable(_Participant.handle,
                        _oahs_Participant); //or theObject
                } catch(Exception ignored) {
                    finally {
                        //try
                        } //finally
                    } //try
                } //Thread
            } catch (Exception intercepted) {
                finally {
                    } //try
                } //method requestAttributeOwnershipAssumption

} //AttributeOwnershipAssumption
public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    FederateInternalError
{
    try {
        final aParticipant _Participant = SeekParticipant(theObject);
        if (_Participant == null) throw new ObjectInstanceNotKnown("Unknown Participant offered");
        if (!offeredAttributes.containsAll(_oahs_Participant)) throw new AttributeNotRecognized("Unexpected attribute set requested");
        if (!_Participant.owned) throw new AttributeNotOwned("Participant not owned");

        //If not threaded off, this throws RTIinternalError:
        Concurrent access attempted to <method name>
        new Thread() { public void run() { MyParticipantOwnershipListener: requestDivestitureConfirmation thread begins"});

        try {
            synchronized(_Participant) {

                _rtiAmbassador.confirmDivestiture(_Participant.handle,
                _oahs_Participant, null);
                _Participant.owned = _Participant.divesting =
                false;
            } //synchronized(_Participant)
            if (whichSemaphore == null) return;
            synchronized(whichSemaphore) {
                whichSemaphore.decreaseCount();
                if (whichSemaphore.zero()) {
                    whichSemaphore.notify();
                } //synchronized(whichSemaphore)
            } catch(Exception ignored) {
            } finally {
        } //try
        } start(); //Thread
    } catch (Exception intercepted) {
    } finally {
        } //try
    } //method requestDivestitureConfirmation
public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
AttributeNotRecognized,
AttributeAcquisitionWasNotRequested,
AttributeAlreadyOwned,
AttributeNotPublished,
FederateInternalError
{
    try {
        final aParticipant _Participant = SeekParticipant(theObject);
        if (_Participant == null) throw new ObjectInstanceNotKnown("Unknown Participant offered");
        if (!securedAttributes.containsAll(_ohs_Participant))
            throw new AttributeNotRecognized("Unexpected attribute set acquired");
        if (_Participant.owned) throw new AttributeAlreadyOwned("Participant already owned");
        //Must switch these right away, otherwise a Federate Responder could be called before the thread has time to launch
        _Participant.owned = !_Participant.inscope = _Participant.divesting = false;
        //There are two circumstances where we acquire Participants:
        //When we want to log in, and when another federate logs out and shuts down.
        //In both cases we set up the PortUpdate Regions such that the Participant
        //arrives associated with _nowhere and _waiting_room.
        //In the first circumstance, the plnHLA_Participant_NameReservation waits on the
        //sem_Participant_acquisition and will move the Participant to the _general ChatRoom.
        if (whichSemaphore != null) {
            synchronized (whichSemaphore) {
                whichSemaphore.value = _Participant.handle.equals(whichObject));
                if (whichSemaphore.value) whichSemaphore.notify();
            } //synchronized (whichSemaphore)
        } //if
        //Just in case we somehow managed to acquire whilst switching to shutting down mode
        if (!_me_shutting_down) return;
        //If not threaded off, this throws RTIInternalError:
        Concurrent access attempted to <method name>
        new Thread() { public void run() {
            divestParticipants();
        }.start(); //Thread
    } catch (Exception intercepted) {
    } finally {
    } //try
} //method attributeOwnershipAcquisitionNotification
public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateInternalError
{
    try {
        final aParticipant _Participant =
            SeekParticipant(theObject);
        if (_Participant == null) throw new
            ObjectInstanceNotKnown("Unknown Participant offered");
        if (!theAttributes.containsAll(_oahs_Participant)) throw
            new AttributeNotRecognized("Unexpected attribute set denied");
        if (_Participant.owned) throw new
            AttributeAlreadyOwned("Participant already owned");
        if (whichSemaphore == null) return;
        synchronized (whichSemaphore) {
            whichSemaphore.value = false;
            if (_Participant.handle.equals(whichObject))
                whichSemaphore.notify();
        } //synchronized (whichSemaphore)
    } catch (Exception intercepted) {
        finally {
            //try
        } //try
    } //method attributeOwnershipUnavailable
public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
  throws ObjectInstanceNotKnown,
          AttributeNotRecognized,
          AttributeNotOwned,
          FederateInternalError
{
  try {
    final aParticipant _Participant =
        SeekParticipant(theObject);
    if (_Participant == null) throw new
        ObjectInstanceNotKnown("Unknown Participant offered");
    if (!candidateAttributes.containsAll(_oahs_Participant))
        throw new AttributeNotRecognized("Unexpected attribute set
requested");
    if (!_Participant.owned) throw new
        AttributeNotOwned("Participant not owned");
    //If request targets the logged-in _me, refuse (there is
    //no RTI call to tell the requester to piss off)
    //Should be just if (_Participant.equals(_me)) since _me
    //is just a placeholder
    //Participant object (not in the HLA fed) when the
    federate is not logged in
    if (_me_logged_in && _Participant.equals(_me))
      return;
  } //if
//Otherwise, acquiesce
//If not threaded off, this throws RTIinternalError:
Concurrent access attempted to <method name>
new Thread() { public void run() {
  try {
    if
    (!_oahs_Participant.containsAll(_rtiAmbassador.attributeOwnershipDives
titureIfWanted(_Participant.handle, _oahs_Participant))) return;
    //Success:
    synchronized(_Participant)
      {
      _Participant.owned = _Participant.divesting =
      _Participant.inscope = false;
      //Region associations are automatically lost
      along with ownership
      } //synchronized(_Participant)
      if (whichSemaphore == null) return;
      synchronized(whichSemaphore)
      {
      whichSemaphore.decreaseCount();
      if (whichSemaphore.zero()) {
      whichSemaphore.notify();
      } //synchronized(whichSemaphore)
      } catch (RTIexception ignored) {
      } finally {
      } //try
      } }.start(); //Thread
} catch (Exception intercepted) {
} finally {
} //try
} //method requestAttributeOwnershipRelease
public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotCanceled,
    FederateInternalError
{
    try {
        final aParticipant _Participant = SeekParticipant(theObject);
        if (_Participant == null) throw new
            ObjectInstanceNotKnown("Unknown Participant offered");
        if (!theAttributes.containsAll(_oahs_Participant)) throw
            new AttributeNotRecognized("Unexpected attribute set cancelled");
        if (_Participant.owned) throw new
            AttributeAlreadyOwned("Participant already owned");
        throw new AttributeAcquisitionWasNotCanceled("I don't think so...");
    } catch (Exception intercepted) {
    } finally {
    } //try
} //method confirmAttributeOwnershipAcquisitionCancellation
/**
 * //method doInformOwnership
 */ //class MyParticipantOwnershipListener
//Interaction Scope Advisories
public class MyCommunicationAdvisoryResponder
    extends FedAmbInteractionAdvisoryResponder
{
    protected void doRelevance(
        InteractionClassHandle theHandle,
        boolean relevant)
throws InteractionClassNotPublished,
    FederateInternalError
    {
        _alone = !relevant;
    } //method doRelevance
} //class MyCommunicationAdvisoryResponder
public class MyParticipantAttributeScopeListener extends FedAmbAttributeScopeAdvisoryListener {
    /**
     * Default constructor in use.
     */
    protected void doScope(
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes,
        boolean inScope)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotSubscribed,
                FederateInternalError
    {
        try {
            //The Participant has just come into or gone out of scope
            //either we or it has changed ChatRooms (it cannot have been truly
            //deleted)
            final aParticipant _Participant = SeekParticipant(theObject);
            if (_Participant == null) throw new ObjectInstanceNotKnown("Unexpected Participant handle");
            if (!theAttributes.containsAll(_oahs_Partner_forUpdate)) throw new AttributeNotRecognized("Unexpected attribute set provided");
            if (!_Participant.subscribed) throw new AttributeNotSubscribed("Participant not subscribed");
            //If we could localDelete Participants as they go out of
            //scope, we would always get
            //a Discovery right before the InScope callback; but we
            //can't do that because of the
            //pRTI RTIambassador-Federate Service Thread synch bug
            //putting the Federate Service
            //Thread in any kind of synch-wait state (a Semaphore,
            //say) causes a federate-triggering
            //RTIambassador call by *any* thread to hang, even if the
            //latter RTIambassador-caller
            //isn't owning any monitors).
            //So we use the inScope property of aParticipant to work
            //around it.
            //Note that we get advisories even for owned objects
            _Participant.inscope = inScope;
            //Strip leading character
            final String theParticipantName = _Participant.name.toString().substring(1);
if (inScope)
{
    //Freshly discovered object, or object that went out of
    //and back in scope
    _Participant.subscribed, handle and name are set
    (owned, divesting at default);
    //logged_in and chat_room_slot will have to await the
    //update listener
    //A newly discovered _Participant will have a
    chat_room_slot of slot_nowhere_ChatRoom,
    //which is not possible once we get at least one update
    (because no federate ever
    //subscribes to the nowhere ChatRoom)
    //Maintain list of Participants (within the current
    ChatRoom)
    final boolean mustAdd = _me.logged_in &
    _Participant.chat_room_slot.equals(_me.chat_room_slot);
    if (mustAdd)
    {
        SwingUtilities.invokeLater(new Runnable() {
            public void run()
            {
                try {
                    //Note that we skip the <All> item
                    for (int i = 1; i < lstSendTo.getItemCount();
                        i++)
                    {
                        if
                        { //The Participant is already listed
                            return;
                        } else if
                        { //Previously unknown Participant, who
                            sorts before the current one (i)
                            lstSendTo.insertItemAt(theParticipantName, i);
                            lblStatus.setText(theParticipantName + " has joined.");
                            return;
                        } //if
                        } //for
                    } //If we make it through the list, must append
                    the previously unknown Participant
                    lstSendTo.insertItemAt(theParticipantName,
                    lstSendTo.getItemCount());
                    lblStatus.setText(theParticipantName + " has
                    joined.");
                    } finally {
                        } //try
                    } }); //Runnable
                } else {
                    } //if
            } //try
        } }); //Runnable
    } else {
    } //if
else {
    // Maintain list of Participants (within the current ChatRoom)
    // Because the _Participant will come back into scope *before* we get an update,
    // we'll mark it as "nowhere" so it behaves as a newly discovered one should
    // it come back into scope
    final boolean mustAdvise = _me_logged_in &&
    _Participant.chat_room_slot.equals(_me.chat_room_slot);

    _Participant.chat_room_slot.setValue(_slot_nowhere_ChatRoom);
    SwingUtilities.invokeLater(new Runnable() { public void run() {
        if (mustAdvise) lblStatus.setText(theParticipantName + " has left.");
        lstSendTo.removeItem(theParticipantName);
    } }); // Runnable

    // What we wanted to do here was localDelete the _Participant that just went out of scope.
    // We would have had to ask another thread to do this RTIambassador call, and we would have
    // had to wait for that other thread to conclude, otherwise the _Participant could come back
    // into scope (and trigger this listener again) before the other thread had time to localDelete,
    // leading to a situation where the RTI thinks we know the _Participant but we've
    // already stricken it off our list...
    // However, the pRTI RTIambassador-Federate Service Thread synch bug is such that if we
    // put the Federate Service Thread in any kind of synch-wait state (a Semaphore, say),
    // a federate-triggering RTIambassador call by *any* thread hangs, even if the latter
    // RTIambassador-caller isn't owning any monitors.
    // Conclusion: We cannot localDelete at all.
    }
  }

}
public class MyChatRoomRegistryInstanceAttributeListener extends FedAmbInstanceAttributeListener {
    protected void doUpdate(
        ObjectInstanceHandle theObject,
        AttributeHandleValueMap theAttributes,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport,
        LogicalTime theTime,
        OrderType receivedOrdering,
        MessageRetractionHandle retractionHandle,
        RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotSubscribed,
        InvalidLogicalTime,
        FederateInternalError {
        try {
            if (!theObject.equals(_ChatRoomRegistry.handle)) throw new ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
            if (!theAttributes.keySet().containsAll(_oahs_ChatRoomRegistry_forUpdate)) throw new AttributeNotRecognized("Unexpected attribute set provided");
            if (_ChatRoomRegistry.owned) throw new FederateInternalError("ChatRoomRegistry owned");
            if (_ChatRoomRegistry.subscribed) throw new AttributeNotSubscribed("ChatRoomRegistry not subscribed");

            //HLAprivilegeToDeleteObject has no content, so we won't decode it

            _ChatRoomRegistry.list.decode((byte[]) theAttributes.get(_oah_ChatRoomRegistry_list));
        }
        catch (RTIexception intercepted) {
            finally {
                //try
                //method doUpdate
            } //class MyChatRoomRegistryInstanceAttributeListener
public class MyChatRoomInstanceAttributeListener
extends FedAmbInstanceAttributeListener
{
    protected void
    doUpdate(
        ObjectInstanceHandle theObject,
        AttributeHandleValueMap theAttributes,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport,
        LogicalTime theTime,
        OrderType receivedOrdering,
        MessageRetractionHandle retractionHandle,
        RegionHandleSet sendRegions)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotSubscribed,
                InvalidLogicalTime,
                FederateInternalError
    {
        try {
            final aChatRoom _chatRoom = SeekChatRoom(theObject);
            if (_chatRoom == null) throw new ObjectInstanceNotKnown("Unknown chatRoom supplied");

            if (!_chatRoom.subscribed) throw new AttributeNotSubscribed("ChatRoom not subscribed");
            synchronized(_chatRoom) {
                // Name is a true attribute, not the object's HLA name
                _chatRoom.name.decode((byte[])theAttributes.get(_oah_chatRoom_name));
                _chatRoom.slot.decode((byte[])theAttributes.get(_oah_chatRoom_slot));
                // Alternately, we could load the IstChatRooms at log-in time...
                if ((_chatRoom.name.toString().equals(_name_general_chatRoom)) ||
                    (_chatRoom.name.toString().equals(_name_waiting_room_chatRoom)))
                    return;

                final String theChatRoomName =
                    _chatRoom.name.toString().substring(1);

            }
        } catch (Exception e) {
            // Handle exceptions
        }
    }
}
SwingUtilities.invokeLater(new Runnable() { public void run() {
    // Note that we skip the <General> item
    for (int i = 1; i < lstChatRooms.getItemCount(); i++)
    {
        // If already listed, done
        if (theChatRoomName.compareTo((String)lstChatRooms.getItemAt(i)) == 0)
            return;
        // Point of insertion found
        if (theChatRoomName.compareTo((String)lstChatRooms.getItemAt(i)) < 0)
        {
            lstChatRooms.insertItemAt(theChatRoomName, i);
            return;
        } // if
    } // for
    // List scanned without finding insertion point:
    append
    lstChatRooms.insertItemAt(theChatRoomName,
    lstChatRooms.getItemCount());
} }); // Runnable
} // synchronized(_ChatRoom)
} catch (RTIexception intercepted) {
} finally {
} // try
} // method doUpdate
} // class MyChatRoomInstanceAttributeListener
public class MyParticipantInstanceAttributeListener
    extends FedAmbInstanceAttributeListener
{
    protected void doUpdate(
        ObjectInstanceHandle theObject,
        AttributeHandleValueMap theAttributes,
        byte[] userSuppliedTag,
        OrderType sentOrdering,
        TransportationType theTransport,
        LogicalTime theTime,
        OrderType receivedOrdering,
        MessageRetractionHandle retractionHandle,
        RegionHandleSet sentRegions)
            throws ObjectInstanceNotKnown,
                        AttributeNotRecognized,
                        AttributeNotSubscribed,
                        InvalidLogicalTime,
                        FederateInternalError
    {
        try {
            final aParticipant _Participant = SeekParticipant(theObject);
            if (_Participant == null) throw new ObjectInstanceNotKnown("Unknown Participant supplied");
            if (!theAttributes.keySet().containsAll(_cahs_Particlent_forUpdate))
                throw new AttributeNotRecognized("Unexpected attribute set supplied");
            if (!_Participant.subscribed) throw new AttributeNotSubscribed("Participant not subscribed");
                //Does this need to be threaded off?
            synchronized(_Participant)
            {
        }
//Name updated at Discovery time

_participant.name.setValue(_rtiAmbassador.getObjectInstanceName(_theObject));
_participant.logged_in.decode((byte[])theAttributes.get(_oahParticipant_logged_in));
_participant.user_handle.decode((byte[])theAttributes.get(_oahParticipant_user_handle));
_participant.chat_room_slot.decode((byte[])theAttributes.get(_oahParticipant_chat_room_slot));

//Ideally, it would make more sense to maintain
1stSendTo on the Scope Advisory callbacks;
//however, when those first occur, we do not know the
Participant’s attributes yet, so we can’t
//tell between those in the "waiting_room" and those in
our ChatRoom.

//When a Participant logs out (from our ChatRoom), it
moves from a given ChatRoom to the
"waiting_room" by first withdrawing to "nowhere", so
it will trigger Scope Advisories
//twice (since we subscribe to "waiting_room" and our
current ChatRoom at all times).
//
// if (_me_logged_in &&
// _participant.chat_room_slot.getValue() != 0)
// if (!_me_logged_in) return;

final String theParticipantName =
_participant.name.toString().substring(1);
//When logged out, _me.chat_room_slot would be
waiting_room, so we don’t want to add those
final boolean mustAdd = _me_logged_in &&
_participant.chat_room_slot.equals(_me.chat_room_slot);
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        try {
            // Note that we skip the <All> item
            for (int i = 1; i < lstSendTo.getItemCount(); i++)
            {
                if (theParticipantName.compareTo(lstSendTo.getItemAt(i)) == 0)
                {
                    // The participant is already listed
                    if (!mustAdd) return;
                    lstSendTo.remove(i);
                    if (_me_logged_in)
                    {
                        lblStatus.setText(theParticipantName + " has left.");
                        return;
                    }
                } else if (theParticipantName.compareTo(lstSendTo.getItemAt(i)) < 0)
                {
                    // Previously unknown participant, who sorts before the current one (i)
                    if (!mustAdd) return;
                    lstSendTo.insertItemAt(theParticipantName, i);
                    lblStatus.setText(theParticipantName + " has joined.");
                    return;
                }
            }
        } finally {
            try {
                // synchronized (_Participant)
                catch (RTIException intercepted) {
                } finally {
                    // method doUpdate
                }
            } // Runnable
        } // try
    } // run
}); // Runnable
} // class MyParticipantInstanceAttributeListener
public class MyChatRoomRegistryInstanceAttributeResponder
    extends FedAmbInstanceAttributeResponder
{
    public void provideAttributeValueUpdate(
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes,
        byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotOwned,
            FederateInternalError
    {
        try {
            if (!theObject.equals(_ChatRoomRegistry.handle)) throw new
                ObjectInstanceNotKnown("Unexpected ChatRoomRegistry handle");
            if (!theAttributes.containsAll(_oahs_ChartRoomRegistry_forUpdate)) throw new
                AttributeNotRecognized("Unexpected attribute set requested");
            if (!_ChatRoomRegistry.owned) throw new
                AttributeNotOwned("ChatRoomRegistry not owned");
            if (!_ChatRoomRegistry.subscribed) throw new
                FederateInternalError("ChatRoomRegistry not subscribed");
        //If not threaded off, this throws RTIinternalError:
            Concurrent access attempted to <method name>
            new Thread() { public void run() {
                try {
                    synchronized(_ChatRoomRegistry)
                    {
                        final AttributeHandleValueMap
                          _ahvm_attributeValues =
                            _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs_ChartRoomRegistry_forUpdate.size());
                        //
                        _ahvm_attributeValues.put(_oah_ChartRoomRegistry_DeletePrivilege,
                          null); //HLAprivilegeToDeleteObject is of undefined type (it has no content)
                        _ahvm_attributeValues.put(_oah_ChartRoomRegistry_list,
                          _ChatRoomRegistry.list.toByteArray());
                        _rtiAmbassador.updateAttributeValues(_ChatRoomRegistry.handle,
                          _ahvm_attributeValues, null);
                    } //synchronized(_ChatRoomRegistry)
                    } catch (Exception ignored) {
                    } finally {
                        //Try
                        _ChatRoomRegistry.start(); //Thread
                    } catch (Exception intercepted) {
                    } finally {
                        //Try
                        //method provideAttributeValueUpdate
                    } //class MyChatRoomRegistryInstanceAttributeResponder
public class MyChatRoomInstanceAttributeResponder extends FedAmbInstanceAttributeResponder
{
    public void provideAttributeValueUpdate(
            ObjectInstanceHandle theObject,
            AttributeHandleSet theAttributes,
            byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
                AttributeNotRecognized,
                AttributeNotOwned,
                FederateInternalError
    {
        try{
            final aChatRoom _ChatRoom = SeekChatRoom(theObject);
            if (_ChatRoom == null) throw new ObjectInstanceNotKnown("ChatRoom instance not known");
            if (!theAttributes.containsAll(_oahs_CHATRoom_forUpdate))
                throw new AttributeNotRecognized("Unexpected attribute set requested");
            if (!_ChatRoom.owned) throw new AttributeNotOwned("Requested ChatRoom not owned");

            new Thread() { public void run() {
                try {
                    synchronized(_ChatRoom)
                    {
                        _ahvm_attributeValues =
                            _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs_CHATRoom_forUpdate.size());
                        _ahvm_attributeValues.put(_oah_CHATRoom_name,
                            _ChatRoom.name.toByteArray());
                        _ahvm_attributeValues.put(_oah_CHATRoom_slot,
                            _ChatRoom.slot.toByteArray());
                        //If not threaded off, this throws RTIinternalError: Concurrent access attempted to <method name>
                        _rtiAmbassador.updateAttributeValues(_ChatRoom.handle,
                            _ahvm_attributeValues, null);
                    } //synchronized(_ChatRoom)
                    } catch (Exception ignored) |
                        finally {
                            } //try
                    } }.start(); //Thread
                } catch (Exception intercepted) |
                    finally { //try
                        } //method provideAttributeValueUpdate
            } //class MychatRoomInstanceAttributeResponder
public class MyParticipantInstanceAttributeResponder
    extends FedAmbInstanceAttributeResponder
{
    public void provideAttributeValueUpdate(
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes,
        byte[] userSuppliedTag)
        throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotOwned,
            FederateInternalError
    {
        try {
            final aParticipant _Participant = SeekParticipant(theObject);
            if (_Participant == null) throw new ObjectInstanceNotKnown("Participant instance not known");
            if (!theAttributes.containsAll(_oabs_Participant_forUpdate)) throw new AttributeNotRecognized("Unexpected attribute set requested");
            if (_Participant.owned) throw new AttributeNotOwned("Requested Participant not owned");

            // We already knew that _rtiAmbassador calls from within
            // _fedAmbassador callbacks had to be threaded off
            // to avoid "RTIinternalError: Concurrent access attempted
to <method name>": now a bug in pRTI causes
            // ANY _rtiAmbassador-calling thread to freeze if it the
call triggers any _fedAmbassador callbacks and the
            // federate thread is currently waiting for a
            // synchronization monitor.
            // So we must avoid having the federate service thread
going into a wait state, which means more threading off!
            // (Or, in this case, moving the synchronization call into
            the child thread)
new Thread() { public void run() {
    try {
        // No need to synchronize earlier on _Participant
        // since the discovery and ownership handlers also synchronize on
        // _theParticpants
        // It is crucial to synchronize here, however, since
        // the object's registration may trigger this handler right away
        // (because of Auto-Provide) -- and the object cannot
        // fill its user_handle field in the same atomic operation.
        synchronized(_Participant) {
            final AttributeHandleValueMap
            _ahvm_attributeValues =
            _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs_Participant_forUpdate.size());
            _ahvm_attributeValues.put(_oah_Participant_logged_in,
            _Participant.logged_in.toByteArray());
            _ahvm_attributeValues.put(_oah_Participant_user_handle,
            _Participant.user_handle.toByteArray());
            _ahvm_attributeValues.put(_oah_Participant_chat_room_slot,
            _Participant.chat_room_slot.toByteArray());
            _rtiAmbassador.updateAttributeValues(_Participant.handle,
            _ahvm_attributeValues, null);
        } //synchronized(_Participant)
        } catch (Exception ignored) {
        } finally {
            } //try
    } }.start(); //Thread
} //method provideAttributeValueUpdate
} //class MyParticipantInstanceAttributeResponder
/**
 * Entry point for java exec command.
 * args[0] (optional) is the RTI host ("localhost" by default)
 * args[1] (optional) is the FDD file path (may be relative or
 * absolute; "MyChat.xml" by default)
 * args[2] (optional) is the federation execution name
 * ("MyChatRoom" by default)
 * args[3] (optional) is the federate name ("MyChatter" by default)
 * @param args the String[] command line arguments
 */
public static void main(String args[])
{
    new MyChat(args).show();
}

// Variables declaration - do not modify
private javax.swing.JButton btnLogon;
private javax.swing.JButton btnNewChatRoom;
private javax.swing.JButton btnSendMessage;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel5;
private javax.swing.JPanel jPanel6;
private javax.swing.JPanel jPanel7;
private javax.swing.JLabel lblChatRoom;
private javax.swing.JLabel lblSendTo;
private javax.swing.JLabel lblStatus;
private javax.swing.JComboBox lstChatRooms;
private javax.swing.JComboBox lstSendTo;
private javax.swing.JTextField txtArea;
private javax.swing.JTextField txtMessage;
private javax.swing.JTextField txtUsername;
// End of variables declaration

//end MyChat
Annex F – The JACK Chat Client

The JACK Chat Client is a command-line user-interface (CLI) application. This was done simply to provide a visual distinction between the two clients; nothing prevents JACK from using a GUI. However, one notes that JACK development cannot be conducted from within a Java IDE, since JAL is no longer Java. One would have to develop the GUI separately and then import it into JACK.

On the other hand, JACK’s built-in thread handling simplifies cross-task synchronization considerably. We started off by importing blocks of the Java client code, but soon realized that it was too “lumpy”; that is to say, tasks had not been broken down into small enough component steps. The way JACK implements semaphores, among other things, forced us to re-distribute the code between the plans. The end result is quite pleasing, JACK plans being a lot easier to read than our original Java chat client code.

The first part of this annex documents the capHLA1516 capability, which is the reusable part of the JACK-HLA interface. The second part details the JACK Chat Client itself.
Part One – The capHLA1516 Capability

The JACK capHLA1516 capability acts as an adaptor between JACK and HLA. Because JACK introduces a lot of automatic code, rather than give a full textual listing of the various files that make up the JACK Chat client, we’ve opted for a summary description similar to the arborescent view used by the JACK Development Environment.

// File: HLA1516.prj
"ProjectName" node: HLA1516
  "Documentation" node
    This is the JACK Repository project file for the HLA1516 package.
    See the "HLA1516 Capability Read Me.txt" file (under Other Files) for details.

Daniel U. Thibault
Daniel.Thibault@DRDC-RDDC.GC.Ca
2004 December 03
revised 2004 January 13
"Design Views" node
  "Overview" node
    "Documentation" node
      Overview of the capHLA1516 capability’s architecture
  "Agent Model" node
    "Agent Types" node
      (empty)
    "Capability Types" node
      "HLA1516" node
        "capHLA1516"
"Plan Types" node
 "HLA1516" node
  "plnHLAannounceSynchronizationPoint"
  "plnHLAattributeIsNotOwned"
  "plnHLAattributeIsOwnedByRTI"
  "plnHLAattributeOwnershipAcquisitionNotification"
  "plnHLAattributeOwnershipUnavailable"
  "plnHLAattributesInScope"
  "plnHLAattributesOutOfScope"
  "plnHLAconfirmAttributeOwnershipAcquisitionCancellation"
  "plnHLAdiscoverObjectInstance"
  "plnHLA federationNotRestored"
  "plnHLA federationNotSaved"
  "plnHLA federationRestoreBegun"
  "plnHLA federationRestored"
  "plnHLA federationRestoreStatusResponse"
  "plnHLA federationSaved"
  "plnHLA federationSynchronized"
  "plnHLA informAttributeOwnership"
  "plnHLA initiateFederateRestore"
  "plnHLA initiateFederateSave"
  "plnHLA objectInstanceNameReservationFailed"
  "plnHLA objectInstanceNameReservationSucceeded"
  "plnHLA provideAttributeValueUpdate"
  "plnHLAreceiveInteraction"
  "plnHLAreflectAttributeValues"
  "plnHLA removeObjectInstance"
  "plnHLA requestAttributeOwnershipAssumption"
  "plnHLA requestAttributeOwnershipRelease"
  "plnHLA requestDivestitureConfirmation"
  "plnHLA requestFederateRestoreFailed"
  "plnHLA requestFederateRestoreSucceeded"
  "plnHLA requestRetraction"
  "plnHLA startRegistrationForObjectClass"
  "plnHLA stopRegistrationForObjectClass"
  "plnHLA synchronizationPointRegistrationFailed"
  "plnHLA synchronizationPointRegistrationSucceeded"
  "plnHLA timeAdvanceGrant"
  "plnHLA timeConstrainedEnabled"
  "plnHLA timeRegulationEnabled"
  "plnHLA turnInteractionsOff"
  "plnHLA turnInteractionsOn"
  "plnHLA turnUpdatesOffForObjectInstance"
  "plnHLA turnUpdatesOnForObjectInstance"
"Event Types" node
"HLA1516" node
"evtHLAannounceSynchronizationPoint"
"evtHLAattributeIsNotOwned"
"evtHLAattributeIsOwnedByRTI"
"evtHLAattributeOwnershipAcquisitionNotification"
"evtHLAattributeOwnershipUnavailable"
"evtHLAattributesInScope"
"evtHLAattributesOutOfScope"
"evtHLAconfirmAttributeOwnershipAcquisitionCancellation"
"evtHLAdiscoverObjectInstance"
"evtHLAfederationNotRestored"
"evtHLAfederationNotSaved"
"evtHLAfederationRestoreBegun"
"evtHLAfederationRestored"
"evtHLAfederationRestoreStatusResponse"
"evtHLAfederationSaved"
"evtHLAfederationSaveStatusResponse"
"evtHLAfederationSynchronized"
"evtHLAinformAttributeOwnership"
"evtHLAinitiateFederateRestore"
"evtHLAinitiateFederateSave"
"evtHLAobjectInstanceNameReservationFailed"
"evtHLAobjectInstanceNameReservationSucceeded"
"evtHLAprovideAttributeValueUpdate"
"evtHLAreceiveInteraction"
"evtHLAreflectAttributeValues"
"evtHLAremoveObjectInstance"
"evtHLArequestAttributeOwnershipAssumption"
"evtHLArequestAttributeOwnershipRelease"
"evtHLArequestDivestitureConfirmation"
"evtHLArequestFederationRestoreFailed"
"evtHLArequestFederationRestoreSucceeded"
"evtHLArequestRetraction"
"evtHLAstartRegistrationForObjectClass"
"evtHLAstopRegistrationForObjectClass"
"evtHLAsynchronizationPointRegistrationFailed"
"evtHLAsynchronizationPointRegistrationSucceeded"
"evtHLAtimeAdvanceGrant"
"evtHLAtimeConstrainedEnabled"
"evtHLAtimeRegulationEnabled"
"evtHLAturnInteractionsOff"
"evtHLAturnInteractionsOn"
"evtHLAturnUpdatesOffForObjectInstance"
"evtHLAturnUpdatesOnForObjectInstance"
"Named Data" node
 "HLA1516" node
  "blfHLA dataHLA"
"Data Model" node
 "Beliefset Types" node
  "HLA1516" node
  "blfHLA"
"View Types" node
  (empty)
"External Classes" node
  (empty)
"Other Files" node
  "HLA1516 Capability Read Me.txt"
//end HLA1516
The `capHLA1516.cap` file contains large boiler-plate blocks; after the comment header (which explains in detail how to use the capability), there are long lists of handled events, posted events, and used plans. In the inner `HLAfederate` class itself, there is a long list of private members holding the various Validation handler references, followed later on by a similar list of `get/set` methods. Finally, there is the `FederateAmbassador` implementation, which follows a simple design pattern of invoking the `Validation` handler if specified, and then posting the event that corresponds to the callback.

```java
// File: capHLA1516.cap
package ca.gc.drdc_rddc.hla.rti1516.jack;
import ca.gc.drdc_rddc.hla.rti1516.FedAmb.*;
import hla.rti1516.*;

/**
 * Encapsulation of the HLA IEEE 1516 interface.
 *
 * To use the capHLA1516 capability:
 *
 * 1) Import the capability into your project (see HLA1516 Capability Read Me.txt for details).
 * 2) Let the agent (or capability) have the capHLA1516 capability.
 * 3) Create a Named Data instance of type ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA and drag it to the capability's external belief data import to replace the default one if you wish.
 * 4) At some point in your code (quite possibly the agent's constructor, but this need not be), create one or more instances of the capability's inner class `capHLA1516.HLAfederate` (see HLA1516 Capability Read Me.txt to learn how to invoke the constructor).
 *
 * The <unique string identifier> used when creating the HLAfederate instances is included as the identifier field of each of the HLA events that will be posted to the enclosing agent/capability. It is intended to be used by the latter to distinguish as needed between the HLAfederate instances it may have created.
 * Each HLAfederate can be passed as a FederateAmbassador argument to the RTIambassador's joinFederationExecution service invocation. By creating multiple RTIambassador and HLAfederate instances, an agent can thus play the roles of several HLA federates.
 * See HLA1516 Capability Read Me.txt to learn how to extend HLAfederate as needed.
 *
 * The capability's inner class (`capHLA1516.HLAfederate`) implements the whole FederateAmbassador interface.
 * Each HLA callback is translated into a JACK HLA event and posted to the enclosing agent (or capability). Overloaded callbacks are rolled into a single event with appropriate nulls.

```

1114 DRDC Valcartier TR 2007-412
The HLAfederate identifier can be queried with getIdentifier and even changed with setIdentifier.
An RTIambassador field (rtiAmbassador) is supplied for convenience (HLAfederate does not use it).
The imported belief data set datHLA stores the associations between HLAfederate identifiers and instance references.
It also insures the uniqueness of identifiers within its enclosing agent's scope; an attempt to create or rename an HLAfederate using an already-extant identifier will throw a BeliefSetException.

DISPOSING OF THE HLAfEDERATE INSTANCE:

Once the HLAfederate instance has outlived its usefulness, it can be removed from datHLA through the removeInstance query.
HLAfederate instances attempt to remove themselves from datHLA when garbage-collected (System.gc()), just in case they were disposed of by abnormal means.

WRITING PLANS TO HANDLE HLA EVENTS:

The enclosing agent/capability should declare it handles only those HLA events that matter to it; the capability handles each of the events it posts using irrelevant plans; this simple trick prevents the run-time from complaining that some events are not handled.

When an agent handles multiple federates, plans can bind to the correct HLAfederate instance by:

1) Declaring "HLA1516.capHLA1516.HLAfederate <HLAfederate_instance>";
2) Reading the belief data set (add "#reads data blfHLA datHLA;" to the plan code or simply drop the capability's imported datHLA on the plan's Belief Data); and
3) Starting the context() method with:
   
   ( null != ( <HLAfederate_instance> = datHLA.getInstance( <handled_event>.identifier ) ) )

Once this context step has succeeded, <HLAfederate_instance> is bound to the HLAfederate instance reference that matches the event's identifier field. Equivalently, you can declare an additional logical variable Object <Object_instance> and start the context() with:

   datHLA.get( <handled_event>.identifier, <Object_instance> ) &&
   ( null != ( <HLAfederate_instance> = (HLAfederate)<Object_instance>.as_object() ));

The latter form may be more convenient if you wish to typecast directly to your HLAfederate sub-class --you'd replace the (HLAfederate) typecast accordingly, of course.
WRITING HLA EVENT VALIDATORS:

Each of the 43 HLA events has a corresponding Validate interface in the ca.gc.drdc_rddc.hla.rtl1516.FedAmb package. For example, evtHLAannounceSynchronizationPoint has a ValidateAnnounceSynchronizationPoint interface.

To have the HLAfederate invoke a validation handler when it receives the HLA callback (within the Federate Service Thread), you need only register it with the HLAfederate instance at any time using the corresponding set method (e.g. setValidateAnnounceSynchronizationPoint()).

ON PLAYING WITH MULTIPLE FEDERATION EXECUTIONS AT ONCE:

The HLA spec and RTI implementations are quirky at this level. Nothing prevents a single RTIambassador instance from creating and deleting multiple federation executions, but it may join only one at a time.

This is because the RTI identifies the federate with the RTIambassador instance, and it considers the RTIambassador committed to the federation execution it joined until it resigns from it.

ON THE USE OF IRRELEVANT PLANS:

In JACK, internal capabilities are always more prominent than plans, so we have a small problem.

If we let capHLA1516 simply post the gamut of HLA events, the run-time will complain that some events are not being handled unless the user supplies "null plans" for each of those events he's not interested in. This is an annoyance.

To solve this, we built into capHLA1516 an event-handling capability for each of the HLA events. None of the capability's internal plans actually *do* anything, which is what we want for null handlers.

But because internal capabilities are more prominent than the agent's plans, if the user wanted to write meaningful handlers for the HLA events he would have to wrap them into a capability of his own, and then make sure that his capability is more prominent than the capHLA1516.

This is workable but unsafe.

To make capHLA1516 foolproof, we simply made each of our internal HLA plans irrelevant --that is to say, they all return false as their relevance() method.

This way, the run-time is satisfied, and the user can have his own picked HLA handling plans wherever he wishes them to be. Our plans will never be invoked and will thus always be glossed over should they be more prominent.

The fact that otherwise-unhandled events end up with no applicable plans at all does not bother JACK overmuch.
public capability
capHLA1516
  extends Capability
{
  #handles external event evtHLAannounceSynchronizationPoint;
  #handles external event evtHLAattributeIsNotOwned;
  #handles external event evtHLAattributeIsOwnedByRTI;
  #handles external event evtHLAAttributeOwnershipNotification;
  #handles external event evtHLAattributeOwnershipUnavailable;
  #handles external event evtHLAattributesInScope;
  #handles external event evtHLAattributesOutOfScope;
  #handles external event evtHLAconfirmAttributeOwnershipAcquisitionCancellation;
  #handles external event evtHLAdiscoverObjectInstance;
  #handles external event evtHLAfederationNotRestored;
  #handles external event evtHLAfederationNotSaved;
  #handles external event evtHLAfederationRestoreBegun;
  #handles external event evtHLAfederationRestored;
  #handles external event evtHLAfederationRestoreStatusResponse;
  #handles external event evtHLAfederationSaved;
  #handles external event evtHLAfederationSynchronized;
  #handles external event evtHLAinformAttributeOwnership;
  #handles external event evtHLAInitiateFederateRestore;
  #handles external event evtHLAInitiateFederateSave;
  #handles external event evtHLAObjectInstanceNameReservationFailed;
  #handles external event evtHLAObjectInstanceNameReservationSucceeded;
  #handles external event evtHLAProvideAttributeValueUpdate;
  #handles external event evtHLAreceiveInteraction;
  #handles external event evtHLAreflectAttributeValue;
  #handles external event evtHLAremoveObjectInstance;
  #handles external event evtHLArequestAttributeOwnershipAssumption;
  #handles external event evtHLArequestAttributeOwnershipRelease;
  #handles external event evtHLArequestDivestitureConfirmation;
  #handles external event evtHLArequestFederationRestoreFailed;
  #handles external event evtHLArequestFederationRestoreSucceeded;
  #handles external event evtHLArequestRetraction;
  #handles external event evtHLAstartRegistrationForObjectClass;
  #handles external event evtHLAstopRegistrationForObjectClass;
  #handles external event evtHLAsynchronizationPointRegistrationFailed;
  #handles external event evtHLAsynchronizationPointRegistrationSucceeded;
  #handles external event evtHLAtimeAdvanceGrant;
  #handles external event evtHLAtimeConstrainedEnabled;
  #handles external event evtHLAtimeRegulationEnabled;
  #handles external event evtHLATurnInteractionsOff;
  #handles external event evtHLATurnInteractionsOn;
  #handles external event evtHLATurnUpdatesOffForObjectInstance;
  #handles external event evtHLATurnUpdatesOnForObjectInstance;
# posts external event evtHLAannounceSynchronizationPoint
evHLAannounceSynchronizationPoint;
# posts external event evtHLAattributeIsNotOwned
evHLAattributeIsNotOwned;
# posts external event evtHLAattributeIsOwnedByRTI
evHLAattributeIsOwnedByRTI;
# posts external event
evtHLAAttributeOwnershipAcquisitionNotification
evHLAAttributeOwnershipAcquisitionNotification;
# posts external event evtHLAattributeOwnershipUnavailable
evHLAAttributeOwnershipUnavailable;
# posts external event evtHLAattributesInScope
evHLAattributesInScope;
# posts external event evtHLAattributesOutOfScope
evHLAattributesOutOfScope;
# posts external event
evtHLAconfirmAttributeOwnershipAcquisitionCancellation
evHLAconfirmAttributeOwnershipAcquisitionCancellation;
# posts external event evtHLAdiscoverObjectInstance
evHLAdiscoverObjectInstance;
# posts external event evtHLAfederationNotRestored
evHLAfederationNotRestored;
# posts external event evtHLAfederationNotSaved
evHLAfederationNotSaved;
# posts external event evtHLAfederationRestoreBegun
evHLAfederationRestoreBegun;
# posts external event evtHLAfederationRestored
evHLAfederationRestored;
# posts external event evtHLAfederationRestoreStatusResponse
evHLAfederationRestoreStatusResponse;
# posts external event evtHLAfederationSaved
evHLAfederationSaved;
# posts external event
evtHLAfederationSaveStatusResponse
evHLAfederationSaveStatusResponse;
# posts external event evtHLAfederationSynchronized
evHLAfederationSynchronized;
# posts external event evtHLAinformAttributeOwnership
evHLAinformAttributeOwnership;
# posts external event evtHLAinitiateFederateRestore
evHLAinitiateFederateRestore;
# posts external event evtHLAinitiateFederateSave
evHLAinitiateFederateSave;
# posts external event evtHLAobjectInstanceNameReservationFailed
evHLAobjectInstanceNameReservationFailed;
# posts external event evtHLAobjectInstanceNameReservationSucceeded
evHLAobjectInstanceNameReservationSucceeded;
# posts external event evtHLAprovideAttributeValueUpdate
evHLAprovideAttributeValueUpdate;
# posts external event evtHLAreceiveInteraction
evHLAreceiveInteraction;
# posts external event evtHLAreflectAttributeValueValues;
# posts external event evtHLAremoveObjectInstance
evHLAremoveObjectInstance;
# posts external event evtHLArequestAttributeOwnershipAssumption
evHLArequestAttributeOwnershipAssumption;
# posts external event evtHLArequestAttributeOwnershipRelease
evHLArequestAttributeOwnershipRelease;
# posts external event evtHLArequestDivestitureConfirmation
evHLArequestDivestitureConfirmation;
# posts external event evtHLArequestFederationRestoreFailed
evHLArequestFederationRestoreFailed;
# posts external event evtHLArequestFederationRestoreSucceeded
evHLArequestFederationRestoreSucceeded;
# posts external event evtHLArequestRetraction
evHLArequestRetraction;
# posts external event evtHLAstartRegistrationForObjectClass
evHLAstartRegistrationForObjectClass;
# posts external event evtHLAstopRegistrationForObjectClass
evHLAstopRegistrationForObjectClass;
# posts external event evtHLAsynchronizationPointRegistrationFailed
evHLAsynchronizationPointRegistrationFailed;
# posts external event evtHLAsynchronizationPointRegistrationSucceeded
evHLAsynchronizationPointRegistrationSucceeded;
# posts external event evtHLAtimeAdvanceGrant
evHLAtimeAdvanceGrant;
# posts external event evtHLAtimeConstrainedEnabled
evHLAtimeConstrainedEnabled;
# posts external event evtHLAtimeRegulationEnabled
evHLAtimeRegulationEnabled;
# posts external event evtHLAturnInteractionsOff
evHLAturnInteractionsOff;
# posts external event evtHLAturnInteractionsOn
evHLAturnInteractionsOn;
# posts external event evtHLAturnUpdatesOffForObjectInstance
evHLAturnUpdatesOffForObjectInstance;
# posts external event evtHLAturnUpdatesOnForObjectInstance
evHLAturnUpdatesOnForObjectInstance;
#uses plan plnHLAannounceSynchronizationPoint;
#uses plan plnHLAattributeIsNotOwned;
#uses plan plnHLAattributeIsOwnedByRTI;
#uses plan plnHLAattributeOwnershipAcquisitionNotification;
#uses plan plnHLAattributeOwnershipUnavailable;
#uses plan plnHLAattributesInScope;
#uses plan plnHLAattributesOutOfScope;
#uses plan plnHLAconfirmAttributeOwnershipAcquisitionCancellation;
#uses plan plnHLAdiscoverObjectInstance;
#uses plan plnHLAfederationNotRestored;
#uses plan plnHLAfederationNotSaved;
#uses plan plnHLAfederationRestoreBegun;
#uses plan plnHLAfederationRestored;
#uses plan plnHLAfederationRestoreStatusResponse;
#uses plan plnHLAfederationSaved;
#uses plan plnHLAfederationSaveStatusResponse;
#uses plan plnHLAfederationSynchronized;
#uses plan plnHLAinfoAttributeOwnership;
#uses plan plnHLAinitiateFederateRestore;
#uses plan plnHLAinitiateFederateSave;
#uses plan plnHLAobjectInstanceNameReservationFailed;
#uses plan plnHLAobjectInstanceNameReservationSucceeded;
#uses plan plnHLAprovideAttributeValueUpdate;
#uses plan plnHLAreceiveInteraction;
#uses plan plnHLAreflectAttributeValues;
#uses plan plnHLAremoveObjectInstance;
#uses plan plnHLArequestAttributeOwnershipAssumption;
#uses plan plnHLArequestAttributeOwnershipRelease;
#uses plan plnHLArequestDivestitureConfirmation;
#uses plan plnHLArequestFederationRestoreFailed;
#uses plan plnHLArequestFederationRestoreSucceeded;
#uses plan plnHLArequestRetraction;
#uses plan plnHLAstartRegistrationForObjectClass;
#uses plan plnHLAstopRegistrationForObjectClass;
#uses plan plnHLAsynchronizationPointRegistrationFailed;
#uses plan plnHLAsynchronizationPointRegistrationSucceeded;
#uses plan plnHLAtimeAdvanceGrant;
#uses plan plnHLAtimeConstrainedEnabled;
#uses plan plnHLAtimeRegulationEnabled;
#uses plan plnHLAturnInteractionsOff;
#uses plan plnHLAturnInteractionsOn;
#uses plan plnHLAturnUpdatesOffForObjectInstance;
#uses plan plnHLAturnUpdatesOnForObjectInstance;

#imports data blfHLA datHLA();
public HLAfederate
getNewHLAfederate(String identifier)
    throws aos.jack.jak.beliefset.BeliefSetException
{
    return new HLAfederate(identifier);
}

public class
HLAfederate
    implements FederateAmbassador
    /**
     * Method signature conflicts prevent all Validate interfaces
     * from being
     * accommodated by a single class. If this had not been the case,
     * HLAfederate could have implemented all of them and used
     * itself as a
     * null handler. Instead we'll guard each invocation.
     */
{
//The instance's identifier
private String _identifier;

//Convenience RTIambassador reference
public RTIambassador rtiAmbassador;
// Validation handlers
private
ValidateAnnounceSynchronizationPoint validateAnnounceSynchronizationPoint;
private
ValidateAttributeIsNotOwned validateAttributeIsNotOwned;
private
ValidateAttributeIsOwnedByRTI validateAttributeIsOwnedByRTI;
private
ValidateAttributeOwnershipAcquisitionNotification validateAttributeOwnershipAcquisitionNotification;
private
ValidateAttributeOwnershipUnavailable validateAttributeOwnershipUnavailable;
private
ValidateAttributesInScope validateAttributesInScope;
private
ValidateAttributesOutOfScope validateAttributesOutOfScope;
private
ValidateConfirmAttributeOwnershipAcquisitionCancellation validateConfirmAttributeOwnershipAcquisitionCancellation;
private
ValidateDiscoverObjectInstance validateDiscoverObjectInstance;
private
ValidateFederationNotRestored validateFederationNotRestored;
private
ValidateFederationNotSaved validateFederationNotSaved;
private
ValidateFederationRestoreBegun validateFederationRestoreBegun;
private
ValidateFederationRestored validateFederationRestored;
private
ValidateFederationRestoreStatusResponse validateFederationRestoreStatusResponse;
private
ValidateFederationSaved validateFederationSaved;
private
ValidateFederationSaveStatusResponse validateFederationSaveStatusResponse;
private
ValidateFederationSynchronized validateFederationSynchronized;
private
ValidateInformAttributeOwnership validateInformAttributeOwnership;
private
ValidateInitiateFederateRestore validateInitiateFederateRestore;
private
ValidateInitiateFederateSave validateInitiateFederateSave;
private
ValidateObjectInstanceNameReservationFailed validateObjectInstanceNameReservationFailed;
private
ValidateObjectInstanceNameReservationSucceeded validateObjectInstanceNameReservationSucceeded;
private ValidateProvideAttributeValueUpdate
validateProvideAttributeValueUpdate;
private ValidateReceiveInteraction validateReceiveInteraction;
private ValidateReflectAttributeValueValues validateReflectAttributeValueValues;
private ValidateRemoveObjectInstance validateRemoveObjectInstance;
private ValidateRequestAttributeOwnershipAssumption
validateRequestAttributeOwnershipAssumption;
private ValidateRequestAttributeOwnershipRelease
validateRequestAttributeOwnershipRelease;
private ValidateRequestDivestitureConfirmation
validateRequestDivestitureConfirmation;
private ValidateRequestFederationRestoreFailed
validateRequestFederationRestoreFailed;
private ValidateRequestFederationRestoreSucceeded
validateRequestFederationRestoreSucceeded;
private ValidateRequestRetraction validateRequestRetraction;
private ValidateStartRegistrationForObjectClass
validateStartRegistrationForObjectClass;
private ValidateStopRegistrationForObjectClass
validateStopRegistrationForObjectClass;
private ValidateSynchronizationPointRegistrationFailed
validateSynchronizationPointRegistrationFailed;
private ValidateSynchronizationPointRegistrationSucceeded
validateSynchronizationPointRegistrationSucceeded;
private ValidateTimeAdvanceGrant validateTimeAdvanceGrant;
private ValidateTimeConstrainedEnabled validateTimeConstrainedEnabled;
private ValidateTimeRegulationEnabled validateTimeRegulationEnabled;
private ValidateTurnInteractionsOff validateTurnInteractionsOff;
private ValidateTurnInteractionsOn validateTurnInteractionsOn;
private ValidateTurnUpdatesOffForObjectInstance
validateTurnUpdatesOffForObjectInstance;
private ValidateTurnUpdatesOnForObjectInstance
validateTurnUpdatesOnForObjectInstance;
/**
 * Constructor.
 * @param identifier A String used to distinguish this instance 
 * from others.
 * @throws BeliefSetException if the identifier is already in use
 */

public HLAfederate(String identifier)
throws aos.jack.jak.beliefset.BeliefSetException
{
    if (datHLA.hasKey(identifier)) throw new 
    aos.jack.jak.beliefset.BeliefSetException("Identifier already in use: " 
    + identifier + ");
    _identifier = identifier;
    datHLA.add(_identifier, this);
} //Constructor

/**
 * Called by the garbage collector on this object before 
 * disposing of it.
 * @throws Throwable if something goes wrong
 */

protected void finalize()
throws Throwable
{
    datHLA.remove(_identifier, this);
} //finalize

/**
 * Gets the current identifier.
 * @return the current String identifier
 */

public String getIdentifier()
{
    return _identifier;
}
/**
 * Sets the identifier to the new value and returns the previous one.
 * @param newIdentifier the new String identifier
 * @return the previous identifier
 * @throws BeliefSetException if the newIdentifier is already in use
 */
public String setIdentifier(String newIdentifier)
    throws aos.jack.jak.beliefset.BeliefSetException
{
    //Is newIdentifier already in use?
    if (datHLA.hasKey(newIdentifier))
        throw new aos.jack.jak.beliefset.BeliefSetException("Identifier already in use: "+ newIdentifier + ";
            String r = _identifier;
            _identifier = newIdentifier;
            datHLA.remove( r, this );
            datHLA.add( _identifier, this );
            return r;
}

/**
 * ####################################
 * Validation handlers access functions
 * ####################################
 */

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAnnounceSynchronizationPoint
    getValidateAnnounceSynchronizationPoint()
{
    return validateAnnounceSynchronizationPoint;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAnnounceSynchronizationPoint
    setValidateAnnounceSynchronizationPoint(ValidateAnnounceSynchronizationPoint
        newValidationHandler)
{
    ValidateAnnounceSynchronizationPoint r =
        validateAnnounceSynchronizationPoint;
    validateAnnounceSynchronizationPoint = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributeIsNotOwned
getValidateAttributeIsNotOwned()
{
    return validateAttributeIsNotOwned;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributeIsNotOwned
setValidateAttributeIsNotOwned(ValidateAttributeIsNotOwned
newValidationHandler)
{
    ValidateAttributeIsNotOwned r = validateAttributeIsNotOwned;
    validateAttributeIsNotOwned = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributeIsOwnedByRTI
getValidateAttributeIsOwnedByRTI()
{
    return validateAttributeIsOwnedByRTI;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributeIsOwnedByRTI
setValidateAttributeIsOwnedByRTI(ValidateAttributeIsOwnedByRTI
newValidationHandler)
{
    ValidateAttributeIsOwnedByRTI r = validateAttributeIsOwnedByRTI;
    validateAttributeIsOwnedByRTI = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributeOwnershipAcquisitionNotification
getValidateAttributeOwnershipAcquisitionNotification()
{
    return validateAttributeOwnershipAcquisitionNotification;
}
/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributeOwnershipAcquisitionNotification
setValidateAttributeOwnershipAcquisitionNotification(ValidateAttribute
OwnershipAcquisitionNotification newValidationHandler)
{
    ValidateAttributeOwnershipAcquisitionNotification r =
    validateAttributeOwnershipAcquisitionNotification;
    validateAttributeOwnershipAcquisitionNotification =
    newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributeOwnershipUnavailable
getValidateAttributeOwnershipUnavailable()
{
    return validateAttributeOwnershipUnavailable;
}
/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributeOwnershipUnavailable
setValidateAttributeOwnershipUnavailable(ValidateAttributeOwnershipUnava
lable newValidationHandler)
{
    ValidateAttributeOwnershipUnavailable r =
    validateAttributeOwnershipUnavailable;
    validateAttributeOwnershipUnavailable = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributesInScope
getValidateAttributesInScope()
{
    return validateAttributesInScope;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributesInScope
setValidateAttributesInScope(ValidateAttributesInScope
newValidationHandler)
{
    ValidateAttributesInScope r = validateAttributesInScope;
    validateAttributesInScope = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateAttributesOutOfScope
getValidateAttributesOutOfScope()
{
    return validateAttributesOutOfScope;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateAttributesOutOfScope
setValidateAttributesOutOfScope(ValidateAttributesOutOfScope
newValidationHandler)
{
    ValidateAttributesOutOfScope r =
    validateAttributesOutOfScope;
    validateAttributesOutOfScope = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateConfirmAttributeOwnershipAcquisitionCancellation getValidateConfirmAttributeOwnershipAcquisitionCancellation()
{
    return validateConfirmAttributeOwnershipAcquisitionCancellation;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateConfirmAttributeOwnershipAcquisitionCancellation setValidateConfirmAttributeOwnershipAcquisitionCancellation(ValidateConfirmAttributeOwnershipAcquisitionCancellation newValidationHandler)
{
    ValidateConfirmAttributeOwnershipAcquisitionCancellation r =
    validateConfirmAttributeOwnershipAcquisitionCancellation;
    validateConfirmAttributeOwnershipAcquisitionCancellation =
    newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateDiscoverObjectInstance getValidateDiscoverObjectInstance()
{
    return validateDiscoverObjectInstance;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateDiscoverObjectInstance setValidateDiscoverObjectInstance(ValidateDiscoverObjectInstance newValidationHandler)
{
    ValidateDiscoverObjectInstance r =
    validateDiscoverObjectInstance;
    validateDiscoverObjectInstance = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationNotRestored
getValidateFederationNotRestored()
{
    return validateFederationNotRestored;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationNotRestored
setValidateFederationNotRestored(ValidateFederationNotRestored
newValidationHandler)
{
    ValidateFederationNotRestored r =
    validateFederationNotRestored;
    validateFederationNotRestored = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationNotSaved
getValidateFederationNotSaved()
{
    return validateFederationNotSaved;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationNotSaved
setValidateFederationNotSaved(ValidateFederationNotSaved
newValidationHandler)
{
    ValidateFederationNotSaved r = validateFederationNotSaved;
    validateFederationNotSaved = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationRestoreBegun getValidateFederationRestoreBegun()
{
    return validateFederationRestoreBegun;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationRestoreBegun setValidateFederationRestoreBegun(ValidateFederationRestoreBegun newValidationHandler)
{
    ValidateFederationRestoreBegun r = validateFederationRestoreBegun;
    validateFederationRestoreBegun = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationRestored getValidateFederationRestored()
{
    return validateFederationRestored;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationRestored setValidateFederationRestored(ValidateFederationRestored newValidationHandler)
{
    ValidateFederationRestored r = validateFederationRestored;
    validateFederationRestored = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationRestoreStatusResponse
getValidateFederationRestoreStatusResponse()
{
    return validateFederationRestoreStatusResponse;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationRestoreStatusResponse
setValidateFederationRestoreStatusResponse(ValidateFederationRestoreStatusResponse
newValidationHandler)
{
    ValidateFederationRestoreStatusResponse r =
    validateFederationRestoreStatusResponse;
    validateFederationRestoreStatusResponse =
    newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationSaved
getValidateFederationSaved()
{
    return validateFederationSaved;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationSaved
setValidateFederationSaved(ValidateFederationSaved
newValidationHandler)
{
    ValidateFederationSaved r = validateFederationSaved;
    validateFederationSaved = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationSaveStatusResponse
getValidateFederationSaveStatusResponse()
{
    return validateFederationSaveStatusResponse;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationSaveStatusResponse
setValidateFederationSaveStatusResponse(ValidateFederationSaveStatusResponse
newValidationHandler)
{
    ValidateFederationSaveStatusResponse r =
    validateFederationSaveStatusResponse;
    validateFederationSaveStatusResponse = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateFederationSynchronized
getValidateFederationSynchronized()
{
    return validateFederationSynchronized;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateFederationSynchronized
setValidateFederationSynchronized(ValidateFederationSynchronized
newValidationHandler)
{
    ValidateFederationSynchronized r =
    validateFederationSynchronized;
    validateFederationSynchronized = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateInformAttributeOwnership
getValidateInformAttributeOwnership()
{
    return validateInformAttributeOwnership;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateInformAttributeOwnership
setValidateInformAttributeOwnership(ValidateInformAttributeOwnership
newValidationHandler)
{
    ValidateInformAttributeOwnership r =
    validateInformAttributeOwnership;
    validateInformAttributeOwnership = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateInitiateFederateRestore
getValidateInitiateFederateRestore()
{
    return validateInitiateFederateRestore;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateInitiateFederateRestore
setValidateInitiateFederateRestore(ValidateInitiateFederateRestore
newValidationHandler)
{
    ValidateInitiateFederateRestore r =
    validateInitiateFederateRestore;
    validateInitiateFederateRestore = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateInitiateFederateSave getValidateInitiateFederateSave()
{
    return validateInitiateFederateSave;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateInitiateFederateSave setValidateInitiateFederateSave(ValidateInitiateFederateSave
newValidationHandler)
{
    ValidateInitiateFederateSave r = validateInitiateFederateSave;
    validateInitiateFederateSave = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateObjectInstanceNameReservationFailed getValidateObjectInstanceNameReservationFailed()
{
    return validateObjectInstanceNameReservationFailed;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateObjectInstanceNameReservationFailed setValidateObjectInstanceNameReservationFailed
(ValidateObjectInstanceNameReservationFailed newValidationHandler)
{
    ValidateObjectInstanceNameReservationFailed r = validateObjectInstanceNameReservationFailed;
    validateObjectInstanceNameReservationFailed = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateObjectInstanceNameReservationSucceeded getValidateObjectInstanceNameReservationSucceeded()
{
    return validateObjectInstanceNameReservationSucceeded;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateObjectInstanceNameReservationSucceeded setValidateObjectInstanceNameReservationSucceeded(ValidateObjectInstanceNameReservationSucceeded newValidationHandler)
{
    ValidateObjectInstanceNameReservationSucceeded r = validateObjectInstanceNameReservationSucceeded;
    validateObjectInstanceNameReservationSucceeded = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateProvideAttributeValueUpdate getValidateProvideAttributeValueUpdate()
{
    return validateProvideAttributeValueUpdate;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateProvideAttributeValueUpdate setValidateProvideAttributeValueUpdate(ValidateProvideAttributeValueUpdate newValidationHandler)
{
    ValidateProvideAttributeValueUpdate r = validateProvideAttributeValueUpdate;
    validateProvideAttributeValueUpdate = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateReceiveInteraction getValidateReceiveInteraction()
{
    return validateReceiveInteraction;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateReceiveInteraction setValidateReceiveInteraction(ValidateReceiveInteraction newValidationHandler)
{
    ValidateReceiveInteraction r = validateReceiveInteraction;
    validateReceiveInteraction = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateReflectAttributeValues getValidateReflectAttributeValues()
{
    return validateReflectAttributeValues;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateReflectAttributeValues setValidateReflectAttributeValues(ValidateReflectAttributeValues newValidationHandler)
{
    ValidateReflectAttributeValues r = validateReflectAttributeValues;
    validateReflectAttributeValues = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRemoveObjectInstance
getValidateRemoveObjectInstance()
{
    return validateRemoveObjectInstance;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRemoveObjectInstance
setValidateRemoveObjectInstance(ValidateRemoveObjectInstance
newValidationHandler)
{
    ValidateRemoveObjectInstance r =
    validateRemoveObjectInstance;
    validateRemoveObjectInstance = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestAttributeOwnershipAssumption
getValidateRequestAttributeOwnershipAssumption()
{
    return validateRequestAttributeOwnershipAssumption;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestAttributeOwnershipAssumption
setValidateRequestAttributeOwnershipAssumption(ValidateRequestAttributeOwnershipAssumption
newValidationHandler)
{
    ValidateRequestAttributeOwnershipAssumption r =
    validateRequestAttributeOwnershipAssumption;
    validateRequestAttributeOwnershipAssumption =
    newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestAttributeOwnershipRelease getValidateRequestAttributeOwnershipRelease()
{
    return validateRequestAttributeOwnershipRelease;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestAttributeOwnershipRelease setValidateRequestAttributeOwnershipRelease(ValidateRequestAttributeOwnershipRelease newValidationHandler)
{
    ValidateRequestAttributeOwnershipRelease r =
    validateRequestAttributeOwnershipRelease;
    validateRequestAttributeOwnershipRelease =
    newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestDivestitureConfirmation getValidateRequestDivestitureConfirmation()
{
    return validateRequestDivestitureConfirmation;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestDivestitureConfirmation setValidateRequestDivestitureConfirmation(ValidateRequestDivestitureConfirmation newValidationHandler)
{
    ValidateRequestDivestitureConfirmation r =
    validateRequestDivestitureConfirmation;
    validateRequestDivestitureConfirmation =
    newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestFederationRestoreFailed getValidateRequestFederationRestoreFailed()
{
    return validateRequestFederationRestoreFailed;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestFederationRestoreFailed setValidateRequestFederationRestoreFailed(ValidateRequestFederationRestoreFailed newValidationHandler)
{
    ValidateRequestFederationRestoreFailed r = validateRequestFederationRestoreFailed;
    validateRequestFederationRestoreFailed = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestFederationRestoreSucceeded getValidateRequestFederationRestoreSucceeded()
{
    return validateRequestFederationRestoreSucceeded;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestFederationRestoreSucceeded setValidateRequestFederationRestoreSucceeded(ValidateRequestFederationRestoreSucceeded newValidationHandler)
{
    ValidateRequestFederationRestoreSucceeded r = validateRequestFederationRestoreSucceeded;
    validateRequestFederationRestoreSucceeded = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateRequestRetraction
getValidateRequestRetraction()
{
    return validateRequestRetraction;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateRequestRetraction
setValidateRequestRetraction(ValidateRequestRetraction
newValidationHandler)
{
    ValidateRequestRetraction r = validateRequestRetraction;
    validateRequestRetraction = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateStartRegistrationForObjectClass
getValidateStartRegistrationForObjectClass()
{
    return validateStartRegistrationForObjectClass;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateStartRegistrationForObjectClass
setValidateStartRegistrationForObjectClass(ValidateStartRegistrationForObjectClass
newValidationHandler)
{
    ValidateStartRegistrationForObjectClass r = validateStartRegistrationForObjectClass;
    validateStartRegistrationForObjectClass = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateStopRegistrationForObjectClass
getValidateStopRegistrationForObjectClass()
{
    return validateStopRegistrationForObjectClass;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateStopRegistrationForObjectClass
setValidateStopRegistrationForObjectClass(ValidateStopRegistrationForObjectClass
newValidationHandler)
{
    ValidateStopRegistrationForObjectClass r =
validateStopRegistrationForObjectClass;
    validateStopRegistrationForObjectClass =
newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateSynchronizationPointRegistrationFailed
getValidateSynchronizationPointRegistrationFailed()
{
    return validateSynchronizationPointRegistrationFailed;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateSynchronizationPointRegistrationFailed
setValidateSynchronizationPointRegistrationFailed(ValidateSynchronizationPointRegistrationFailed
newValidationHandler)
{
    ValidateSynchronizationPointRegistrationFailed r =
validateSynchronizationPointRegistrationFailed;
    validateSynchronizationPointRegistrationFailed =
newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateSynchronizationPointRegistrationSucceeded
getValidateSynchronizationPointRegistrationSucceeded()
{
    return validateSynchronizationPointRegistrationSucceeded;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateSynchronizationPointRegistrationSucceeded
setValidateSynchronizationPointRegistrationSucceeded(ValidateSynchronizationPointRegistrationSucceeded newValidationHandler)
{
    ValidateSynchronizationPointRegistrationSucceeded r =
    validateSynchronizationPointRegistrationSucceeded;
    validateSynchronizationPointRegistrationSucceeded =
    newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTimeAdvanceGrant
getValidateTimeAdvanceGrant()
{
    return validateTimeAdvanceGrant;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTimeAdvanceGrant
setValidateTimeAdvanceGrant(ValidateTimeAdvanceGrant newValidationHandler)
{
    ValidateTimeAdvanceGrant r = validateTimeAdvanceGrant;
    validateTimeAdvanceGrant = newValidationHandler;
    return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTimeConstrainedEnabled
getValidateTimeConstrainedEnabled()
{
    return validateTimeConstrainedEnabled;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTimeConstrainedEnabled
setValidateTimeConstrainedEnabled(ValidateTimeConstrainedEnabled
newValidationHandler)
{
    ValidateTimeConstrainedEnabled r =
    validateTimeConstrainedEnabled;
    validateTimeConstrainedEnabled = newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTimeRegulationEnabled
getValidateTimeRegulationEnabled()
{
    return validateTimeRegulationEnabled;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTimeRegulationEnabled
setValidateTimeRegulationEnabled(ValidateTimeRegulationEnabled
newValidationHandler)
{
    ValidateTimeRegulationEnabled r =
    validateTimeRegulationEnabled;
    validateTimeRegulationEnabled = newValidationHandler;
    return r;
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTurnInteractionsOff getValidateTurnInteractionsOff()
{
  return validateTurnInteractionsOff;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTurnInteractionsOff setValidateTurnInteractionsOff(ValidateTurnInteractionsOff newValidationHandler)
{
  ValidateTurnInteractionsOff r = validateTurnInteractionsOff;
  validateTurnInteractionsOff = newValidationHandler;
  return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTurnInteractionsOn getValidateTurnInteractionsOn()
{
  return validateTurnInteractionsOn;
}

/**
 * Sets the Validation handler to the new value and returns the previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTurnInteractionsOn setValidateTurnInteractionsOn(ValidateTurnInteractionsOn newValidationHandler)
{
  ValidateTurnInteractionsOn r = validateTurnInteractionsOn;
  validateTurnInteractionsOn = newValidationHandler;
  return r;
}
/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTurnUpdatesOffForObjectInstance
getValidateTurnUpdatesOffForObjectInstance()
{
    return validateTurnUpdatesOffForObjectInstance;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTurnUpdatesOffForObjectInstance
setValidateTurnUpdatesOffForObjectInstance(ValidateTurnUpdatesOffForObjectInstance
newValidationHandler)
{
    ValidateTurnUpdatesOffForObjectInstance r =
    validateTurnUpdatesOffForObjectInstance;
    validateTurnUpdatesOffForObjectInstance =
newValidationHandler;
    return r;
}

/**
 * Gets the current Validation handler.
 * @return the current Validation handler
 */
public ValidateTurnUpdatesOnForObjectInstance
getValidateTurnUpdatesOnForObjectInstance()
{
    return validateTurnUpdatesOnForObjectInstance;
}

/**
 * Sets the Validation handler to the new value and returns the
 * previous one.
 * @param newValidationHandler the new Validation handler
 * @return the previous Validation handler
 */
public ValidateTurnUpdatesOnForObjectInstance
setValidateTurnUpdatesOnForObjectInstance(ValidateTurnUpdatesOnForObjectInstance
newValidationHandler)
{
    ValidateTurnUpdatesOnForObjectInstance r =
    validateTurnUpdatesOnForObjectInstance;
    validateTurnUpdatesOnForObjectInstance =
newValidationHandler;
    return r;
}
/**
 * ################################################################
 * FederateAmbassador interface
 * ################################################################
 */

public void synchronizationPointRegistrationSucceeded(
    String synchronizationPointLabel)
throws FederateInternalError {
    if (null!=validateSynchronizationPointRegistrationSucceeded)
        validateSynchronizationPointRegistrationSucceeded.validate(
            synchronizationPointLabel);
    postEvent(evHLAisynchronizationPointRegistrationSucceeded.created{
        _identifier,
        synchronizationPointLabel
    });
} //synchronizationPointRegistrationSucceeded

public void synchronizationPointRegistrationFailed(
    String synchronizationPointLabel,
    SynchronizationPointFailureReason reason)
throws FederateInternalError {
    if (null!=validateSynchronizationPointRegistrationFailed)
        validateSynchronizationPointRegistrationFailed.validate(
            synchronizationPointLabel,
            reason);
    postEvent(evHLAisynchronizationPointRegistrationFailed.created{
        _identifier,
        synchronizationPointLabel,
        reason
    });
} //synchronizationPointRegistrationFailed

public void announceSynchronizationPoint(
    String synchronizationPointLabel,
    byte[] userSuppliedTag)
throws FederateInternalError {
    if (null!=validateAnnounceSynchronizationPoint)
        validateAnnounceSynchronizationPoint.validate(
            synchronizationPointLabel,
            userSuppliedTag);
    postEvent(evHLAannounceSynchronizationPoint.created{
        _identifier,
        synchronizationPointLabel,
        userSuppliedTag
    });
} //announceSynchronizationPoint
public void federationSynchronized(String synchronizationPointLabel)
    throws FederateInternalError
{
    if (null != validateFederationSynchronized)
    {
        validateFederationSynchronized.validate(
            synchronizationPointLabel);
        postEvent(evHLAfederationSynchronized.create(
            _identifier,
            synchronizationPointLabel))
    }
}

public void initiateFederateSave(String label)
    throws UnableToPerformSave, FederateInternalError
{
    if (null != validateInitiateFederateSave)
    {
        validateInitiateFederateSave.validate(
            label);
        postEvent(evHLAinitiateFederateSave.create(
            _identifier,
            label,
            null))
    }
}

public void initiateFederateSave(String label, LogicalTime time)
    throws InvalidLogicalTime, UnableToPerformSave, FederateInternalError
{
    if (null != validateInitiateFederateSave)
    {
        validateInitiateFederateSave.validate(
            label, time);
        postEvent(evHLAinitiateFederateSave.create(
            _identifier,
            label,
            time))
    }
}
public void federationSaved()
throws FederateInternalError
{
    if (null!=validateFederationSaved)
        validateFederationSaved.validate();
    postEvent(evHLAfederationSaved.create(
            _identifier)
    );
} //federationSaved

public void federationNotSaved(
    SaveFailureReason reason)
throws FederateInternalError
{
    if (null!=validateFederationNotSaved)
        validateFederationNotSaved.validate(
            reason);
    postEvent(evHLAfederationNotSaved.create(
            _identifier,
            reason)
    );
} //federationNotSaved

public void federationSaveStatusResponse(
    FederateHandleSaveStatusPair[] response)
throws FederateInternalError
{
    if (null!=validateFederationSaveStatusResponse)
        validateFederationSaveStatusResponse.validate(
            response);
    postEvent(evHLAfederationSaveStatusResponse.create(
            _identifier,
            response)
    );
} //federationSaveStatusResponse

public void requestFederationRestoreSucceeded(
    String label)
throws FederateInternalError
{
    if (null!=validateRequestFederationRestoreSucceeded)
        validateRequestFederationRestoreSucceeded.validate(
            label);
    postEvent(evHLArequestFederationRestoreSucceeded.create(
            _identifier,
            label)
    );
} //requestFederationRestoreSucceeded
public void requestFederationRestoreFailed(
    String label)
throws FederateInternalError
{
    if (null!=validateRequestFederationRestoreFailed)
        validateRequestFederationRestoreFailed.validate(
            label);
    postEvent(evHLArequestFederationRestoreFailed.create(
        _identifier,
        _label)
    );
} //requestFederationRestoreFailed

public void federationRestoreBegun()
throws FederateInternalError
{
    if (null!=validateFederationRestoreBegun)
        validateFederationRestoreBegun.validate();
    postEvent(evHLAinitiateFederateRestore.create(
        _identifier)
    );
} //federationRestoreBegun

public void initiateFederateRestore(
    String    label,
    FederateHandle federateHandle)
throws SpecifiedSaveLabelDoesNotExist,
       CouldNotInitiateRestore,
       FederateInternalError
{
    if (null!=validateInitiateFederateRestore)
        validateInitiateFederateRestore.validate(
            label,
            federateHandle);
    postEvent(evHLAinitiateFederateRestore.create(
        _identifier,
        _label,
        federateHandle)
    );
} //initiateFederateRestore

public void federationRestored()
throws FederateInternalError
{
    if (null!=validateFederationRestored)
        validateFederationRestored.validate();
    postEvent(evHLAinitiateFederateRestore.create(
        _identifier)
    );
} //federationRestored
public void federationNotRestored(
    RestoreFailureReason reason)
throws FederateInternalError
{
    if (null!=validateFederationNotRestored)
        validateFederationNotRestored.validate(
            reason);
    postEvent(evHLAfederationNotRestored.create(
            _identifier,
            reason)) ;
}; //federationNotRestored

public void federationRestoreStatusResponse(
    FederateHandleRestoreStatusPair[] response)
throws FederateInternalError
{
    if (null!=validateFederationRestoreStatusResponse)
        validateFederationRestoreStatusResponse.validate(
            response);
    postEvent(evHLAfederationRestoreStatusResponse.create(
            _identifier,
            response)) ;
}; //federationRestoreStatusResponse

public void startRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished,
        FederateInternalError
{
    if (null!=validateStartRegistrationForObjectClass)
        validateStartRegistrationForObjectClass.validate(
            theClass);
    postEvent(evHLAstartRegistrationForObjectClass.create(
            _identifier,
            theClass)) ;
}; //startRegistrationForObjectClass

public void stopRegistrationForObjectClass(
    ObjectClassHandle theClass)
throws ObjectClassNotPublished,
        FederateInternalError
{
    if (null!=validateStopRegistrationForObjectClass)
        validateStopRegistrationForObjectClass.validate(
            theClass);
    postEvent(evHLAstopRegistrationForObjectClass.create(
            _identifier,
            theClass)) ;
}; //stopRegistrationForObjectClass
public void turnInteractionsOn(
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished,
    FederateInternalError
{
    if (null != validateTurnInteractionsOn)
        validateTurnInteractionsOn.validate(
            theHandle);
    postEvent(evHLAturnInteractionsOn.create(
            identifier,
            theHandle)) ;
} //turnInteractionsOn

public void
turnInteractionsOff( 
    InteractionClassHandle theHandle)
throws InteractionClassNotPublished, 
    FederateInternalError
{
    if (null != validateTurnInteractionsOff)
        validateTurnInteractionsOff.validate(
            theHandle);
    postEvent(evHLAturnInteractionsOff.create(
            identifier, 
            theHandle)) ;
} //turnInteractionsOff

public void
objectInstanceNameReservationSucceeded(
    String objectName)
throws UnknownName, 
    FederateInternalError
{
    if (null != validateObjectInstanceNameReservationSucceeded)
        validateObjectInstanceNameReservationSucceeded.validate(
            objectName);
    postEvent(evHLAobjectInstanceNameReservationSucceeded.create(
            identifier, 
            objectName)) ;
} //objectInstanceNameReservationSucceeded
public void objectInstanceNameReservationFailed(
    String objectName)
    throws UnknownName,
        FederateInternalError
{
    if (null != validateObjectNameReservationFailed)
        validateObjectNameReservationFailed.validate(
            objectName);
    postEvent(evHLA.objectInstanceNameReservationFailed.create(
        _identifier,
        objectName)
    );
}
} //objectInstanceNameReservationFailed

public void discoverObjectInstance(
    ObjectInstanceHandle theObject,
    ObjectClassHandle theObjectClass,
    String objectName)
    throws CouldNotDiscover,
        ObjectClassNotRecognized,
        FederateInternalError
{
    if (null != validateDiscoverObjectInstance)
        validateDiscoverObjectInstance.validate(
            theObject,
            theObjectClass,
            objectName);
    postEvent(evHLA.discoverObjectInstance.create(
        _identifier,
        theObject,
        theObjectClass,
        objectName)
    );
}
} //discoverObjectInstance
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    if (null != validateReflectAttributeValues)
        validateReflectAttributeValues.validate(
            theObject,
            theAttributes,
            userSuppliedTag,
            sentOrdering,
            theTransport);
    postEvent(evHLAreflectAttributeValues.create(
            identifier,
            theObject,
            theAttributes,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            null,
            null,
            null,
            null));
} //reflectAttributeValues
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
throws ObjectInstanceNotKnown, 
      AttributeNotRecognized,
      AttributeNotSubscribed,
      FederateInternalError
{
    if (null!=validateReflectAttributeValues)
        validateReflectAttributeValues.validate(
            theObject,
            theAttributes,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            sentRegions);
    postEvent(evHLAreflectAttributeValues.create(
        _identifier,
        theObject,
        theAttributes,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        null,
        null,
        null,
        sentRegions)
    );
} //reflectAttributeValues
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValuePair theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    if (null!-validateReflectAttributeValues)
    validateReflectAttributeValues.validate(
        theObject,
        theAttributes,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering);
    postEvent(evHLAreflectAttributeValues.create(
        _identifier,
        theObject,
        theAttributes,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering,
        null,
        null)
    );
} //reflectAttributeValues
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
  throws ObjectInstanceNotKnown,
         AttributeNotRecognized,
         AttributeNotSubscribed,
         FederateInternalError
{
    if (null!=validateReflectAttributeValues)
        validateReflectAttributeValues.validate(
            theObject,
            theAttributes,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            sentRegions);
    postEvent(evHLAreflectAttributeValues.create(
        _identifier,
        theObject,
        theAttributes,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering,
        null,
        sentRegions)
        );
}  //reflectAttributeValues
public void reflectAttributeValueValues(
  ObjectInstanceHandle theObject,
  AttributeHandleValueMap theAttributes,
  byte[] userSuppliedTag,
  OrderType sentOrdering,
  TransportationType theTransport,
  LogicalTime theTime,
  OrderType receivedOrdering,
  MessageRetractionHandle retractionHandle)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    InvalidLogicalTime,
    FederateInternalError
{
  if (null != validateReflectAttributeValueValues)
    validateReflectAttributeValueValues.validate(
      theObject,
      theAttributes,
      userSuppliedTag,
      sentOrdering,
      theTransport,
      theTime,
      receivedOrdering,
      retractionHandle);
  postEvent(evHLAReflectAttributeValueValues.create(
    _identifier,
    theObject,
    theAttributes,
    userSuppliedTag,
    sentOrdering,
    theTransport,
    theTime,
    receivedOrdering,
    retractionHandle,
    null)
    );
} //reflectAttributeValueValues
public void reflectAttributeValues(
    ObjectInstanceHandle theObject,
    AttributeHandleValueMap theAttributes,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle retractionHandle,
    RegionHandleSet sentRegions)
    throws ObjectInstanceNotKnown,
            AttributeNotRecognized,
            AttributeNotSubscribed,
            InvalidLogicalTime,
            FederateInternalError
    {
        if (null!=validateReflectAttributeValues)
            validateReflectAttributeValues.validate(
                theObject,
                theAttributes,
                userSuppliedTag,
                sentOrdering,
                theTransport,
                theTime,
                receivedOrdering,
                retractionHandle,
                sentRegions);
        postEvent(evHLAreflectAttributeValues.create(
            identifier,
            theObject,
            theAttributes,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            retractionHandle,
            sentRegions)
    );
} //reflectAttributeValues
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport)
throws InteractionClassNotRecognized,
            InteractionParameterNotRecognized,
            InteractionClassNotSubscribed,
            FederateInternalError
{
    if (null!=validateReceiveInteraction)
        validateReceiveInteraction.validate(
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport);
    postEvent(evHLAreceiveInteraction.create(
        identifier,
        interactionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        null,
        null,
        null,
        null)
    );
} //receiveInteraction
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byteArray userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    RegionHandleSet sentRegions)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    if (null!-validateReceiveInteraction)
    validateReceiveInteraction.validate(
        interactionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        sentRegions);
    postEvent(evHLAreceiveInteraction.create(
        _identifier,
        InteractionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        null,
        null,
        null,
        sentRegions);
};
} //receiveInteraction
public void
receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValuePairMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    if (null!=validateReceiveInteraction)
        validateReceiveInteraction.validate(
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering);
    postEvent(evHLA_receiveInteraction.create(
            identifier,
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            null,
            null)
        );
} //receiveInteraction
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    RegionHandleSet sentRegions)
throws InteractionClassNotRecognized,
    InteractionParameterNotRecognized,
    InteractionClassNotSubscribed,
    FederateInternalError
{
    if (null!=validateReceiveInteraction)
        validateReceiveInteraction.validate(
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            sentRegions);
    postEvent(evHLAreceiveInteraction.create(
        identifier,
        interactionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering,
        null,
        sentRegions)
    );
} //receiveInteraction
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle)
throws InteractionClassNotRecognized,
          InteractionParameterNotRecognized,
          InteractionClassNotSubscribed,
          InvalidLogicalTime,
          FederateInternalError
{
    if (null != validateReceiveInteraction)
        validateReceiveInteraction.validate(
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            messageRetractionHandle);
    postEvent(evHLAreceiveInteraction.create(
        _identifier,
        interactionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering,
        messageRetractionHandle,
        null));
} //receiveInteraction
public void receiveInteraction(
    InteractionClassHandle interactionClass,
    ParameterHandleValueMap theParameters,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    TransportationType theTransport,
    LogicalTime theTime,
    OrderType receivedOrdering,
    MessageRetractionHandle messageRetractionHandle,
    RegionHandleSet sentRegions)
    throws InteractionClassNotRecognized,
                 InteractionParameterNotRecognized,
                 InteractionClassNotSubscribed,
                 InvalidLogicalTime,
                 FederateInternalError
{
    if (null!-validateReceiveInteraction)
        validateReceiveInteraction.validate(
            interactionClass,
            theParameters,
            userSuppliedTag,
            sentOrdering,
            theTransport,
            theTime,
            receivedOrdering,
            messageRetractionHandle,
            sentRegions);
    postEvent(evHLAreceiveInteraction.create(
        identifier,
        InteractionClass,
        theParameters,
        userSuppliedTag,
        sentOrdering,
        theTransport,
        theTime,
        receivedOrdering,
        messageRetractionHandle,
        sentRegions)
    );
} //receiveInteraction
public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering)
throws ObjectInstanceNotKnown,
    FederateInternalError
{
    if (null == validateRemoveObjectInstance)
        validateRemoveObjectInstance.validate(
            theObject,
            userSuppliedTag,
            sentOrdering);
    postEvent(evHLAremoveObjectInstance.create(
        _identifier,
        theObject,
        userSuppliedTag,
        sentOrdering,
        null,
        null,
        null
    ));
} //removeObjectInstance

public void removeObjectInstance(
    ObjectInstanceHandle theObject,
    byte[] userSuppliedTag,
    OrderType sentOrdering,
    LogicalTime theTime,
    OrderType receivedOrdering)
throws ObjectInstanceNotKnown,
    FederateInternalError
{
    if (null == validateRemoveObjectInstance)
        validateRemoveObjectInstance.validate(
            theObject,
            userSuppliedTag,
            sentOrdering,
            theTime,
            receivedOrdering);
    postEvent(evHLAremoveObjectInstance.create(
        _identifier,
        theObject,
        userSuppliedTag,
        sentOrdering,
        theTime,
        receivedOrdering,
        null)
    ));
} //removeObjectInstance
public void removeObjectInstance(
   ObjectInstanceHandle theObject,
   byte[] userSuppliedTag,
   OrderType sentOrdering,
   LogicalTime theTime,
   OrderType receivedOrdering,
   MessageRetractionHandle retractionHandle)
   throws ObjectInstanceNotKnown,
              InvalidLogicalTime,
              FederateInternalError
{
   if (null != validateRemoveObjectInstance)
      validateRemoveObjectInstance.validate(
         theObject,
         userSuppliedTag,
         sentOrdering,
         theTime,
         receivedOrdering,
         retractionHandle);
   postEvent(evHLAremoveObjectInstance.create(
         _identifier,
         theObject,
         userSuppliedTag,
         sentOrdering,
         theTime,
         receivedOrdering,
         retractionHandle));

} //removeObjectInstance

public void attributesInScope(
   ObjectInstanceHandle theObject,
   AttributeHandleSet theAttributes)
   throws ObjectInstanceNotKnown,
              AttributeNotRecognized,
              AttributeNotSubscribed,
              FederateInternalError
{
   if (null != validateAttributesInScope)
      validateAttributesInScope.validate(
         theObject,
         theAttributes);
   postEvent(evHLAattributesInScope.create(
         _identifier,
         theObject,
         theAttributes));

} //attributesInScope
public void attributesOutOfScope(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotSubscribed,
    FederateInternalError
{
    if (null != validateAttributesOutOfScope)
        validateAttributesOutOfScope.validate(
            theObject,
            theAttributes);
    postEvent(evHLAattributesOutOfScope.create(
        _identifier,
        theObject,
        theAttributes)
    );
} // attributesOutOfScope

public void provideAttributeValueUpdate(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    FederateInternalError
{
    if (null != validateProvideAttributeValueUpdate)
        validateProvideAttributeValueUpdate.validate(
            theObject,
            theAttributes,
            userSuppliedTag);
    postEvent(evHLAprovideAttributeValueUpdate.create(
        _identifier,
        theObject,
        theAttributes,
        userSuppliedTag)
    );
} // provideAttributeValueUpdate
public void turnUpdatesOnForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        FederateInternalError
{
    if (null != validateTurnUpdatesOnForObjectInstance)
        validateTurnUpdatesOnForObjectInstance.validate(
            theObject,
            theAttributes);
    postEvent(evHLAturnUpdatesOnForObjectInstance.create(
        _identifier,
        theObject,
        theAttributes));
}

// turnUpdatesOnForObjectInstance

public void turnUpdatesOffForObjectInstance(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        FederateInternalError
{
    if (null != validateTurnUpdatesOffForObjectInstance)
        validateTurnUpdatesOffForObjectInstance.validate(
            theObject,
            theAttributes);
    postEvent(evHLAturnUpdatesOffForObjectInstance.create(
        _identifier,
        theObject,
        theAttributes));
}

// turnUpdatesOffForObjectInstance


public void requestAttributeOwnershipAssumption(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError
{
    if (null!=validateRequestAttributeOwnershipAssumption)
        validateRequestAttributeOwnershipAssumption.validate(
            theObject,
            offeredAttributes,
            userSuppliedTag);
    postEvent(evHLArequestAttributeOwnershipAssumption.create(
        _identifier,
        theObject,
        offeredAttributes,
        userSuppliedTag))
};
} //requestAttributeOwnershipAssumption

public void requestDivestitureConfirmation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet offeredAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeNotOwned,
    AttributeDivestitureWasNotRequested,
    FederateInternalError
{
    if (null!=validateRequestDivestitureConfirmation)
        validateRequestDivestitureConfirmation.validate(
            theObject,
            offeredAttributes);
    postEvent(evHLArequestDivestitureConfirmation.create(
        _identifier,
        theObject,
        offeredAttributes))
};
} //requestDivestitureConfirmation
public void attributeOwnershipAcquisitionNotification(
    ObjectInstanceHandle theObject,
    AttributeHandleSet securedAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAcquisitionWasNotRequested,
    AttributeAlreadyOwned,
    AttributeNotPublished,
    FederateInternalError
{
    if (null!=validateAttributeOwnershipAcquisitionNotification)
        validateAttributeOwnershipAcquisitionNotification.validate(
            theObject,
            securedAttributes,
            userSuppliedTag);
    postEvent(evHLAattributeOwnershipAcquisitionNotification.create(
        _identifier,
        theObject,
        securedAttributes,
        userSuppliedTag)
    );
} //attributeOwnershipAcquisitionNotification

public void attributeOwnershipUnavailable(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
    AttributeNotRecognized,
    AttributeAlreadyOwned,
    AttributeAcquisitionWasNotRequested,
    FederateInternalError
{
    if (null!=validateAttributeOwnershipUnavailable)
        validateAttributeOwnershipUnavailable.validate(
            theObject,
            theAttributes);
    postEvent(evHLAattributeOwnershipUnavailable.create(
        _identifier,
        theObject,
        theAttributes)
    );
} //attributeOwnershipUnavailable
public void requestAttributeOwnershipRelease(
    ObjectInstanceHandle theObject,
    AttributeHandleSet candidateAttributes,
    byte[] userSuppliedTag)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeNotOwned,
        FederateInternalError
{
    if (null != validateRequestAttributeOwnershipRelease)
    {
        validateRequestAttributeOwnershipRelease.validate(
            theObject,
            candidateAttributes,
            userSuppliedTag);
        postEvent(evHLArequestAttributeOwnershipRelease.create(
            _identifier,
            theObject,
            candidateAttributes,
            userSuppliedTag));
    }
} //requestAttributeOwnershipRelease

public void confirmAttributeOwnershipAcquisitionCancellation(
    ObjectInstanceHandle theObject,
    AttributeHandleSet theAttributes)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        AttributeAlreadyOwned,
        AttributeAcquisitionWasNotCanceled,
        FederateInternalError
{
    if (null != validateConfirmAttributeOwnershipAcquisitionCancellation)
    {
        validateConfirmAttributeOwnershipAcquisitionCancellation.validate(
            theObject,
            theAttributes);
        postEvent(evHLAconfirmAttributeOwnershipAcquisitionCancellation.create(
            _identifier,
            theObject,
            theAttributes));
    }
} //confirmAttributeOwnershipAcquisitionCancellation
public void informAttributeOwnership(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute,
    FederateHandle theOwner)
throws ObjectInstanceNotKnown, AttributeNotRecognized, FederateInternalError
{
    if (null!=validateInformAttributeOwnership)
        validateInformAttributeOwnership.validate(
            theObject, theAttribute, theOwner);
    postEvent(evHLAinformAttributeOwnership.create(
        _identifier, theObject, theAttribute, theOwner));
}
//informAttributeOwnership

public void attributeIsNotOwned(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown, AttributeNotRecognized, FederateInternalError
{
    if (null!=validateAttributeIsNotOwned)
        validateAttributeIsNotOwned.validate(
            theObject, theAttribute);
    postEvent(evHLAattributeIsNotOwned.create(
        _identifier, theObject, theAttribute));
}
//attributeIsNotOwned
public void
attributeIsOwnedByRTI(
    ObjectInstanceHandle theObject,
    AttributeHandle theAttribute)
throws ObjectInstanceNotKnown,
        AttributeNotRecognized,
        FederateInternalError
{
    if (null != validateAttributeIsOwnedByRTI)
        validateAttributeIsOwnedByRTI.validate(
            theObject,
            theAttribute);
    postEvent(evHLAattributeIsOwnedByRTI.create(
        _identifier,
        theObject,
        theAttribute))
); //attributeIsOwnedByRTI

public void
timeRegulationEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
        NoRequestToEnableTimeRegulationWasPending,
        FederateInternalError
{
    if (null != validateTimeRegulationEnabled)
        validateTimeRegulationEnabled.validate(
            time);
    postEvent(evHLAtimeRegulationEnabled.create(
        _identifier,
        time))
); //timeRegulationEnabled

public void
timeConstrainedEnabled(
    LogicalTime time)
throws InvalidLogicalTime,
        NoRequestToEnableTimeConstrainedWasPending,
        FederateInternalError
{
    if (null != validateTimeConstrainedEnabled)
        validateTimeConstrainedEnabled.validate(
            time);
    postEvent(evHLAtimeConstrainedEnabled.create(
        _identifier,
        time))
); //timeConstrainedEnabled
public void
timeAdvanceGrant(
    LogicalTime theTime)
throws InvalidLogicalTime,
    JoinedFederateIsNotInTimeAdvancingState,
    FederateInternalError
{
    if (null!=validateTimeAdvanceGrant)
        validateTimeAdvanceGrant.validate(
            theTime);
    postEvent(evHLA.timeAdvanceGrant.create(
            identifier,
            theTime));
};
} //timeAdvanceGrant

public void
requestRetraction(
    MessageRetractionHandle theHandle)
throws FederateInternalError
{
    if (null!=validateRequestRetraction)
        validateRequestRetraction.validate(
            theHandle);
    postEvent(evHLA.requestRetraction.create(
            identifier,
            theHandle));
};
} //requestRetraction
} //HLAfederate

//end capHLA1516
The blfHLA belief set is a simple hash map used by the event-handling plans to retrieve the HLAfederate instance associated with the callback.

```java
// File: blfHLA.bel
package ca.gc.drdc_rddc.hla.rti1516.jack;

/**
 * Matches the HLAfederate identifiers with their instance references
 * The complex queries hasKey, get Instance and removeInstance all expect
 * an identifier String argument.
 * The first one returns true if a tuple featuring this identifier
 * exists, false otherwise.
 * The second one returns the HLAfederate instance matched to the
 * identifier, null otherwise.
 * The last one returns true if the tuple was removed, false otherwise.
 */

public beliefset
blfHLA
    extends ClosedWorld
{
    #key field String identifier;
    #value field Object instance;
    #indexed query get(String identifier, Object instance);
    #indexed query get(logical String identifier, Object instance);
    #indexed query get(String identifier, logical Object instance);
    #indexed query get(logical String identifier, logical Object instance);

    #function query public boolean hasKey(String identifier)
    {
        logical Object $some_instance;
        try
        {
            // Check whether or not there already exists an instance with
            // that identifier
            return get( identifier, $some_instance ).next();
        } catch (aos.jack.jak.beliefset.BeliefSetException bse) [
            bse.printStackTrace();
            //try
            return false;
        } //hasKey
```
#function query public capHLA1516.HLAfederate getInstance(String identifier)
{
    logical Object $some_instance;
    try
    {
        if (! get( identifier, $some_instance ).next() ) return null;
        return (capHLA1516.HLAfederate) ($some_instance.as_object());
    } catch (aos.jack.jak.beliefset.BeliefSetException bse) {
        bse.printStackTrace();
    } //try
    return null;
} //getInstance

#function query public boolean removeInstance(String identifier)
{
    logical Object $some_instance;
    try
    {
        if (get( identifier, $some_instance ).next())
        {
            remove(identifier, $some_instance.as_object());
            return true;
        } //if
    } catch (aos.jack.jak.beliefset.BeliefSetException bse) {
        bse.printStackTrace();
    } //try
    return false;
} //removeInstance

//end blfHLA
The 43 events all follow the same pattern: their fields consist of a String identifier and the callback’s parameters. They have just one constructor each. When a byte[] appears in the parameters, it is wrapped in an OMT HLAopaqueData to avoid JACK compilation errors.

`// File: evtHLAannounceSynchronizationPoint.event`  
`package ca.gc.drdc_rddc.hla.rti1516;`  

`import ca.gc.drdc_rddc.hla.rti1516.omt.*;`  
`import hla.rti1516.*;`  

`/**`  
`* Occurs when the RTI notifies the federate that a new synchronization point has been registered. Note that JACK baulks at byte[] fields, so we use an HLAopaqueData field to wrap the userSuppliedTag. */`  

`public event evtHLAannounceSynchronizationPoint extends MessageEvent {`  
`    public String identifier;`  
`    public String synchronizationPointLabel;`  
`    public HLAopaqueData userSuppliedTag;`  

`    #posted as`  
`    create(`  
`        String identifier,`  
`        String synchronizationPointLabel,`  
`        byte[] userSuppliedTag)`  
`    {`  
`        this.identifier = identifier;`  
`        this.synchronizationPointLabel = synchronizationPointLabel;`  
`        try {`  
`            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);`  
`        } catch (Exception ignored) {} //CouldNotDecode, null`  
`    }`  

`    userSuppliedTag`  
`}`  

`//end evtHLAannounceSynchronizationPoint`
// File: evtHLAattributeIsNotOwned.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI answers the federate’s QueryAttributeOwnership request.
 * The specified attribute is unowned.
 */
public event evtHLAattributeIsNotOwned extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandle theAttribute;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandle theAttribute)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttribute = theAttribute;
    }
}

//end evtHLAattributeIsNotOwned

// File: evtHLAattributeIsOwnedByRTI.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI answers the federate’s QueryAttributeOwnership request.
 * The specified attribute is owned by the RTI.
 */
public event evtHLAattributeIsOwnedByRTI extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandle theAttribute;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandle theAttribute)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttribute = theAttribute;
    }
}

//end evtHLAattributeIsOwnedByRTI
public event evtHLAattributeOwnershipAcquisitionNotification extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet securedAttributes;
    public HLAopaqueData userSuppliedTag;

    #posted as
create{
        String identifier, 
        ObjectInstanceHandle theObject, 
        AttributeHandleSet securedAttributes, 
        byte[] userSuppliedTag) {

            this.identifier = identifier;
            this.theObject = theObject;
            this.securedAttributes = securedAttributes;
            try {
                this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
            } catch (Exception ignored) {} //CouldNotDecode, null
            userSuppliedTag
        }
    }
} //end evtHLAattributeOwnershipAcquisitionNotification
// File: evtHLAattributeOwnershipUnavailable.event
package ca.gc.drdc.rddc.hla.rti1516.

import hla.rti1516.*

/**
 * Occurs when the RTI informs the federate that the specified attribute instances are not available for acquisition.
 */

public event evtHLAattributeOwnershipUnavailable extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

// File: evtHLAattributesInScope.event
package ca.gc.drdc.rddc.hla.rti1516.

import hla.rti1516.*

/**
 * Occurs when the RTI advises the federate that the specified attribute instances are now in scope.
 */

public event evtHLAattributesInScope extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

//end evtHLAattributesInScope
package ca.gc.drdc.rddc.hla.rti1516.
import hla.rti1516.*

/**
 * Occurs when the RTI advises the federate that the specified attribute instances are now out of scope.
 */

public event evtHLAattributesOutOfScope extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

//end evtHLAattributesOutOfScope

/**
 * Occurs when the RTI requests that the federate confirm its intent to cancel the pending acquisition of the specified instance attributes.
 */

public event evtHLAconfirmAttributeOwnershipAcquisitionCancellation extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

//end evtHLAconfirmAttributeOwnershipAcquisitionCancellation
package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the federate discovers a new object instance.
 */
public event evtHLAdiscoverObjectInstance extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public ObjectClassHandle theObjectClass;
    public String objectName;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        ObjectClassHandle theObjectClass,
        String objectName)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theObjectClass = theObjectClass;
        this.objectName = objectName;
    }
}

// File: evtHLAdiscoverObjectInstance.event

package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that the federation restore concluded unsuccessfully.
 */
public event evtHLAfederationNotRestored extends MessageEvent {
    public String identifier;
    public RestoreFailureReason reason;

    #posted as
    create(
        String identifier,
        RestoreFailureReason reason)
    {
        this.identifier = identifier;
        this.reason = reason;
    }
}

// File: evtHLAfederationNotRestored.event

public event evtHLAfederationNotSaved extends MessageEvent {
    public String identifier;
    public SaveFailureReason reason;

    //posted as
    create(
        String identifier,
        SaveFailureReason reason)
    {  
        this.identifier = identifier;
        this.reason = reason;
    }
}

public event evtHLAfederationRestoreBegun extends MessageEvent {
    public String identifier;

    //posted as
    create(
        String identifier)
    {  
        this.identifier = identifier;
    }
}

// File: evtHLAfederationNotSaved.event
package ca.gc.drdc_rddc.hla.rti1516;
import hla.rti1516.*;

/**
  Occurs when the RTI notifies the federate that the federation save concluded unsuccessfully.
  */

// File: evtHLAfederationRestoreBegun.event
package ca.gc.drdc_rddc.hla.rti1516;
import hla.rti1516.*;

/**
  Occurs when the RTI notifies the federate that the federation restore has begun.
  */
// File: evtHLAfederationRestored.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that the federation restore concluded successfully.
 */

public event evtHLAfederationRestored extends MessageEvent {
    public String identifier;

    #posted as
    create(
        String identifier)
    {
        this.identifier = identifier;
    }
}

//end evtHLAfederationRestored

// File: evtHLAfederationRestoreStatusResponse.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI responds to the federate’s query about the federation restore status.
 */

public event evtHLAfederationRestoreStatusResponse extends MessageEvent {
    public String identifier;
    public FederateHandleRestoreStatusPair[] response;

    #posted as
    create(
        String identifier,
        FederateHandleRestoreStatusPair[] response)
    {
        this.identifier = identifier;
        this.response = response;
    }
}

//end evtHLAfederationRestoreStatusResponse
// File: evtHLAfederationSaved.event
package ca.gc.drdc.rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that the federation save concluded successfully.
 */

public event evtHLAfederationSaved extends MessageEvent {
    public String identifier;

    #posted as
    create{
        String identifier)
        { this.identifier = identifier;
        }
    }
    //end evtHLAfederationSaved

// File: evtHLAfederationSaveStatusResponse.event
package ca.gc.drdc.rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI responds to the federate's query about the federation save status.
 */

public event evtHLAfederationSaveStatusResponse extends MessageEvent {
    public String identifier;
    public FederateHandleSaveStatusPair[] response;

    #posted as
    create{
        String identifier,
        FederateHandleSaveStatusPair[] response)
        { this.identifier = identifier;
          this.response = response;
        }
    }
    //end evtHLAfederationSaveStatusResponse
package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that the specified synchronization point has been achieved.
 */

public event evtHLAfederationSynchronized extends MessageEvent {
    public String identifier;
    public String synchronizationPointLabel;

    #posted as
    create(
        String identifier,
        String synchronizationPointLabel)
    {
        this.identifier = identifier;
        this.synchronizationPointLabel = synchronizationPointLabel;
    }
}
//end evtHLAfederationSynchronized

package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI answers the federate's QueryAttributeOwnership request.
 * The specified attribute is owned by the specified federate.
 */

public event evtHLAinformAttributeOwnership extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandle theAttribute;
    public FederateHandle theOwner;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandle theAttribute,
        FederateHandle theOwner)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttribute = theAttribute;
        this.theOwner = theOwner;
    }
}
//end evtHLAinformAttributeOwnership
import hla.rti1516.*;

/**
 Occurs when the RTI notifies the federate that it should start
 restoring its state.
 */

public event evtHLAinitiateFederateRestore extends MessageEvent {
    public String identifier;
    public String label;
    public FederateHandle federateHandle;

    #posted as
    create(String identifier,
            String label,
            FederateHandle federateHandle)
    {
        this.identifier = identifier;
        this.label = label;
        this.federateHandle = federateHandle;
    }
}
//end evtHLAinitiateFederateRestore

/**
 Occurs when the RTI notifies the federate that it should start saving
 its state.
 If a time is specified, that is the time at which the save should
 occur.
 */

public event evtHLAinitiateFederateSave extends MessageEvent {
    public String identifier;
    public String label;
    public LogicalTime time;

    #posted as
    create(String identifier,
            String label,
            LogicalTime time)
    {
        this.identifier = identifier;
        this.label = label;
        this.time = time;
    }
}
//end evtHLAinitiateFederateSave
// File: evtHLAObjectInstanceNameReservationFailed.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that its name reservation request failed.
 */
public event evtHLAObjectInstanceNameReservationFailed extends
MessageEvent {
    public String identifier;
    public String objectName;

    #posted as
    create(
        String identifier,
        String objectName)
    {
        this.identifier = identifier;
        this.objectName = objectName;
    }
}
//end evtHLAObjectInstanceNameReservationFailed

// File: evtHLAObjectInstanceNameReservationSucceeded.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that its name reservation request succeeded.
 */
public event evtHLAObjectInstanceNameReservationSucceeded extends
MessageEvent {
    public String identifier;
    public String objectName;

    #posted as
    create(
        String identifier,
        String objectName)
    {
        this.identifier = identifier;
        this.objectName = objectName;
    }
}
//end evtHLAObjectInstanceNameReservationSucceeded
package ca.gc.drdc_rddc.hla.rti1516.jack;

import ca.gc.drdc_rddc.hla.rti1516.omt.*;
import hla.rti1516.*;

/**
 * Occurs when the RTI requests an update of some of an object's attribute values.
 * Note that JACK baulks at byte[] fields, so we use an HLAopaqueData field to wrap the userSuppliedTag.
 */
public event evtHLAprovideAttributeValueUpdate extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;
    public HLAopaqueData userSuppliedTag;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes,
        byte[] userSuppliedTag)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
        try {
            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
        } catch (Exception ignored) {} //CouldNotDecode, null
    }
}

//end evtHLAprovideAttributeValueUpdate
// File: evtHLAreceiveInteraction.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import ca.gc.drdc_rddc.hla.rti1516.*;
import hla.rti1516.*;

/**
 * Occurs when an interaction happens. 
 * Note that JACK baulks at byte[] fields, so we use an HLAopaqueData field to wrap the userSuppliedTag.
 */
public event evtHLAreceiveInteraction extends MessageEvent {
    public String identifier;
    public InteractionClassHandle interactionClass;
    public ParameterHandleValueMap theParameters;
    public HLAopaqueData userSuppliedTag;
    public OrderType sentOrdering;
    public TransportationType theTransport;
    public LogicalTime theTime;
    public OrderType receivedOrdering;
    public MessageRetractionHandle messageRetractionHandle;
    public RegionHandleSet sentRegions;

    #posted as
    create(
        String identifier, 
        InteractionClassHandle interactionClass, 
        ParameterHandleValueMap theParameters, 
        byte[] userSuppliedTag, 
        OrderType sentOrdering, 
        TransportationType theTransport, 
        LogicalTime theTime, 
        OrderType receivedOrdering, 
        MessageRetractionHandle messageRetractionHandle, 
        RegionHandleSet sentRegions)
    {
        this.identifier = identifier;
        this.interactionClass = interactionClass;
        this.theParameters = theParameters;
        try {
            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
        } catch (Exception ignored) {} //CouldNotDecode, null
        userSuppliedTag
        this.sentOrdering = sentOrdering;
        this.theTransport = theTransport;
        this.theTime = theTime;
        this.receivedOrdering = receivedOrdering;
        this.messageRetractionHandle = messageRetractionHandle;
        this.sentRegions = sentRegions;
    }
}

//end evtHLAreceiveInteraction
// File: evtHLAreflectAttributeValues.event
package ca.gc.drdc.rddc.hla.rti1516.jack;

import ca.gc.drdc.rddc.hla.rti1516.omt.*;
import hla.rti1516.*;

/**
 * Occurs when the RTI supplies an update for a subscribed object's attributes.
 * Note that JACK baulks at byte[] fields, so we use an HLAopaqueData field to wrap the userSuppliedTag.
 */
public event evtHLAreflectAttributeValues extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleValueMap theAttributes;
    public HLAopaqueData userSuppliedTag;
    public OrderType sentOrdering;
    public TransportationType theTransport;
    public LogicalTime theTime;
    public OrderType receivedOrdering;
    public MessageRetractionHandle retractionHandle;
    public RegionHandleSet sentRegions;

    #posted as
    create {
        String identifier, ObjectInstanceHandle theObject, AttributeHandleValueMap theAttributes, byte[] userSuppliedTag, OrderType sentOrdering, TransportationType theTransport, LogicalTime theTime, OrderType receivedOrdering, MessageRetractionHandle retractionHandle, RegionHandleSet sentRegions
        {
            this.identifier = identifier;
            this.theObject = theObject;
            this.theAttributes = theAttributes;
            try {
                this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
            } catch (Exception ignored) {} //CouldNotDecode, null
            userSuppliedTag
            this.sentOrdering = sentOrdering;
            this.theTransport = theTransport;
            this.theTime = theTime;
            this.receivedOrdering = receivedOrdering;
            this.retractionHandle = retractionHandle;
            this.sentRegions = sentRegions;
        }
    }
    //end evtHLAreflectAttributeValues
// File: evtHLAremoveObjectInstance.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import ca.gc.drdc_rddc.hla.rti1516.omt.*;
import hla.rti1516.*;

/**
 * Occurs when the RTI informs the federate that an object instance has
 * been deleted.
 * Note that JACK baulks at byte[] fields, so we use an HLAopaqueData
 * field to wrap the userSuppliedTag.
 */
public event evtHLAremoveObjectInstance extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public HLAopaqueData userSuppliedTag;
    public OrderType sentOrdering;
    public LogicalTime theTime;
    public OrderType receivedOrdering;
    public MessageRetractionHandle retractionHandle;

    #posted as
    create(
        String identifier, 
        ObjectInstanceHandle theObject, 
        byte[] userSuppliedTag, 
        OrderType sentOrdering, 
        LogicalTime theTime, 
        OrderType receivedOrdering, 
        MessageRetractionHandle retractionHandle)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        try {
            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
        } catch (Exception ignored) {} //CouldNotDecode, null
        this.sentOrdering = sentOrdering;
        this.theTime = theTime;
        this.receivedOrdering = receivedOrdering;
        this.retractionHandle = retractionHandle;
    }
}

//end evtHLAremoveObjectInstance
public event evtHLArequestAttributeOwnershipAssumption extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet offeredAttributes;
    public HLAopaqueData userSuppliedTag;

    #posted as
    create {
        String identifier, 
        ObjectInstanceHandle theObject, 
        AttributeHandleSet offeredAttributes, 
        byte[] userSuppliedTag)

        this.identifier = identifier;
        this.theObject = theObject;
        this.offeredAttributes = offeredAttributes;
        try {
            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
        } catch (Exception ignored) {} //CouldNotDecode, null
        userSuppliedTag
    }
}
package ca.gc.drdc_rddc.hla.rti1516.jack;

import ca.gc.drdc_rddc.hla.rti1516.ont.*;
import hla.rti1516.*;

/**
 * Occurs when the RTI requests that the federate relinquish ownership of
 * the specified attribute instances.
 * Note that JACK baulks at byte[] fields, so we use an HLAopaqueData
 * field to wrap the userSuppliedTag.
 */
public event evtHLArequestAttributeOwnershipRelease extends
MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet candidateAttributes;
    public HLAopaqueData userSuppliedTag;

    #posted as
    create (String identifier,
            ObjectInstanceHandle theObject,
            AttributeHandleSet candidateAttributes,
            byte[] userSuppliedTag) {
        this.identifier = identifier;
        this.theObject = theObject;
        this.candidateAttributes = candidateAttributes;
        try {
            this.userSuppliedTag = new HLAopaqueData(userSuppliedTag);
        } catch (Exception ignored) {} //CouldNotDecode, null
        userSuppliedTag
    }
}
// File: evtHLArequestDivestitureConfirmation.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI requests that the federate confirm its intent to
 * divest itself of some attribute instances.
 */
public event evtHLArequestDivestitureConfirmation extends MessageEvent
{
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet offeredAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet offeredAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.offeredAttributes = offeredAttributes;
    }
}

//end evtHLArequestDivestitureConfirmation

// File: evtHLArequestFederationRestoreFailed.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI reports that the federate's specified federation
 * restore request completed unsuccessfully.
 */
public event evtHLArequestFederationRestoreFailed extends MessageEvent
{
    public String identifier;
    public String label;

    #posted as
    create(
        String identifier,
        String label)
    {
        this.identifier = identifier;
        this.label = label;
    }
}

//end evtHLArequestFederationRestoreFailed
// File: evtHLARequestFederationRestoreSucceeded.event
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI reports that the federate's specified federation
 * restore request completed successfully.
 */
public event evtHLARequestFederationRestoreSucceeded extends MessageEvent {
    public String identifier;
    public String label;

    #posted as
    create(
        String identifier,
        String label)
    {
        this.identifier = identifier;
        this.label = label;
    }
}
//end evtHLARequestFederationRestoreSucceeded

// File: evtHLARequestRetraction.event
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that the specified event has
 * been retracted.
 */
public event evtHLARequestRetraction extends MessageEvent {
    public String identifier;
    public MessageRetractionHandle theHandle;

    #posted as
    create(
        String identifier,
        MessageRetractionHandle theHandle)
    {
        this.identifier = identifier;
        this.theHandle = theHandle;
    }
}
//end evtHLARequestRetraction
package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are active subscribers for the specified object class.
 */
public event evtHLAstartRegistrationForObjectClass extends MessageEvent {
    public String identifier;
    public ObjectClassHandle theClass;

    #posted as
    create(
        String identifier,
        ObjectClassHandle theClass)
    {
        this.identifier = identifier;
        this.theClass   = theClass;
    }
}

package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are no active subscribers for the specified object class.
 */
public event evtHLAstopRegistrationForObjectClass extends MessageEvent {
    public String identifier;
    public ObjectClassHandle theClass;

    #posted as
    create(
        String identifier,
        ObjectClassHandle theClass)
    {
        this.identifier = identifier;
        this.theClass   = theClass;
    }
}

// File: evtHLAstartRegistrationForObjectClass.event
package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are active subscribers for the specified object class.
 */
public event evtHLAstartRegistrationForObjectClass extends MessageEvent {
    public String identifier;
    public ObjectClassHandle theClass;

    #posted as
    create(
        String identifier,
        ObjectClassHandle theClass)
    {
        this.identifier = identifier;
        this.theClass   = theClass;
    }
}

// File: evtHLAstopRegistrationForObjectClass.event
package ca.gc.drdc.rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are no active subscribers for the specified object class.
 */
public event evtHLAstopRegistrationForObjectClass extends MessageEvent {
    public String identifier;
    public ObjectClassHandle theClass;

    #posted as
    create(
        String identifier,
        ObjectClassHandle theClass)
    {
        this.identifier = identifier;
        this.theClass   = theClass;
    }
}
// File: evtHLAsynchronizationPointRegistrationFailed.event
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI signals that a synchronization point registration was unsuccessful.
 */
public event evtHLAsynchronizationPointRegistrationFailed extends MessageEvent {
    public String identifier;
    public String synchronizationPointLabel;
    public SynchronizationPointFailureReason reason;

    #posted as
    create(
        String identifier,
        String synchronizationPointLabel,
        SynchronizationPointFailureReason reason)
    {
        this.identifier = identifier;
        this.synchronizationPointLabel = synchronizationPointLabel;
        this.reason = reason;
    }
}

//end evtHLAsynchronizationPointRegistrationFailed

// File: evtHLAsynchronizationPointRegistrationSucceeded.event
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI signals that a synchronization point registration was successful.
 */
public event evtHLAsynchronizationPointRegistrationSucceeded extends MessageEvent {
    public String identifier;
    public String synchronizationPointLabel;

    #posted as
    create(
        String identifier,
        String synchronizationPointLabel)
    {
        this.identifier = identifier;
        this.synchronizationPointLabel = synchronizationPointLabel;
    }
}

//end evtHLAsynchronizationPointRegistrationSucceeded
// File: evtHLAtimeAdvanceGrant.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that it is has been granted
 * an advance to the specified time.
 */
public event evtHLAtimeAdvanceGrant extends MessageEvent {
    public String identifier;
    public LogicalTime time;

    #posted as
    create(
        String identifier,
        LogicalTime time)
    {
        this.identifier = identifier;
        this.time = time;
    }
}
//end evtHLAtimeAdvanceGrant

// File: evtHLAtimeConstrainedEnabled.event
package ca.gc.drdc_rddc.hla.rti1516.jack;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that it is time-constrained
 * as of the specified time.
 */
public event evtHLAtimeConstrainedEnabled extends MessageEvent {
    public String identifier;
    public LogicalTime time;

    #posted as
    create(
        String identifier,
        LogicalTime time)
    {
        this.identifier = identifier;
        this.time = time;
    }
}
//end evtHLAtimeConstrainedEnabled
// File: evtHLAtimeRegulationEnabled.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that it is time-regulating as of the specified time.
 */
public event evtHLAtimeRegulationEnabled extends MessageEvent {
    public String identifier;
    public LogicalTime time;

    #posted as
    create(
        String identifier,
        LogicalTime time
    )
    {
        this.identifier = identifier;
        this.time = time;
    }
}
//end evtHLAtimeRegulationEnabled

// File: evtHLAturnInteractionsOff.event
package ca.gc.drdc_rddc.hla.rti1516.

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are no active subscribers for the specified interaction class.
 */
public event evtHLAturnInteractionsOff extends MessageEvent {
    public String identifier;
    public InteractionClassHandle theHandle;

    #posted as
    create(
        String identifier,
        InteractionClassHandle theHandle
    )
    {
        this.identifier = identifier;
        this.theHandle = theHandle;
    }
}
//end evtHLAturnInteractionsOff
package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are active subscribers for the specified interaction class.
 */
public event evtHLAturnInteractionsOn extends MessageEvent {
    public String identifier;
    public InteractionClassHandle theHandle;

    #posted as
    create(
        String identifier,
        InteractionClassHandle theHandle
    )
    {
        this.identifier = identifier;
        this.theHandle = theHandle;
    }
}

package ca.gc.drdc_rddc.hla.rti1516;

import hla.rti1516.*;

/**
 * Occurs when the RTI notifies the federate that there are no active subscribers for the specified object instance attributes.
 */
public event evtHLAturnUpdatesOffForObjectInstance extends MessageEvent {
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes
    )
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

// File: evtHLAturnInteractionsOn.event

// File: evtHLAturnUpdatesOffForObjectInstance.event
public event evtHLAturnUpdatesOnForObjectInstance extends MessageEvent
{
    public String identifier;
    public ObjectInstanceHandle theObject;
    public AttributeHandleSet theAttributes;

    #posted as
    create(
        String identifier,
        ObjectInstanceHandle theObject,
        AttributeHandleSet theAttributes)
    {
        this.identifier = identifier;
        this.theObject = theObject;
        this.theAttributes = theAttributes;
    }
}

// end evtHLAturnUpdatesOnForObjectInstance
The 43 plans are all built from the same very simple pattern. For this reason, a template is provided below. The only variable is the `<callback_name>` term, which can be any of the corresponding 43 event names.

```java
// File: pln<callback_name>.plan
package ca.gc.drdc_rddc.hla.rti1516.jack;

/**
 * Null handler for the evt<callback_name>.
 * These plans are only meant to keep the run-time from complaining when
 * the agent using this capability is instantiated.
 */
public plan pln<callback_name>
    extends Plan
{
    #handles event evt<callback_name> ev;

    static boolean relevant(evt<callback_name> ev)
    {
        return false;
    }

    context()
    {
        true;
    }

    #reasoning method
    body()
    {
    }
}
//end pln<callback_name>
```
Part Two – The JACK Chat Client Proper

// File: HLAchat.prj
"ProjectName" node: HLAchat
  "Introduction" node
    (textual description of the chat client)
  "Design Views" node
    "Design Views (Federation Architecture)" node
      "Overview ChatRoomSlots" node
        (explanatory diagram and "Documentation" node)
      "Overview Interactions" node
        (explanatory diagram and "Documentation" node)
      "Overview Objects" node
        (explanatory diagram and "Documentation" node)
      "Overview UserHandleSlots" node
        (explanatory diagram and "Documentation" node)
    "Design Views (Application Architecture)" node
      "Application_Overview" node
        (explanatory diagram and "Documentation" node)
      "Data_Control" node
        (explanatory diagram and "Documentation" node)
        "Group_Who" diagram and "Documentation" node)
      "Joining_Group" diagram and "Documentation" node)
      "Leaving_Group" diagram and "Documentation" node)
      "Listing_Groups" diagram and "Documentation" node)
      "Logging_Out" diagram and "Documentation" node)
      "Logging_In" diagram and "Documentation" node)
      "ProcessChat" diagram and "Documentation" node)
      "ProcessData" diagram and "Documentation" node)
      "Sending_Message_Group" diagram and "Documentation" node)
      "Sending_Message_User" diagram and "Documentation" node)
      "StartClient" diagram and "Documentation" node)
      "StartServer" diagram and "Documentation" node)
  "Agent Model" node
    "Agent Types" node
      "Client"
      "Server"
    "Capability Types" node
      "capHLA1516"
      "capProcessChat"
      "capProcessData"
    "Plan Types" node
      "Plan Types - Client" node
        "plnProcessGrpNames"
        "plnProcessGrpWhoRes"
        "plnProcessJoinRes"
        "plnProcessLeaveGrpRes"
        "plnProcessLoginRes"
        "plnProcessMessageUsrRes"
        "plnProcessNoGrp"
        "plnProcessRelayMessg"
"Plan Types - HLA" node

"ChatRoom" node
- "plnHLA_ChatRoom_Discovery"
- "plnHLA_ChatRoom_OwnershipAcquisitionFailed"
- "plnHLA_ChatRoom_OwnershipAcquisitionNotification"
- "plnHLA_ChatRoom_OwnershipAssumptionRequest"
- "plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest"
- "plnHLA_ChatRoom_OwnershipReleaseRequest"
- "plnHLA_ChatRoom_ProvideAttributeValueUpdate"
- "plnHLA_ChatRoom_ReflectAttributeValueUpdate"
- "plnHLA_ChatRoom_Removal"
- "plnHLA_WaitingRoom_NameReservation_Failed"
- "plnHLA_WaitingRoom_NameReservation_Succeeded"

"ChatRoomRegistry" node
- "plnHLA_AcquireChatRoomRegistry"
- "plnHLA_ChatRoomRegistry_Discovery"
- "plnHLA_ChatRoomRegistry_NameReservation_Failed"
- "plnHLA_ChatRoomRegistry_NameReservation_Succeeded"
- "plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed"
- "plnHLA_ChatRoomRegistry_OwnershipAcquisitionNotification"
- "plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest"
- "plnHLA_ChatRoomRegistry_OwnershipDivestitureConfirmationRequest"
- "plnHLA_ChatRoomRegistry_OwnershipReleaseRequest"
- "plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate"
- "plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate"

"Interaction" node
- "plnHLA_Interaction"
- "plnHLA_InteractionScopeAdvisory_Off"
- "plnHLA_InteractionScopeAdvisory_On"

"Participant" node
- "plnHLA_Participant_AttributeScopeAdvisory_In"
- "plnHLA_Participant_AttributeScopeAdvisory_Out"
- "plnHLA_Participant_Discovery"
- "plnHLA_Participant_NameReservation_Failed"
- "plnHLA_Participant_NameReservation_Succeeded"
- "plnHLA_Participant_OwnershipAcquisitionFailed"
- "plnHLA_Participant_OwnershipAcquisitionNotification"
- "plnHLA_Participant_OwnershipAssumptionRequest"
- "plnHLA_Participant_OwnershipDivestitureConfirmationRequest"
- "plnHLA_Participant_OwnershipReleaseRequest"
- "plnHLA_Participant_ProvideAttributeValueUpdate"
- "plnHLA_Participant_ReflectAttributeValueUpdate"
- "plnHLA_ParticipantEntersGeneralChatRoom"

"plnHLA_ForceDivest"

"Plan Types - Server" node
- "plnJoinGrp"
- "plnLeaveGrp"
- "plnListGrp"
- "plnLoginUser"
- "plnLogoutUser"
- "plnMessageGrp"
- "plnMessageUser"
- "plnRequestGrpWho"

"from [ca.gc.drdc_rddc.hla.rtl516.jack]* node
("pln<callback_name>" plans)
"Event Types" node
  "Event Types - Client" node
    *evtJoinGrp*
    *evtLeaveGrp*
    *evtListGrp*
    *evtLogout*
    *evtMessageGrp*
    *evtMessageUsr*
    *evtRequestGrpWho*
    *evtUserLogin*
  "Event Types - HLA HLA node
    *evtHLA_AcquireChatRoomRegistry*
    *evtHLA_ForceDivest*
    *evtHLA_ParticipantEntersGeneralChatRoom*
  "Event Types - Server" node
    *evtGrpName*
    *evtGrpWhoRes*
    *evtJoinRes*
    *evtLeaveGrpRes*
    *evtLoginRes*
    *evtMessageUsrRes*
    *evtNoGrp*
    *evtRelayMessg*
    "from [ca.gc.drdc_rddc.rtdc.hla.rtl1516 ack]" node
      ("evt<callback_name> events"
       "Named Data" node
         *blfHLA datHLA*
         *blfUsers blfdatUsers"
         *viewKeyboard viewdatKeyboard"
       "Data Model" node
         *Beliefset Types" node
           *blfUsers"
             "from [ca.gc.drdc_rddc.rtdc.hla.rtl1516 ack]" node
               *blfHLA*
         "View Types" node
           *viewKeyboard"
         "External Classes" node
           (empty)
       "Other Files" node
         *server\MyObservable.java"
         *StartClient.java"
         *StartServer.java"
     //end HLAchat
The `StartClient` class is used to bootstrap the JACK Chat client. It is invoked simply by "java StartClient [<server_host>] <server_port>" where `<server_host>` defaults to "localhost". It connects the client to the server and then processes a minimal set of commands (quit/exit or login) until a login is successful, at which point the Client agent takes over.

```java
// File: StartClient.java
import aos.jack.jak.core.Dci;
import aos.jack.jak.core.DciException;
import aos.dci.Portal;
import aos.jack.Kernel;
import java.io.*;
import java.util.StringTokenizer;
import java.lang.Thread; // for yield()
import java.net.InetAddress; // for getting the host address
import java.net.UnknownHostException;
import java.util.Random;
import client.*;

public class StartClient {
    static BufferedReader buf;
    // static final int MAX_USER_PER_IP = 1000;

    public static void main(String[] args) {
        String host, port;

        args = Kernel.init(args);
        if ((args.length < 1) || (args.length > 2)) {
            System.err.println("Chat client usage: [<server_host>] <server_port>");
            System.exit(0);
        }

        System.out.println("Starting chat client...");
        System.out.println("Identifying local host...");
        if (args.length == 1) {
            host = "localhost";
            port = args[0];
        } else {
            host = args[0];
            port = args[1];
        }

        // Note that we can't create name inside the try block
        // (it becomes local to it) and that we must initialise it
        // to satisfy the compiler (which is worried we may escape
        // from the try/catch without setting 'name')
        String name = ";
```
try {
    /* Get the IP address of this machine and create a unique portal name for this process.
       The original code drew a random number for its portal and hoped for the best:
       name = (new Random()).nextInt(MAX_USER_PER_IP) + ":" +
              (InetAddress.getLocalHost()).getHostAddress();

       Here we use the System.currentTimeMillis() to improve our chances.
       I guess we could also use the hashCode() of a throwaway object.
       */
    name = System.currentTimeMillis() + ":" +
           (InetAddress.getLocalHost()).getHostAddress();
} catch (UnknownHostException uhe) {
    System.err.println(uhe);
    System.err.println("Unable to retrieve the host. Exiting...");
    System.exit(0);
}
System.out.println("Local host identified.");
System.out.println("Looking up chat server...");
try {
    /* Create a portal for this process with the name being the local host address with a pseudo-random number prefixed (see above).
       Since we have not provided a specific port number for the local portal, we will be given a random free port.
       */
    Dci.create(name, "");
    System.out.println("Local Portal created as \"" + name + "\"");
    System.out.println("Local Portal host is " + Dci.getHost());
    System.out.println("Local Portal port is " + Dci.getPort());
    /* Connect to the nameserver */
    String rdesc = (host + ":" + port);
    Dci.nameserver(rdesc);
    System.out.println("Nameserver address is \"" + rdesc + "\"");
}
// Try to ping the server agent. This will try a few times.
if (Dci.multiPingOk("ChatServer@ServerPortal"))
{
    System.out.println("Found Server agent ChatServer@ServerPortal");
    startChat();
} else {
    System.err.println("Chat Server agent doesn’t exist!");
}
} catch (DciException de) {
    System.err.println("DCI Exception caught!");
    System.err.println(de);
    System.exit(0);
}

public static void startChat()
{
    buf = new BufferedReader(new InputStreamReader(System.in));
    try {
        System.out.println("\n\n***************************\n" + "\n**\nWELCOME TO " + "AGENT-CHAT" + "**\n" + "\n***************************\n\n\n");
        String line = "";
        //This'll loop until a Quit/Exit or a successfull login do
        {
            System.err.print("ChatServer@ServerPortal: ");
            line = buf.readLine(); // read in commands.
        } while (!processLine(line));
        //Once out of this, the ProcessLogRes Plan will generate
        Events
        }
    } catch (IOException ioe)
    {
        System.err.println(ioe);
    }
}
public static boolean processLine(String line)
{
    StringTokenizer tokenizer = new StringTokenizer(line, " ");
    //blank command line?
    if ( !tokenizer.hasMoreTokens() ) return false;
    int numTokens = tokenizer.countTokens();
    String command = tokenizer.nextToken();

    if ( (command.equals("quit")) || (command.equals("exit")) )
    {
        //'Quit'/'Exit' command
        System.out.println("\nExiting...");
        System.exit(0);
        return true;
    }
    else if ( command.equals("login") && (numTokens == 2) )
    {
        //'Login <username>' command
        String username = tokenizer.nextToken();
        System.out.println("Creating Client " + username + ";");
        //Try launching a Client agent
        Client user = new Client(username);
        //The original code launched the login process like this:
        //user.send("ChatServer@ServerPortal", ((new UserLogin()) .login(username))
        //That line now appears in the Client's constructor

        //Wait for the login request to be processed
        do
        {
            Thread.yield();
        } while ( user.getLoginPendingStatus() );

        //Now it's safe to check the login status
        if (user.getLogStatus() == false)
        {
            System.out.println("Login failed.");
            return false;
        } else {
            //Login successful; exit the StartClient command loop
            System.out.println("Login successful.");
            return true;
        }
    } else {
        //Unrecognised command
        System.out.println("***Use either 'login <username>', 'quit'
or 'exit'\n      ");
        return false;
    }
}
//end StartClient
The StartServer class is used to launch the JACK Chat server. It is invoked simply by “java StartServer [<server_port>]”, where <server_port> defaults to "5000". It sets up the JACK networking required for Client-Server connections and serves to report Server activity.

```java
// File: StartServer.java
import aos.jack.jak.core.Dci;
import aos.jack.jak.core.DciException;
import aos.jack.Kernel;

import java.io.*;
import server.*;

public class StartServer
{
    static BufferedReader buf;

    public static void main(String[] args)
    {
        args = Kernel.init(args);

        String port;
        String[] HLAargs; //Will be passed to HLAmyChat
        //HLAargs is args.subarray(1);
        if (args.length >= 1)
        {
            port = args[0]; // grab port number from extra args
            //Oddly, there is no Array (or java.util) method
            //to extract a sub-array (like String.substring)
            HLAargs = new String[args.length - 1];
            for (int i = 1; i < args.length; i++) HLAargs[i - 1] = args[i];
        }
        else {
            port = "5000"; // use default port number
            HLAargs = new String[0]; //Not the same as null!
        }
    }
```
try {
    Dci.nameserver(port); // create a nameserver
    System.out.println("New 'nameserver' created.");
    //Dci.create may throw aos.jack.jak.core.DciException
    //but there is no known recovery method; the Dci service
    //is left in a bad state and we cannot just try creating
    //on a different port.
    try {
        Dci.create("ServerPortal", port); // create a new portal
    } catch (aos.jack.jak.core.DciException de) {
        System.err.println("+++ ServerPortal could not be created
     +++");
        System.err.println("+++ Try again with a different port
     +++");
        System.exit(0);
    }
    System.out.println("Server Portal Host: " + Dci.getHost());
    System.out.println("Server Port: " + Dci.getPort());
    Server server = new Server("ChatServer", HLAargs); // create server agent
    System.out.println("Server agent " + server.name() + " created.");
    System.out.println("+++ Server startup concluded +++");
    //The original code did not need any finalizer, so it just
    waited for the app to be interrupted:
    // System.out.println("Hit Ctrl-C to terminate.");
    //We cannot afford that, so we do this instead:
    try {
        //When run from the Compiler Utility, the JDE owns
        System.in, so this does not work
        System.err.println("Hit any key to terminate:
     ");
        buf = new BufferedReader(new InputStreamReader(System.in));
        String line = buf.readLine(); // read in any key
    } catch (IOException ioe) {
        System.err.println(ioe);
    }
// Shut down cleanly
// LogoutUsr plan will do this:
// HLAchat.finalize();
// Essentially, we need to post a LogoutAllUsersIGE
// whose plan, like MsgGrpPlan, retrieves all Client users
// and posts a Logout event for each.

System.out.println("+++ Server shut down +++");
System.exit(0);
} catch (DciException de) {
    System.err.println("DCI Exception caught!");
    System.err.println(de);
    System.exit(0);
}

// end StartServer
The Client agent was only modified from the AOS-supplied demonstration to the extent of fixing one broken line (the demonstration apparently predates version 4) and changing most identifiers to make them more intelligible. The line that sends the UserLogin event was also moved from StartClient to the Client constructor.

```java
// File: Client.agent
package client;

import java.io.*;
import server.evtGrpName;
import server.evtLeaveGrpRes;
import server.evtJoinRes;
import server.evtNoGrp;
import server.evtGrpWhoRes;
import server.evtMessageUsrRes;
import server.evtRelayMessage;
import server.evtLoginRes;

/**
 * The Client agent represents a single user within the chat network.
 */
public agent
Client
extends Agent
{
    #has capability capProcessChat cap;
    #handles event evtGrpName;
    #handles event evtGrpWhoRes;
    #handles event evtJoinRes;
    #handles event evtLeaveGrpRes;
    #handles event evtLoginRes;
    #handles event evtMessageUsrRes;
    #handles event evtNoGrp;
    #handles event evtRelayMessage;
    #sends event evtJoinGrp evJoinGrp;
    #sends event evtLeaveGrp evLeaveGrp;
    #sends event evtListGrp evListGrp;
    #sends event evtLogout evLogout;
    #sends event evtMessageGrp evMessageGrp;
    #sends event evtMessageUser evMessageUser;
    #sends event evtRequestGrpWho evRequestGrpWho;
    #sends event evtUserLogin evUserLogin;
    #uses plan plnProcessGrpNames;
    #uses plan plnProcessGrpWhoRes;
    #uses plan plnProcessJoinRes;
    #uses plan plnProcessLeaveGrpRes;
    #uses plan plnProcessLoginRes;
    #uses plan plnProcessMessageUsrRes;
    #uses plan plnProcessNoGrp;
    #uses plan plnProcessRelayMessage;
    #private data viewKeyboard viewdataKeyboard();
}```
// Final result of login request
private boolean logged = false;
// True while a login request is pending
private boolean login_pending = false;
// Current group to which the client belongs
private String group;
// Login name of the user
private String name;

public Client(String name)
{
    super(name);
    // Initiate login request
    setLoginPendingStatus(true);
    send("ChatServer@ServerPortal", evUserLogin.login(name));
}

public boolean getLoginPendingStatus()
{
    return login_pending;
}

public boolean getLogStatus()
{
    return logged;
}

String getGroup()
{
    return group;
}

String getUserName()
{
    // Was getName() but this now conflicts with an inherited final
    return name;
}

void setGroup(String group)
{
    this.group = group;
}

public void setLoginPendingStatus(boolean status)
{
    login_pending = status;
}

public void setLogStatus(boolean status)
{
    logged = status;
}
public capability capProcessChat extends Capability {
    #handles external event evtLoginRes;
    #sends event evtJoinGrp ev;
    #sends event evtLeaveGrp ev1;
    #sends event evtListGrp ev5;
    #sends event evtLogout ev6;
    #sends event evtMessageGrp ev4;
    #sends event evtMessageUsr ev2;
    #sends event evtRequestGrpWho ev3;
    #uses plan plnProcessLoginRes;
    #imports data viewKeyboard viewdatKeyboard();
}
//end capProcessChat
The `viewKeyboard` view wraps the keyboard primitives, turning command-line input into events sent to the Server. The `EventSource` interface mentioned here is not documented at all by AOS.

```java
// File: viewKeyboard.view
package client;

import java.util.StringTokenizer;
import aos.jack.jak.agent.Agent;

/**
 * This view wraps some keyboard primitives.
 */
public view viewKeyboard implements EventSource
{
    EventRecipient self;
    evtJoinGrp joinGrpEv;
    evtLeaveGrp leaveGrpEv;
    evtListGrp listGrpEv;
    evtMessageGrp msgGrpEv;
    evtMessageUsr msgUsrEv;
    evtRequestGrpWho grpWhoEv;

    public boolean attach(EventRecipient er)
    {
        self = er;
        joinGrpEv = (evtJoinGrp) er.findEvent(
            evtJoinGrp.class.getName() );
        leaveGrpEv = (evtLeaveGrp) er.findEvent(
            evtLeaveGrp.class.getName() );
        grpWhoEv = (evtRequestGrpWho) er.findEvent(
            evtRequestGrpWho.class.getName() );
        msgGrpEv = (evtMessageGrp) er.findEvent(
            evtMessageGrp.class.getName() );
        msgUsrEv = (evtMessageUsr) er.findEvent(
            evtMessageUsr.class.getName() );
        listGrpEv = (evtListGrp) er.findEvent(
            evtListGrp.class.getName() );
        return true;
    }
```
public MessageEvent processLine(String line)
{
    StringTokenizer tokenizer = new StringTokenizer(line, " ");
    String cmd = " ";
    if ( tokenizer.hasMoreTokens() ) cmd = tokenizer.nextToken();
    if ( cmd.equals("join") )
    {
        if ( tokenizer.hasMoreTokens() )
        {
            String group = tokenizer.nextToken();
            System.out.println("***Now joining group: " + group);
            return joinGrpEv.join(group, ((Client) self).getUserName());
        } else {
            System.err.println("***ERROR: Specify the group to join");
        }
    }
    else if ( cmd.equals("leave") )
    {
        if ( tokenizer.hasMoreTokens() )
        {
            String group = tokenizer.nextToken();
            System.out.println("***Now leaving group: " + group);
            return leaveGrpEv.leave( group, ((Client) self).getUserName() );
        } else {
            System.err.println("***ERROR: Specify the group to leave");
        }
    }
    else if ( cmd.equals("who") )
    {
        if ( tokenizer.hasMoreTokens() ) // 'who group_name'
        {
            String group = tokenizer.nextToken();
            return grpWhoEv.grpWho( group );
        } else {
            // 'who'
            return grpWhoEv.grpWho( ((Client) self).getGroup() );
        }
    }
    else if ( cmd.equals("msg") )
    {
        String msg = " ";
        while ( tokenizer.hasMoreTokens() )
        {
            msg += tokenizer.nextToken() + " ";
        }
        System.out.println("***Message was sent to the " + ((Client) self).getGroup() + " group.");
        return msgGrpEv.msgGrp( msg, ((Client) self).getGroup(), ((Client) self).getUserName() );
    }
}
else if (cmd.equals("msgusr")) {
    if (tokenizer.hasMoreTokens()) {
        String user = tokenizer.nextToken();
        String msg = "";
        // read in the message
        while (tokenizer.hasMoreTokens()) {
            msg += tokenizer.nextToken() + " ";
        }
        // send message to user
        return msgUsrEv.message(msg, user, ((Client) self).getUserName());
    } else {
        System.err.println("***ERROR: Specify the destination username");
    }
} else if (cmd.equals("group")) {
    return listGrpEv.list();
} else if (cmd.equals("help")) {
    System.out.println("\nCOMMANDS:");
    System.out.println("========");
    System.out.println("\nmsg <message_text> \t Send a message to everyone in the current group.");
    System.out.println("\nmsgusr <username> <message_text> \t Send a message to the user 'username'.");
    System.out.println("\nwho \t	\t Lists all users in the current group.");
    System.out.println("\nwho <group_name> \t Lists all users in the group 'group_name'.");
    System.out.println("\njoin <group_name> \t Join the group 'group_name'.");
    System.out.println("\nleave <group_name> \t Leave the group 'group_name' (Get removed from it)." );
    System.out.println("\ngroup \t	 Lists all the groups that exist.");
    System.out.println("\nlogout \t\t Logout (Exit program).\n");
} else if (cmd.equals("*")) {
    // nothing entered. Do nothing.
    return null;
} else {
    System.err.println("\n***ERROR: Unrecognised command. Type 'help' for a list of commands." );
    return null;
}
return null;

//end viewKeyboard
The **evtUserLogin** event represents a login request. The **user** is the requested user name.

```java
// File: evtUserLogin.event
package client;

/**
 * This event is sent by the Client to the Server, to request a login.
 */
public event
evtUserLogin
    extends MessageEvent
{
    public String username;

    #posted as
    login(String user)
    {
        username = user;
    }
}
//end evtUserLogin
```

The **evtLoginRes** event represents the outcome of the login request. The **status** indicates whether it was successful or not, **user** is the requested username and **location** the network-name of the originating JACK agent.

```java
// File: evtLoginRes.event
package server;

/**
 * This event is an envelope for the data that is sent back to the client
 * from the server, as a result of the login procedure.
 */
public event
evtLoginRes
    extends MessageEvent
{
    public boolean status;
    public String location;
    public String user;

    #posted as
    result(String username, String location, boolean logStatus)
    {
        user = username;
        this.location = location;
        status = logStatus;
    }
}
//end evtLoginRes
```
The `plnProcessLoginRes` plan processes the outcome of the login request. If a failure, it shuts down the agent, and control returns to the `StartClient` command-line loop. Otherwise, it starts prompting `viewKeyboard` to parse the user commands.

```java
// File: plnProcessLoginRes.plan
package client;

import java.io.*;
import java.util.StringTokenizer;
import server.*;
import server.evtLoginRes;

/**
 * This plan processes the outcome of the login request; if successful, it initiates the processing of the Client agent's chat commands.
 */
public plan plnProcessLoginRes
  extends Plan
{
  public static BufferedReader buf = new BufferedReader( new InputStreamReader( System.in ) );
  #handles event evtLoginRes logResEvi
  #sends event evtJoinGrp joinGrpEvi;
  #sends event evtLeaveGrp leaveGrpEvi;
  #sends event evtListGrp listGrpEvi;
  #sends event evtLogout logoutEvi;
  #sends event evtMessageGrp msgGrpEvi;
  #sends event evtMessageUsr msgUsrEvi;
  #sends event evtRequestGrpWho grpWhoEvi;
  #uses interface Client self;
  #uses data viewKeyboard viewdatKeyboard;

  static boolean relevant(evtLoginRes ev)
  {
    return true;
  }

  context()
  {
    true;
  }
```
# reasoning method

body()
{
    self.setLogStatus(logResEv.status);
    self.setLoginPendingStatus(false);
    self.setName(logResEv.user);

    if (logResEv.status == true)
    {
        self.setGroup("General");

        System.out.println("\n\n------------------------------------
System.out.println("Welcome to the II\"General\" chat
IIGeneral II group!
\n\n")
        System.out.println("------------------------------------
\n\n")
        //See below for an explanation why this line is no good:
        // System.out.println( (logResEv.getAgent()).name() + "; ");
        // System.out.println("***You are "+
(llogResEv.getAgent()).name() + "; ";
        System.out.println("***You are "+self.getUserName() + "; ");
        String line = ";
    @action()
    {
        try
        {
            line = buf.readLine();
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```java
StringTokenizer tokenizer = new StringTokenizer(line, " ");
boolean logout = false;

while ( !line.equals("logout") )
{
    MessageEvent me = vewdatKeyboard.processLine(line);
    if ( me != null ) @send ( logResEv.from, me );

    // There is no good way to have this chat application
    // write a prompt line. This old code:
    // System.out.print( (logResEv.getAgent()).name() + ": ");
    // does not work because it writes the prompt, goes into
    // buffer read-line mode, and *then* the other plan threads
    // spit out various feedback, which gets tacked at the end
    // of the prompt as a result. Maybe we could implement

    // some
    // kind of critical-section semaphore or some such, but
    it's
    // not worth the effort. So we settle for this sort-of
    prompt:
    // System.out.println( "***You are " +
    (logResEv.getAgent()).name() + ": ***" );
    System.out.println( "***You are " + self.getUserName() +
    " ***" );

    @action()
    { try
        { line = buf.readLine();
            catch (IOException ioe) { System.err.println(ioe); }
        }
    }

    if ( line.equals("logout") )
    {
        @send( logResEv.from, logoutEv.logout(logResEv.user) );
        System.out.println( "***Now exiting..." );
        System.exit(0);
    }
    else{
        System.err.println("***ERROR: User " + logResEv.user + "
        already logged in...Please wait...");
        // Don't we need to kill the agent here? This was missing.
        self.finish();
    }
}
@end plnProcessLoginRes
```

The **evtJoinGrp** event represents a request to join (and create, if necessary) a group. The **group** is the name of the group being joined, and **user** the name of the user joining the group.

```java
// File: evtJoinGrp.event
package client;

/**
 * This event is sent from the Client to the Server agent as a request to join a group.
 */
public event evtJoinGrp
    extends MessageEvent
{
    public String group;
    public String user;

    #posted as
    join(String grp, String usr)
    {
        group = grp;
        user = usr;
    }
}
//end evtJoinGrp
```

The **evtLeaveGrp** event represents a request to leave a group. The **group** is the name of the group being left behind, and **user** the name of the user leaving the group.

```java
// File: evtLeaveGrp.event
package client;

/**
 * This event triggers the removal of the user (sender) from the group specified.
 */
public event evtLeaveGrp
    extends MessageEvent
{
    public String group;
    public String user;

    #posted as
    leave(String grp, String usr)
    {
        group = grp;
        user = usr;
    }
}
//end evtLeaveGrp
```
The `evtListGrp` event represents a request to list the extant groups. It has no members.

```java
// File: evtListGrp.event
package client;

/**
 * This event is sent from the Client agent to initiate the retrieval of the group names that exist in the GroupList beliefset.
 */
public event evtListGrp extends MessageEvent
{
    #posted as
    list() { ; }
}
//end evtListGrp
```

The `evtLogout` event indicates to the Server that the Client is withdrawing from the Chat network and shutting down. The `user` member is the name of the user shutting down.

```java
// File: evtLogout.event
package client;

/**
 * This event initiates the logout procedure.
 */
public event evtLogout extends MessageEvent
{
    public String user;

    #posted as
    logout(String username)
    {
        user = username;
    }
}
//end evtLogout
```
The `evtMessageGrp` represents a message being sent by the client to the members of a given group. The `fromUser` is the name of the user sending the message, the `group` is the target group, and `msg` is the message being sent.

```java
// File: evtMessageGrp.event
package client;

/**
 * This event is an envelope for the data (including the message) required to send a message from one user to all users of a particular group.
 */
public event evtMessageGrp extends MessageEvent {
    public String fromUser;
    public String group;
    public String msg;

    #posted as
    msgGrp(String msg, String grp, String user) {
        msg = msg;
        group = grp;
        fromUser = user;
    }
}
//end evtMessageGrp
```

The `evtMessageUsr` represents a private message being sent by the client to a specific other user. The `fromUser` is the name of the user sending the message, the `toUser` is the target user, and `msg` is the message being sent.

```java
// File: evtMessageUsr.event
package client;

/**
 * This event is an envelope for the data required (including the message) to send a message from one user to another.
 */
public event evtMessageUsr extends MessageEvent {
    public String fromUser;
    public String msg;
    public String toUser;

    #posted as
    message(String msg, String to, String from) {
        msg = msg;
        toUser = to;
        fromUser = from;
    }
}//end evtMessageUsr
```
**The `evtRequestGrpWho` event represents a request to list the currently visible other users (those in the same group). The `grp` is the group being queried (which could have been different from the current group in the original JACK demonstration).**

```java
// File: evtRequestGrpWho.event
package client;

/**
 * This event initiates the processing of the 'who' command.
 */
public class evtRequestGrpWho extends MessageEvent {
    public String grouPi = friend(evtRequestGrpWho);
    private
    #posted as
    grpWho(String grp)
    { group = grp;
    }
    //end evtRequestGrpWho
```
The `MyObservable` class is a descendant of `java.util.Observable` adapted for use with the `aos.jack.util.cursor.Change` class.

```java
// File: MyObservable.java
package server;

/*
 * MyObservable.java
 * A simple, usable Observable.
 */
public class MyObservable
    extends java.util.Observable
{
    // Stupidly, both clearChanged() and setChanged() are protected,
    // preventing use "out of the box" of java.util.Observable

    // Changing visibility from protected to public
    // Also changing behaviour somewhat:
    // if there are no observers yet, we simply flip to "changed";
    // otherwise we flip to "changed" and then notifyObservers
    // (and thus revert to unchanged).
    public void setChanged()
    {
        if (countObservers() > 0)
        {
            super.setChanged();
            notifyObservers();
        } else {
            super.setChanged();
        }
    }

    // Changing visibility from protected to public
    public void clearChanged()
    {
        super.clearChanged();
    }
}

// end MyObservable
```
The server agent was extensively rewritten to adapt JACK to an HLA context. It is fairly large because of the HLAchat inner class, which reproduces the lower-level functionality of the Java Chat client.

A second development cycle would consist in re-organizing the Java code to follow more closely the JACK structure, and in merging the JACK Client and Server into a single Chatter agent. The Java FederateAmbassador functionalities could be separated from the GUI-related code and encapsulated into a stand-alone class. MyChat could then declare an inner class that descends directly from this class, and the agent’s HLAchat inner class could do likewise. This would remove a large amount of code duplication.
package server;

import java.io.File;
import java.util.HashMap;
import java.util.Iterator;
import hla.rti1516.*;
import se.pitch.prti1516.RTI;
import ca.gc.drdc_rddc.hla.rti1516FedAmb.*;
import ca.gc.drdc_rddc.hla.rti1516.jack.capHLA1516;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAAttributeOwnershipAcquisitionNotification;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAAttributeOwnershipUnavailable;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAAttributesInScope;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAAttributesOutOfScope;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAObjectInstance;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAObjectInstanceNameReservationFailed;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAObjectInstanceNameReservationSucceeded;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAProvideAttributeValueUpdate;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAReceiveInteractions;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAReflectAttributeValues;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLARemoveObjectInstance;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLARequestAttributeOwnershipAssumption;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLARequestDivestitureConfirmation;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLATurnInteractionsOff;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLATurnInteractionsOn;
import client.evtUserLogin;

/**
 * The Server agent represents the central server which processes all data incoming from the users logged onto the chat network.
 */
public agent Server
    extends Agent
{
    #has capability capHLA1516 capHLA;
    #has capability capProcessData cap;
# handles event evtHLA_AcquireChatRoomRegistry;
# handles event evtHLA_ForceDivest;
# handles event evtHLA_ParticipantEntersGeneralChatRoom;
# handles event evtHLA_attributeOwnershipAcquisitionNotification;
# handles event evtHLA_attributeOwnershipUnavailable;
# handles event evtHLA_attributesInScope;
# handles event evtHLA_attributesOutOfScope;
# handles event evtHLA_discoverObjectInstance;
# handles event evtHLA_objectInstanceNameReservationFailed;
# handles event evtHLA_objectInstanceNameReservationSucceeded;
# handles event evtHLA_provideAttributeValueUpdate;
# handles event evtHLA_receiveInteraction;
# handles event evtHLA_removeObjectInstance;
# handles event evtHLA_requestAttributeOwnershipAssumption;
# handles event evtHLA_requestAttributeOwnershipRelease;
# handles event evtHLA_requestDivestitureConfirmation;
# handles event evtHLA_turnInteractionsOff;
# handles event evtHLA_turnInteractionsOn;
# handles event evtUserLogin;

# posts event evtHLA_AcquireChatRoomRegistry
evtHLA_AcquireChatRoomRegistry;
# posts event evtHLA_ForceDivest
evtHLA_ForceDivest;
# posts event evtHLA_ParticipantEntersGeneralChatRoom
evtHLA_ParticipantEntersGeneralChatRoom;

# sends event evtLoginRes evLoginRes;
# sends event evtRelayMessg evRelayMessg;

# uses plan plnHLA_AcquireChatRoomRegistry;
# uses plan plnHLA_ChatRoom_Discovery;
# uses plan plnHLA_ChatRoom_OwnershipAcquisitionFailed;
# uses plan plnHLA_ChatRoom_OwnershipAcquisitionNotification;
# uses plan plnHLA_ChatRoom_OwnershipAssumptionRequest;
# uses plan plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest;
# uses plan plnHLA_ChatRoom_OwnershipReleaseRequest;
# uses plan plnHLA_ChatRoom_ProvideAttributeValueUpdate;
# uses plan plnHLA_ChatRoom_ReflectAttributeValueUpdate;
# uses plan plnHLA_ChatRoom_Removal;

# uses plan plnHLA_ChatRoomRegistry_Discovery;
# uses plan plnHLA_ChatRoomRegistry_NameReservation_Failed;
# uses plan plnHLA_ChatRoomRegistry_NameReservation_Succeeded;
# uses plan plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed;
# uses plan plnHLA_ChatRoomRegistry_OwnershipAcquisitionNotification;
# uses plan plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest;
# uses plan plnHLA_ChatRoomRegistry_OwnershipDivestitureConfirmationRequest;
# uses plan plnHLA_ChatRoomRegistry_OwnershipReleaseRequest;
# uses plan plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate;
# uses plan plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate;
public Server(String name, String[] HLAargs) {
    super (name);
    //Note that super() must be the first statement, 
    //so storing the HLAargs can't occur until the super's 
    //consequences have run out (including the UserData View 
    //constructor).
    //This is another reason why having any constructor invoke 
    //_HLAchat.connect() is a bad idea.
    _HLAargs = HLAargs;
}

public Server(String name, String[] HLAargs, String[] HLAargs) {
    super (name);
    //Note that super() must be the first statement, 
    //so storing the HLAargs can't occur until the super's 
    //consequences have run out (including the UserData View 
    //constructor).
    //This is another reason why having any constructor invoke 
    //_HLAchat.connect() is a bad idea.
    _HLAargs = HLAargs;
}

private data blfHLA datHLA();
private data blfUsers blfdatUsers();
/** 
 * Server Agent constructor. 
 */

DRDC Valcartier TR 2007-412 1233
/**
   //Exposing the event factories could be done like this:
   public evtHLAattributeOwnershipAcquisitionNotification
   getEvtHLAattributeOwnershipAcquisitionNotification()
   {
       return eVHLAattributeOwnershipAcquisitionNotification;
   } //getEvtHLAattributeOwnershipAcquisitionNotification
   and so on...
   */
   //Exposing the inner class constructor
   //to circumvent a Plan pre-processor bug; once that is fixed,
   //we can replace calls to this with:
   //
   // self.new HLAchat(capHLA, the_username)
   // where self is a Server instance reference and
   // capHLA is the HLAFederate's enclosing capHLA1516 instance.
   public HLAchat
   newHLAchat(String the_username)
   throws aos.jack.jak.beliefset.BeliefSetException
   {
       return this.new HLAchat(capHLA, the_username);
   } //newHLAchat

   // #################################################################
   // # HLChat #
   // #################################################################
   */

   * HLChat.java
   *
   * Based on the NetBeans MyChat.java application, this handles the
   * HLA interface of the JACK chat federation.
   * Besides the chat-specific stuff, the key interface layer here
   * consists of the FedAmbWrapper listener/responder stubs, which
   * simply rewrap the callback parameters into a JACK event.
   * An agent instance reference is needed for two main reasons:
   * 1) the agent supplies the postEvent and send methods
   * 2) the agent exposes (privately) the Event factories
   * (as long as the agent declares that it posts or sends them)
   * Because the Event factories are built by the pre-processor,
   * their accessibility cannot be modified directly here; the best
   * that could be done would be to provide an access method.
   * Having the HLChat class inner to the agent solves these
   * problems nicely.
   *
   * Note that the pRTI1516 classes dragged in by the RTIambassador
   * and FederateAmbassador turn out to be non-serializable (in their
   * innards), which means an HLChat class cannot be wrapped into a
   * sent message (although it could still be posted). Since the
   * agent must manage several HLChat instances in any case, putting
   * their references in a BeliefSet makes more sense. It becomes
   * relatively easy to recover the HLChat reference as part of each
   * plan's context() method.
   */
public class HLAchat
    extends ca.gc.drdc.rddc.hla.rti1516.jack.capHLA1516.HLAfederate
{
    public static final int CRC_PORT = 8989;
    public static final String _name_ChatRoomRegistry = "ChatRoomRegistry";
    public static final String _name_nowhere_ChatRoom = "nowhere";
    public static final String _name_waiting_room_ChatRoom = "waiting_room";
    public static final String _name_general_ChatRoom = "<General>";
    public static final short slot_nowhere_ChatRoom = 0;
    public static final short slot_waiting_room_ChatRoom = 1;
    public static final short slot_general_ChatRoom = 2;
    public static final short slot_FirstFreeChatRoomSlot = 3;

    private String rtiHost = "localhost";
    //HLAargs [0]
    private String fdd = "C:\Program Files\RTI1516\Samples\chat\MyChat.xml"; //HLAargs [1]
    private String fedex = "D:\Documents and Settings\dtibault\Mes Documents\Java Projects\MyChat.xml";
    //HLAargs [1]
    private String fedname = "MyChatRoom";
    //HLAargs [2]
    private String fedname = "MyChatter";
    //HLAargs [3]
    private FederateHandle _federateHandle;

    //References to the RTI and Federate ambassadors
    public RTIambassador _rtiAmbassador; //returns super.rtiAmbassador
    public FedAmbWrapper _fedAmbassador;

    //The JACK username
    //Used to identify the HLAchat instance
    //public String username; //use super.getIdentifier() instead

    //Whether HLA is connected or not (ie, whether the federate has joined)
    public boolean connected = false;

    public InteractionClassHandle _ich_ChatRoom;          
    public InteractionClassHandle _ich_Communication;      
    public ParameterHandle _iph_Communication_message;    
    public ParameterHandle _iph_Communication_sender;     

    public ObjectClassHandle _och_ChatRoomRegistry;       
    public ObjectClassHandle _och_ChatRoomRegistry_DeletePrivilege;
    public AttributeHandleSet _oahs_ChatRoomRegistry;     
    public AttributeHandleSet _oahs_ChatRoomRegistry_forUpdate;
    public AttributeHandle _oah_ChatRoomRegistry_list;     
    public AttributeHandleValueMap _oahvm_ChatRoomRegistry;
public ObjectClassHandle _och_ChatRoom;
public AttributeHandleSet _oahs_ChatRoom;
public AttributeHandleSet _oahs_ChatRoom_forUpdate;
public AttributeHandle _oah_ChatRoom_DeletePrivilege;
public AttributeHandle _oah_ChatRoom_slot;
public AttributeHandle _oah_ChatRoom_name;
// public AttributeHandleValueMap _oahVM_ChatRoom;

public ObjectClassHandle _oah_Participant;
public AttributeHandleSet _oahs_Participant;
public AttributeHandleSet _oahs_Participant_forUpdate;
// public AttributeHandle _oahs_Participant_chat_room_slot;
public AttributeHandle _oah_Participant_DeletePrivilege;
public AttributeHandle _oah_Participant_logged_in;
public AttributeHandle _oah_Participant_user_handle;
public AttributeHandle _oah_Participant_chat_room_slot;
// public AttributeHandleValueMap _oahVM_Participant;

public DimensionHandle _dch_UserHandleSlots;
public DimensionHandleSet _dchs_UserHandleSlotsSet;
public DimensionHandle _dch_ChatRoomSlots;
public DimensionHandleSet _dchs_ChatRoomSlotsSet;

//The handle of the Region matched to "_nowhere" (slot 0)
public RegionHandle _rh_nowhere_ChatRoom;
public RegionHandleSet _rhss_nowhere_ChatRoom;
//The handle of the Region matched to the "_waiting_room" ChatRoom (slot 1)
public RegionHandle _rh_waiting_room_ChatRoom;
public RegionHandleSet _rhss_waiting_room_ChatRoom;
//The handle of the Region matched to the "<General>" ChatRoom (slot 2)
public RegionHandle _rh_general_ChatRoom;
public RegionHandleSet _rhss_general_ChatRoom;
//The handle of the Region matched to our current ChatRoom, if any (null otherwise)
public RegionHandle _rh_current_ChatRoom;
public RegionHandleSet _rhss_current_ChatRoom;
//The handle of the Region matched to our current UserHandle, if any (null otherwise)
public RegionHandle _rh_myParticipantRegion;
public RegionHandleSet _rhss_myParticipantRegion;

//The list (size 2) of AttributeSet-RegionSet pairs used for Participant publish/subscribe
//The AttributeSet will always be _oahs_Participant_forUpdate;
//the RegionSet will always be one of _rh_nowhere_ChatRoom,
//_rh_waiting_room_ChatRoom or _rh_current_ChatRoom
public AttributeSetRegionSetPairList _asrspl_Participant_nowhere;
public AttributeSetRegionSetPairList _asrspl_Participant_waiting_room;
public AttributeSetRegionSetPairList _asrspl_Participant_current;
public boolean _me_logged_in = false;
public boolean _me_logging_in = false;
public boolean _alone = true;
public boolean _me_shutting_down = false;

// The unique ChatRoomRegistry object instance
public aChatRoomRegistry _chatRoomRegistry = null;

// The ChatRoomRegistry Semaphore, used in lieu of synchronized(_chatRoomRegistry)
public aos.jack.util.thread.Semaphore _sem_chatRoomRegistry =
new aos.jack.util.thread.Semaphore();

// The nowhere ChatRoom is never instantiated
// The waiting_room ChatRoom
final public aChatRoom _waiting_room_chatRoom = new aChatRoom(_name_waiting_room_chatRoom, _slot_waiting_room_chatRoom);
// The general ChatRoom
public aChatRoom _general_chatRoom; // = new aChatRoom(_name_general_chatRoom, _slot_general_chatRoom);
// The ChatRoom we're in
public aChatRoom _myChatRoom;

// The set of ChatRooms (keys are ObjectInstanceHandle, values are aChatRoom)
public HashMap _theChatRooms = new HashMap();
// The ChatRooms Semaphore, used in lieu of synchronized(_theChatRooms)
public aos.jack.util.thread.Semaphore _sem_theChatRooms =
new aos.jack.util.thread.Semaphore();
// Note that JACK usually defines a Semaphore at the Agent level,
// but in our case it must be at the HLAChat instance level

public aParticipant _me = new aParticipant();
// The _me Semaphore, used in lieu of synchronized(_me)
public aos.jack.util.thread.Semaphore _sem_me = new aos.jack.util.thread.Semaphore();
// List of known Participants (including ourselves); keys are ObjectInstanceHandles, values are aParticipant objects
public HashMap _theParticipants = new HashMap();
// The Participants Semaphore, used in lieu of synchronized(_theParticipants)
public aos.jack.util.thread.Semaphore _sem_theParticipants = new aos.jack.util.thread.Semaphore();
// Set of observable semaphores
// Name reservation semaphore
public MyObservable sem_name_reservation = new MyObservable();
// ChatRoomRegistry Discovery semaphore
public MyObservable sem_ChatRoomRegistry_discovery = new MyObservable();
// ChatRoomRegistry Acquisition semaphore
public MyObservable sem_ChatRoomRegistry_acquisition = new MyObservable();
// ChatRoomRegistry Divestiture semaphore
public MyObservable sem_ChatRoomRegistry_divestiture = new MyObservable();
// ChatRoom Discovery semaphore
public MyObservable sem_ChatRoom_discovery = new MyObservable();
// ChatRoom Acquisition semaphore
public MyObservable sem_ChatRoom_acquisition = new MyObservable();
// ChatRoom Divestiture semaphore
public MyObservable sem_ChatRoom_divestiture = new MyObservable();
// Participant Discovery semaphore
public MyObservable sem_Particpant_discovery = new MyObservable();
// Participant Acquisition semaphore
public MyObservable sem_Particpant_acquisition = new MyObservable();
// Participant Divestiture semaphore
public MyObservable sem_Particpant_divestiture = new MyObservable();
/**
 * Constructor for HLAchat.
 * Because HLAchat extends
 * ca.gc.drdc_rddc.hla.rti1516.jack.capHLA1516.HLAfederate,
 * the constructor needs a reference to the HLAfederate's
 * enclosing instance (the capability)
 */

public HLAchat(ca.gc.drdc_rddc.hla.rti1516.jack.capHLA1516 the_cap,
String the_username)
throws aos.jack.jak.beliefset.BeliefSetException
{
    // Until the pre-processor bug is fixed, we cannot do this:
    the_cap.super(the_username);
    // To get around the bug, compile with the above line
    // commented out, then edit Server.java to uncomment the line
    // and recompile Server.class.
    // super constructor registers the username with datHLA;
    // throws aos.jack.jak.beliefset.BeliefSetException if already
    in use
    System.out.println("HLAchat connecting...");
    connected = joinFederation();
    if (!connected) return;
    System.out.println("HLAchat connected...");
    if (!getHandles()) return;
    System.out.println("HLAchat handles obtained...");
    try {
        _ChatRoomRegistry = new aChatRoomRegistry(); // We couldn't
do this in the field declarations because of the escaping Exception
        catch (CouldNotDecode ignored) {} // try
        // We let the plans do the dispatching at the Relevance level,
        so we do:

        // Freeing up the other Semaphores
        _sem_chatRoomRegistry.signal();
        _sem_theChatRooms.signal();
        _sem_theParticipants.signal();
        _sem_me.signal();
        System.out.println("HLAchat initialisation complete");
    } // constructor

    private boolean joinFederation()
    {
        // Remaining initialisation
        try
        {
            // Process the command-line arguments, if any
            // if (Server.this._HLAargs != null)
            if (_HLAargs != null)
            {
                if (_HLAargs.length > 0) rtiHost = _HLAargs[0];
                if (_HLAargs.length > 1) fdd = _HLAargs[1];
                if (_HLAargs.length > 2) fedex = _HLAargs[2];
                if (_HLAargs.length > 3) fedname = _HLAargs[3];
            } // if
//Get the RTIambassador
try
{
  _rtiAmbassador = rtiAmbassador = RTI.getRTIambassador(
  rtiHost, CRC_PORT);
} catch (Exception e) {
  System.out.println("+++ HLAchat: Unable to connect to CRC on " + rtiHost + ":" + CRC_PORT );
  return false;
} //try
System.out.println("HLAchat initialising - RTIambassador obtained");

//Destroy any lingering empty federation execution
try
{
  _rtiAmbassador.destroyFederationExecution( fedex );
  System.out.println("HLAchat initialising - Previous " + fedex + " federation destroyed");
} catch (FederatesCurrentlyJoined ignored) {
  catch (FederationExecutionDoesNotExist ignored) {
    //try

  //Create the federation execution
  final File fddFile = new File(fdd);
  try
  {
    _rtiAmbassador.createFederationExecution( fedex, fddFile.toURL() );
    System.out.println("HLAchat initialising - " + fedex + " federation created");
  } catch (FederationExecutionAlreadyExists ignored) {
    catch (CouldNotOpenFDD cnofl {
      System.out.println("+++ HLAchat: Could not open FDD "+ fddFile.getAbsolutePath().toString() + "");
      return false;
    } catch (ErrorReadingFDD erf) {
      System.out.println("+++ HLAchat: Corrupt FDD "+ fddFile.getAbsolutePath().toString() + "");
      return false;
    } //try

  //Join the federation execution
  federateHandle = _rtiAmbassador.joinFederationExecution( fedname, fedex, this, null );
  System.out.println("HLAchat initialising - Joined as " + _federateHandle);
  return true;
} catch (Exception e) {
  e.printStackTrace();
  return false;
} //try
} //joinFederation
public void resignFederationExecution()
{
    try
    {
        //Any remaining owned objects will be deleted (there
        //should be none when _alone, really) (well, maybe some Region
        //objects...)
        System.out.println("HLAchat shutting down - Resigning
        //federation");
        _rtiAmbassador.resignFederationExecution(ResignAction.DELETE_OBJECTS_THEN_DIVEST);
        //OwnershipAcquisitionPending, FederateOwnsAttributes,
        //FederateNotExecutionMember, RTIInternalError
        try
        {
            _rtiAmbassador.destroyFederationExecution( fedex );
            //FederatesCurrentlyJoined, FederationExecutionDoesNotExist, RTIInternalError
        } catch (FederatesCurrentlyJoined ignored) {
        }
        System.out.println("HLAchat shutting down - Federation "
        + fedex + " destroyed");
        connected = false;
    } catch (RTIexception e) {
        e.printStackTrace();
    } //try
} //resignFederationExecution

private boolean getHandles()
{
    try
    {
        //Obtain object/interaction and parameter/attribute
        handles
        _ich_Communication =
        _rtiAmbassador.getInteractionClassHandle("Communication");
        _ich_Communication_message =
        _rtiAmbassador.getParameterHandle(_ich_Communication, "message");
        _ich_Communication_sender =
        _rtiAmbassador.getParameterHandle(_ich_Communication, "sender");
_och_ChatRoomRegistry = _rtiAmbassador.getObjectClassHandle("ChatRoomRegistry");
_och_ChatRoomRegistry_DeletePrivilege = _rtiAmbassador.getAttributeHandle(_och_ChatRoomRegistry, RTI.PrivilegeToDeleteObjectName); //HLA.privilegeToDeleteObjectName
_och_ChatRoomRegistry_list = _rtiAmbassador.getAttributeHandle(_och_ChatRoomRegistry, "list");
//Build the attribute handle sets
_ohs_ChatRoomRegistry = _rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_ChatRoomRegistry.add(_och_ChatRoomRegistry_list);
_ohs_ChatRoomRegistry.add(_och_ChatRoomRegistry_DeletePrivilege);
_ohs_ChatRoomRegistry_addUpdate = _rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_ChatRoomRegistry_addUpdate.add(_ohs_ChatRoomRegistry_list);

_och_ChatRoom = _rtiAmbassador.getObjectClassHandle("ChatRoom");
_och_ChatRoom_DeletePrivilege = _rtiAmbassador.getAttributeHandle(_och_ChatRoom, RTI.PrivilegeToDeleteObjectName);
_och_ChatRoom_name = _rtiAmbassador.getAttributeHandle(_och_ChatRoom, "name");
_och_ChatRoom_slot = _rtiAmbassador.getAttributeHandle(_och_ChatRoom, "slot");
//Build the attribute handle sets
_ohs_ChatRoom = _rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_ChatRoom.add(_och_ChatRoom_DeletePrivilege);
_ohs_ChatRoom.add(_och_ChatRoom_name);
_ohs_ChatRoom.add(_och_ChatRoom_slot);
_ohs_ChatRoom_addUpdate = _rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_ChatRoom_addUpdate.add(_och_ChatRoom_name);
_ohs_ChatRoom_addUpdate.add(_och_ChatRoom_slot);

_och_Participant = _rtiAmbassador.getObjectClassHandle("Participant");
_och_Participant_DeletePrivilege = _rtiAmbassador.getAttributeHandle(_och_Participant, RTI.PrivilegeToDeleteObjectName);
_och_Participant_logged_in = _rtiAmbassador.getAttributeHandle(_och_Participant, "logged_in");
_och_Participant_user_handle = _rtiAmbassador.getAttributeHandle(_och_Participant, "user_handle");
_och_Participant_chat_room_slot = _rtiAmbassador.getAttributeHandle(_och_Participant, "chat_room_slot");
//Build the attribute handle sets
_ohs_Participant =
_rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_Participant.add(_ohs_Participant_DeletePrivilege);
_ohs_Participant.add(_ohs_Participant_logged_in);
_ohs_Participant.add(_ohs_Participant_user_handle);
_ohs_Participant.add(_ohs_Participant_chat_room_slot);
_ohs_Participant_forUpdate =
_rtiAmbassador.getAttributeHandleSetFactory().create();
_ohs_Participant_forUpdate.add(_ohs_Participant_logged_in);
_ohs_Participant_forUpdate.add(_ohs_Participant_user_handle);
_ohs_Participant_forUpdate.add(_ohs_Participant_chat_room_slot);
// _ohs_Participant_chat_room_slot =
_rtiAmbassador.getAttributeHandleSetFactory().create();
// _ohs_Participant_chat_room_slot.add(_ohs_Participant_chat_room_slot);

_dh_UserHandleSlots = _rtiAmbassador.getDimensionHandle("UserHandleSlots");
_dh_ChatRoomSlots = _rtiAmbassador.getDimensionHandle("ChatRoomSlots");
_dhs_UserHandleSlotsSet =
_rtiAmbassador.getDimensionHandleSetFactory().create();
_dhs_UserHandleSlotsSet.add(_dh_UserHandleSlots);
_dhs_ChatRoomSlotsSet =
_rtiAmbassador.getDimensionHandleSetFactory().create();
_dhs_ChatRoomSlotsSet.add(_dh_ChatRoomSlots);

_rhs_nowhere_ChatRoom =
_rtiAmbassador.getRegionHandleSetFactory().create();
_rh_nowhere_ChatRoom = _rtiAmbassador.createRegion(_dhs_ChatRoomSlotsSet);
// nowhere_ChatRoom is slot 0
_rhs_nowhere_ChatRoom.add(_rh_nowhere_ChatRoom);
_rtiAmbassador.commitRegionModifications(_rhs_nowhere_ChatRoom);

_rhs_waiting_room_ChatRoom =
_rtiAmbassador.getRegionHandleSetFactory().create();
_rh_waiting_room_ChatRoom = _rtiAmbassador.createRegion(_dhs_ChatRoomSlotsSet);
// waiting_room_ChatRoom is slot 1
_rtiAmbassador.setRangeBounds(_rh_waiting_room_ChatRoom, _dh_ChatRoomSlots, new RangeBounds(_slot_waiting_room_ChatRoom, 1 + _slot_waiting_room_ChatRoom));
_rhs_waiting_room_ChatRoom.add(_rh_waiting_room_ChatRoom);
_rtiAmbassador.commitRegionModifications(_rhs_waiting_room_ChatRoom);
// The General ChatRoom
  _rhs_general_ChatRoom = _rtiAmbassador.getRegionHandleSetFactory().create();
  _rh_general_ChatRoom = _rtiAmbassador.createRegion(_dhs_ChatRoomSlotsSet);
  // General ChatRoom is slot 2
  _rtiAmbassador.setRangeBounds(_rh_general_ChatRoom, _dh_ChatRoomSlots, new RangeBounds(_slot_general_ChatRoom, 1 + _slot_general_ChatRoom));
  _rhs_general_ChatRoom.add(_rh_general_ChatRoom);
  _rtiAmbassador.commitRegionModifications(_rhs_general_ChatRoom);

  /*
   * For chat room slot filtering, we associate all "forUpdate" attributes to _dh_ChatRoomSlots regions
   * We'd get InvalidRegionContext when subscribing if we included the DeletePrivilege
   */
  asrspl_Participant_nowhere = _rtiAmbassador.getAttributeSetRegionSetPairListFactory().create(1);
  asrspl_Participant_nowhere.add(new AttributeRegionAssociation(_oahs_Participant_forUpdate, _rhs_nowhere_ChatRoom));
  asrspl_Participant_waiting_room = _rtiAmbassador.getAttributeSetRegionSetPairListFactory().create(1);
  asrspl_Participant_waiting_room.add(new AttributeRegionAssociation(_oahs_Participant_forUpdate, _rhs_waiting_room_ChatRoom));
  // There'll be no Communication traffic through the waiting room; this is used only to find out
  // (through the Interaction Advisories) whether there are other federates or not.
  return true;
} catch (Exception e) {
  e.printStackTrace();
  return false;
} //try
} //getHandles
public void
doUnsubscribe()
{
    try
    {
        rtlAmbassador.unsubscribeInteractionClassWithRegions(
            _ich_Communication, _rhs_waiting_room_ChatRoom);
        rtlAmbassador.unsubscribeObjectClassAttributes(
            _och_ChatRoom, _oahs_ChatRoom_forUpdate);
        _rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(
            _och_Participant, _asrspl_Participant_waiting_room);
        rtlAmbassador.unsubscribeObjectClassAttributes(
            _och_ChatRoomRegistry, _oahs_ChatRoomRegistry_forUpdate);
    } catch (Exception e) {
        e.printStackTrace();
    } //try
} //doUnsubscribe

public void
doUnpublish()
{
    try
    {
        rtlAmbassador.unsubscribeInteractionClass(
            _ich_Communication);
        rtlAmbassadorunsubscribeObjectClassAttributes(
            _och_ChatRoom, _oahs_ChatRoom_forUpdate);
        rtlAmbassador.unsubscribeObjectClassAttributesWithRegions(
            _och_Participant, _asrspl_Participant_forUpdate);
        rtlAmbassador.unsubscribeObjectClassAttributes(
            _och_ChatRoomRegistry, _oahs_ChatRoomRegistry_forUpdate);
    } catch (Exception e) {
        e.printStackTrace();
    } //try
} //doUnpublish

//Should be called when shutting down (by the garbage collector)
public void finalize()
    throws Throwable
{
    super.finalize();
} //finalize
The various Seek utilities need to synchronize on the HashMap being searched; within a Java thread, this is done through the Semaphore.threadWait() method. Within a Plan, however, this is done through @wait_for(Semaphore.planWait()) and the two shall not mix. Although these methods are within a Java object, they are all called from within a Plan, so what shall we do? We could make them subtasks (i.e. stand-alone Plans with corresponding Events) but this seems too much overhead. So instead we move the synchronization outside the methods, to the invoking Plans.

There turns out to be much less need for synchronization anyway; since there is but one Server agent, it keeps only one Plan active at a time, and the default SimpleTaskManager takes a "depth-first" approach. This means a Plan is suspended only by a wait_for. FederateAmbassador callbacks are all sent to the Server's posting queue, so a certain sequentiality is maintained. The Java HLA chat agent was much harder to code because it involved a proliferation of threads.

This utility method finds the Participant object in the hashmap by its ObjectInstanceHandle.

@returns the sought Participant object, or null if not present

public aParticipant SeekParticipant(ObjectInstanceHandle theObject)
{
  return (aParticipant)_theParticipants.get(theObject);
} //SeekParticipant

This utility method finds the ChatRoom object in the hashmap by its ObjectInstanceHandle.

@returns the sought ChatRoom object, or null if not present

public aChatRoom SeekChatRoom(ObjectInstanceHandle theObject)
{
  return (aChatRoom)_theChatRooms.get(theObject);
} //SeekChatRoom
/**
 * This utility method finds the ChatRoom object in the hashmap by its name.
 * @param name a String specifying the ChatRoom.name sought
 * @return the sought ChatRoom object, or null if not present
 */
public aChatRoom SeekChatRoomByName(String name)
{
    aChatRoom _ChatRoom;
    for (Iterator i = _theChatRooms.entrySet().iterator();
        i.hasNext();)
    {
        _ChatRoom =
        (aChatRoom)((java.util.Map.Entry)i.next()).getValue();
        if (_ChatRoom.name.toString().equals(name)) return
    _ChatRoom;
    } //for
    return null;
} //SeekChatRoomByName

/**
 * This utility method finds the ChatRoom object in the hashmap by its slot.
 * @param slot a short specifying the ChatRoom.slot sought
 * @return the sought ChatRoom object, or null if not present
 */
public aChatRoom SeekChatRoomBySlot(short slot)
{
    aChatRoom _ChatRoom;
    for (Iterator i = _theChatRooms.entrySet().iterator();
        i.hasNext();)
    {
        _ChatRoom =
        (aChatRoom)((java.util.Map.Entry)i.next()).getValue();
        if (_ChatRoom.slot.getValue() == slot) return _ChatRoom;
    } //for
    return null;
} //SeekChatRoomBySlot
/**
 * This utility method finds the Participant object in the
 * hashmap by its name.
 * @param name a String specifying the Participant.name sought
 * @return the sought Participant object, or null if not present
 */
public aParticipant
SeekParticipantByName(String name)
{
    aParticipant _Participant;
    for (Iterator i = _theParticipants.entrySet().iterator();
i.hasNext();)
    {
        _Participant =
        (aParticipants)(java.util.Map.Entry)i.next().getValue();
        if (_Participant.name.toString().equals(name)) return
        _Participant;
    } //for
    return null;
} // SeekParticipantByName

/**
 * This utility method finds the Participant object in the
 * hashmap by its user_handle.
 * @param user_handle an int specifying the
 * Participant.user_handle sought
 * @return the sought Participant object, or null if not present
 */
public aParticipant
SeekParticipantByUserHandle(int user_handle)
{
    aParticipant _Participant;
    for (Iterator i = _theParticipants.entrySet().iterator();
i.hasNext();)
    {
        _Participant =
        (aParticipants)(java.util.Map.Entry)i.next().getValue();
        if (_Participant.user_handle.getValue() == user_handle)
        return _Participant;
    } //for
    return null;
} // SeekParticipantByUserHandle
public class aChatRoomRegistry {
    // Whether we own the object or not
    public boolean owned = false;
    // Whether we subscribe to the object class or not
    // (and therefore whether the value is up to date or not)
    public boolean subscribed = false;
    // Whether we are divesting the object or not (makes sense only if owned)
    public boolean divesting = false;
    // The ChatRoomRegistry object's name
    final public String name = _name_ChatRoomRegistry;
    // The ChatRoomRegistry object's handle
    public ObjectInstanceHandle handle = null;
    // The ChatRoomRegistry object's list field
    public HLAChatRoomRegistryEntries list;

    /**
     * Default constructor; the name is unique and the list pre-loaded with the "waiting_room" and <General> ChatRooms.
     */
    public aChatRoomRegistry() throws CouldNotDecode {
        list = new HLAChatRoomRegistryEntries();
        list.add( list.size(), new HLAChatRoomRegistryEntry( new HLAAunicodeString( _name_waiting_room_ChatRoom ), new HLAIinteger16BE( _slot_waiting_room_ChatRoom ) ) );
        list.add( list.size(), new HLAChatRoomRegistryEntry( new HLAAunicodeString( _name_general_ChatRoom ), new HLAIinteger16BE( _slot_general_ChatRoom ) ) );
    } //constructor

    /**
     * Exposing the constructor
     */
    public aChatRoomRegistry getNewChatRoomRegistry() {
        return new aChatRoomRegistry();
    } //getNewChatRoomRegistry
*/
/**
 * Used by various methods to remove a ChatRoom from the registry and update the latter.
 * Remember that the ChatRoomRegistry only maps ChatRoom names to slots; it does not link
 * to ChatRoom object instances at all.
 * @param slot a short specifying the ChatRoom entry to remove
 */

public void RemoveChatRoomAndUpdateRegistry(short slot)
{
    //Presumes _ChatRoomRegistry has been synchronized on
    try
    {
        boolean updateNeeded = false;
        if (slot <= _slot_general_ChartRoom) return; //waiting_room
        for (int i = 0; i < _ChatRoomRegistry.list.size(); i++)
        {
            if (slot == (_HLAChartRoomRegistryEntry)_ChatRoomRegistry.list.get(i).slot.getValue())
            {
                _ChatRoomRegistry.list.remove(i);
                updateNeeded = true;
                break;
            }
        }
        if (!updateNeeded) return;
        AttributeHandleValueMap _ahvm_ChartRoomRegistry =
            _rtiAmbassador.getAttributeHandleValueMapFactory().create(_oahs_ChartRoomRegistry_forUpdate.size());
        _ahvm_ChartRoomRegistry.put(_oah_ChartRoomRegistry_list,
        _ChatRoomRegistry.list.toByteArray());
        _rtiAmbassador.updateAttributeValues(_ChatRoomRegistry.handle, _ahvm_ChartRoomRegistry, null);
    } catch (Exception ignored) {
    }
    //RemoveChatRoomAndUpdateRegistry
}
/**
 * Used by various methods to add a ChatRoom to the registry and update the latter.
 * The ChatRoom is also added to _theChatRooms
 * @param name a String specifying the ChatRoom name to give to the new entry (it'll be prefixed by the method)
 * @return the newly created aChatRoom object (null in case of failure)
 */

public aChatRoom
AddChatRoomAndUpdateRegistry(String name)
{
    // Presumes _ChatRoomRegistry has been synchronized on
    try
    {
        // To allow user names and chat room names to be anything,
        // the latter will be prefixed by "p" and "c", respectively, whilst our reserved names
        // will be prefixed by "r"
        String newChatRoomName = "r" + name;
        aChatRoom ChatRoom = SeekChatRoomByName(newChatRoomName);
        if (ChatRoom != null) return null;
        // At this point we know the name is OK; now find a slot
        short candidate = _slot_FirstFreeChatRoomSlot;
        // Synchronization on _theChatRooms would normally be required but we'll omit it here
        for (; null != SeekChatRoomBySlot(candidate); candidate++)
        {
            // We'll use candidate for the slot value
            _ChatRoomRegistry.list.add(_ChatRoomRegistry.list.size(),
                new HLAChatRoomRegistryEntry( newChatRoomName, new HLAinteger16BE( candidate ) ) );
            AttributeHandleValueMap _ahvm_ChatRoomRegistry =
                _rtiAmbassador.getAttributeHandleValueMapFactory().create( _oahs_ChatRoomRegistry_forUpdate.size() );
            _ahvm_ChatRoomRegistry.put( _oah_ChatRoomRegistry_list,
                _ChatRoomRegistry.list.toByteArray() );
            _rtiAmbassador.updateAttributeValues( _ChatRoomRegistry.handle, _ahvm_ChatRoomRegistry, null );
            _ChatRoom = new aChatRoom( newChatRoomName, candidate );
            _ChatRoom.owned = _ChatRoom.subscribed = true;
            // synchronized(_theChatRooms)
            _theChatRooms.put(_ChatRoom.handle =
                _rtiAmbassador.registerObjectInstance( _och_ChatRoom ), _ChatRoom );
            // synchronized(_theChatRooms)
            return _ChatRoom;
        }
        catch (Exception ignored) {}
    }
    return null;
    // AddChatRoomAndUpdateRegistry
public class aChatRoom
{
    // Whether we own the object or not
    public boolean owned = false;
    // Whether we subscribe to the object class or not
    // (and therefore whether the value is up to date or not)
    public boolean subscribed = false;
    // Whether we are divesting the object or not (makes sense only if owned)
    public boolean divesting = false;
    // The object's handle
    public ObjectInstanceHandle handle = null;
    // The ChatRoom object's name field
    public HLAunicodeString name;
    // The ChatRoom object's slot field
    public HLAinteger16BE slot;

    /**
     * Constructs a chat room named "unknown" with slot -1 (an illegal value, really).
     *
     * @return void
     */
    public aChatRoom() {
        this("unknown", (short)-1);
    }
    //constructor

    /**
     * Constructs a chat room of the specified name and slot.
     * @param aName a String specifying the ChatRoom's name field
     * @param aSlot a Short specifying the ChatRoom's slot field
     * @return void
     */
    public aChatRoom(String aName, short aSlot) {
        slot = new HLAinteger16BE(aSlot);
        name = new HLAunicodeString(aName);
    }
    //constructor
} //aChatRoom
// Exposing the constructors (until the Plan pre-processor bug is fixed)
public aChatRoom
getNewChatRoom()
{
    return new aChatRoom("unknown", (short)-1);
} // getNewChatRoom

public aChatRoom
getNewChatRoom(String aName, short aSlot)
{
    return new aChatRoom(aName, aSlot);
} // getNewChatRoom

// **************************** aParticipant ****************************

public class aParticipant
{
    // Whether we own the object or not
    public boolean owned = false;
    // Whether we subscribe to the object class or not
    // (and therefore whether the value is up to date or not)
    public boolean subscribed = false;
    // Whether we are divesting the object or not (makes sense only if owned)
    public boolean divesting = false;
    // Whether the object is in scope or not (relevant only if owned)
    public boolean inscope = false;
    // (we're forced to do this instead of localDelete)
    public boolean logged_in;
    // The object's handle
    public ObjectInstanceHandle handle = null;
    // The object's name
    public HLAunicodeString name = new HLAunicodeString();
    // The logged-in field
    public HLAboolean logged_in;
    // The user handle field
    public HLAinteger32BE user_handle;
    // The current chat room slot field
    public HLAinteger16BE chat_room_slot;
/**
 * Default constructor; the name is empty, logged_in is
 * HLA false, user handle is 0 and chat_room_slot is 0 (waiting_room).
 */
public aParticipant()
{
    logged_in = new HLAboolean(false);
    user_handle = new HLAinteger32BE(0);
    chat_room_slot = new HLAinteger16BE(_slot_nowhere_ChatRoom);
} //constructor
} //aParticipant

//Exposing the constructor (until the Plan pre-processor bug is fixed)
public aParticipant
getNewParticipant()
{
    return new aParticipant();
} //getNewParticipant

// #######################################################################
// Listeners and Responders #######################################################################

/**
 * Where the Java application builds various FedAmb
 * Listener/Responder classes which the FedAmbWrapper dispatches
 * based on the callback's key parameter (typically the
 * interaction or object class), here we instead rely on the
 * generic listeners/responders which each post an event --the
 * FedAmbWrapper would have these set up using the null
 * discriminant. Dispatching now occurs at the plan level: each
 * HLA event is handled by a variety of plans, each with
 * different Relevance conditions (based on the event's key
 * field).
 */

/**
 * To invoke an inner constructor from outside the class,
 * you must use an instance of the enclosing class, like this:
 * instance_of_Server.new
 MyInteractionAdvisoryResponder(instance_of_HLAchat)
 */
Because the JACK event classes are public, in the same package, and expose publicly the posting methods, we could systematically have each handler do:

```java
_server.postEvent(new evtHLAInteractionScopeAdvisory().create(_HLAchat, <HLA arguments>));
```

...except that this causes a NullPointerException in the JACK posting internals.

Instead, we must have the Server agent "post" each event to start it to create the factories, and then add public methods to make these (private) factories available here:

```java
_server.postEvent(_server.getEvtHLAInteractionScopeAdvisory().create(_HLAchat, <HLA arguments>));
```

Alternately, any inner class of the agent can access the instances directly since private access extends to the enclosing class’ body (in this case the agent).

```
} //HLAchat
//end Server
```

The `blfUsers` belief set is used by the Server to map usernames to the Client agent instance names.

```
// File: blfUsers.bel
package server;

/**
 * The Users beliefset maps a username to the corresponding agent's full name (with the portal name).
 */
public beliefset blfUsers extends ClosedWorld {
    #key field String location;
    #key field String username;
    #indexed query get(String location, String username);
    #indexed query get(logical String location, String username);
    #indexed query get(logical String location, logical String username);
    #indexed query get(String location, logical String username);
}
//end blfUsers
```
The `capProcessData` capability regroups the events the Server receives from its Clients.

```java
// File: capProcessData.cap
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import client.evtJoinGrp;
import client.evtLeaveGrp;
import client.evtListGrp;
import client.evtLogout;
import client.evtMessageGrp;
import client.evtMessageUser;
import client.evtRequestGrpWho;

/**
 * This capability implements the Server agent's data processing.
 */
public capability capProcessData extends Capability
    {
    #handles external event evtJoinGrp;
    #handles external event evtLeaveGrp;
    #handles external event evtListGrp;
    #handles external event evtLogout;
    #handles external event evtMessageGrp;
    #handles external event evtMessageUser;
    #handles external event evtRequestGrpWho;
    #posts external event evHLA_AcquireChatRoomRegistry
    evHLA_AcquireChatRoomRegistry;
    #posts external event evHLA_ForceDivest evHLA_ForceDivest;
    #posts external event evJoinGrp evJoinGrp;
    #sends event evtGrpName evGrpName;
    #sends event evtGrpWhoRes evGrpWhoRes;
    #sends event evJoinRes evJoinRes;
    #sends event evLeaveGrpRes evLeaveGrpRes;
    #sends event evMessageUserRes evMessageUserRes;
    #sends event evNoGrp evNoGrp;
    #uses plan plnJoinGrp;
    #uses plan plnLeaveGrp;
    #uses plan plnListGrp;
    #uses plan plnLogoutUser;
    #uses plan plnMessageGrp;
    #uses plan plnMessageUser;
    #uses plan plnRequestGrpWho;
    #imports data blfHLA datHLA();
    #imports data blfUsers blfUsers();
}
//end capProcessData
```
The `evtGrpName` event represents a group name being sent to a Client in response to a request for a listing of current groups. The name is the group name instance carried.

```java
// File: evtGrpName.event
package server;

/**
 * This event is an envelope for the name of a group that currently exist in the chat network.
 */
public event evtGrpName
    extends MessageEvent
{
    public String name;

    #posted as
    group(String name)
    {
        this.name = name;
    }
}
//end evtGrpName
```

The `evtGrpWhoRes` event represents a user name being sent to a Client in response to a request for a listing of those users sharing the current group. The user is the user name instance carried.

```java
// File: evtGrpWhoRes.event
package server;

/**
 * This event is an envelope for the username retrieved by the GrpWhoPlan plan.
 */
public event evtGrpWhoRes
    extends MessageEvent
{
    public String user;

    #posted as
    result(String usr)
    {
        user = usr;
    }
}
//end evtGrpWhoRes
```
The `evtJoinRes` event represents the outcome of a user’s request to join a Chat group. The `group` is the name of the group the user wished to join; the `result` is the Boolean outcome.

```java
package server;

/**
 * This event is an envelope for the data that is sent back to the Client agent, from the server, as a result of the join group procedure. The data stored in this event indicates the success or failure of the procedure.
 */
public event evtJoinRes extends MessageEvent {
    public boolean result;
    public String group;

    #posted as
    result(String grp, boolean res)
    {
        group = grp;
        result = res;
    }
}
```

The `evtLeaveGrpRes` event represents the outcome of the user’s request to leave a Chat group. The `group` is the name of the group the user wished to leave; the `result` is the Boolean outcome.

```java
// File: evtLeaveGrpRes.event
package server;

/**
 * This event holds data which represents the success or failure of the removal of a user from a specified group. It stores this information in the form of a boolean to indicate the result and the name of the group.
 */
public event evtLeaveGrpRes
    extends MessageEvent
{
    public boolean result;
    public String group;

    #posted as
    result(boolean res, String grp)
    {
        result = res;
        group = grp;
    }
}
//end evtLeaveGrpRes
```
The **evtLoginRes** event represents the outcome of a user’s request to log into the Chat federation. The **status** is the Boolean outcome; the **location** is the name of the JACK Client agent; the **user** is the user name.

```java
// File: evtLoginRes.event
package server;

/**
   This event is an envelope for the data that is sent back to the client from the server, as a result of the login procedure.
*/
public event evtLoginRes
   extends MessageEvent
{
    public boolean status;
    public String location;
    public String user;

    #posted as
    result(String username, String location, boolean logStatus)
    {
      user = username;
      this.location = location;
      status = logStatus;
    }
}
//end evtLoginRes
```

The **evtMessageUsrRes** event represents the outcome of a request to send a private message to some other user. The **user** is the target user name; the **result** is the Boolean outcome.

```java
// File: evtMessageUsrRes.event
package server;

/**
   This event holds data which indicates whether or not the message was successfully sent to the user.
*/
public event evtMessageUsrRes
   extends MessageEvent
{
    public boolean result;
    public String user;

    #posted as
    result(boolean status, String user)
    {
      this.result = status;
      this.user = user;
    }
}
//end evtMessageUsrRes
```
The **evtNoGrp** event is sent to the Client when it requests a listing of users for a non-existent chat group. The **group** is the name of the requested group.

```java
// File: evtNoGrp.event
package server;

/**
 * This event is sent to the Client agent to indicate that no group exists with that name.
 */
public event evtNoGrp
    extends MessageEvent
{
    public String group;
    #posted as
    group(String grp)
    {
        group = grp;
    }
}
//end evtNoGrp
```

The **evtRelayMessg** event represents a message being relayed by the Server to the Client. Note that no distinction is made between public and private messages. The **messg** is the relayed message; the **fromUsr** is the originating user’s name.

```java
// File: evtRelayMessg.event
package server;

/**
 * This event is an envelope for the message being sent.
 */
public event evtRelayMessg
    extends MessageEvent
{
    public String fromUsr;
    public String messg;
    #posted as
    relay(String msg, String user)
    {
        messg = msg;
        fromUsr = user;
    }
}
//end evtRelayMessg
```
The `evtHLA_AcquireChatRoomRegistry` event represents the sub-task of acquiring the ChatRoomRegistry. It is posted by the Server to itself. The identifier indicates which HLAchat is involved.

```java
// File: evtHLA_AcquireChatRoomRegistry.event
package server;

/**
 * Occurs when a plan needs to subtask the acquisition of the ChatRoomRegistry.
 */
public event evtHLA_AcquireChatRoomRegistry extends Event
{
    public String identifier;
    
    #posted as
    create(String identifier)
    {
        this.identifier = identifier;
    }
}
@end evtHLA_AcquireChatRoomRegistry
```

The `evtHLA_ForceDivest` event represents the sub-task of aggressively divesting all of a federate's owned objects, as part of its log out procedure. It is posted by the Server to itself. The identifier indicates which HLAchat is involved.

```java
// File: evtHLA_ForceDivest.event
package server;

/**
 * Occurs when a JACK client wants to shutdown and must therefore divest itself of any owned objects.
 */
public event evtHLA_ForceDivest extends Event
{
    public String identifier;
    
    #posted as
    create(String identifier)
    {
        this.identifier = identifier;
    }
}
@end evtHLA_ForceDivest
```
The `evtHLA_ParticipantEntersGeneralChatRoom` event represents the sub-task of entering the Participant object in the General ChatRoom, as part of the log in procedure. It is posted by the Server to itself. The `identifier` indicates which HLAChat is involved.

```java
// File: evtHLA_ParticipantEntersGeneralChatRoom.event
package server;

/**
 * Handles the process of entering the (owned) Participant
 * into the General ChatRoom
 */
public event evtHLA_ParticipantEntersGeneralChatRoom
    extends Event
{
    public String identifier;

    #posted as
    create(String identifier)
    {
        this.identifier = identifier;
    }
}
```

//end evtHLA_ParticipantEntersGeneralChatRoom
The plnJoinGrp plan handles the process of joining a chat group.

// File: plnJoinGrp.java
package server;

import hla.rti1516.*;
import client.evtJoinGrp;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * This plan handles the evtJoinGrp event, sent by the Client when it
 * wants to Join a Group (ChatRoom).
 * It initially checks whether or not the group specified already exists.
 * If the group exists then the user is added to the group.
 * Otherwise a new group is created and the user is added to it.
 * In both cases, to follow the HLA chat rules we combine the actual join
 * with a Leave Group.
 */

public plan plnJoinGrp
    extends Plan

    { logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtJoinGrp ev;
    #posts event evtHLA_AcquireChatRoomRegistry
    evHLA_AcquireChatRoomRegistry;
    #sends event evtJoinRes evJoinRes;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant (evtJoinGrp ev)
    { return true;
    }

    context()
    {
     datHLA.get( ev.user, _HLA ) &&
     ( (HLA - (Server.HLAchat)_HLA.as_object()) != null);
    }
# reasoning method

body()
{
  // Because the viewKeyboard.processLine uses a StringTokenizer,
  // the ev.group String is guaranteed to be non-trivial
  // (not zero length, nor with leading or trailing blanks).
  // However, in the old JACK code, you could "Join" the default
  // ChatRoom ("General"); within the HLA chat rules, you cannot.
  // You use the Leave command for that.
  // Since the operation is essentially the same, we will change
  // the plnLeaveGrp so it simply posts an evtJoinGrp with
  // "<General>" as the target.
  // We could implement this by either:
  // 1) Adding a boolean field to the evtJoinGrp so we can
distinguish between Join and Leave; or
  // 2) Exploiting the known characteristics of ev.group by
  // sending a zero-length ev.group.
  // We cannot do 1) because that would entail changing the
  // viewKeyboard in the client package. Hence, we stipulate Leave
  // will post an evtJoinGrp with a zero-length ev.group
  // If joining the _<General> ChatRoom, no need to change
ChatRoomRegistry
  String _newChatRoomName;
  Server.HLAchat.aChatRoom _newChatRoom;
  // Whether the target ChatRoom existed already or not
  // (also used in the evtJoinRes fired back at the Client)
  boolean extantGroup;
  if ( ev.group.length() != 0 )
  {
    _newChatRoomName = "c" + ev.group;
    @wait_for ( HLA._sem_theChatRooms.planWait() );
    synchronized(_theChatRooms)
    {
      _newChatRoom = HLA.SeekChatRoomByName(_newChatRoomName);
    } //synchronized(_theChatRooms)
    HLA._sem_theChatRooms.signal();
    extantGroup = (null != _newChatRoom);
    // If not found, create it
    if (!extantGroup)
    {
      @subtask( evHLA_AcquireChatRoomRegistry.create( ev.user )
    } //HILA.username
    //
    synchronized(HLA.AcquireChatRoomRegistry())
    {
      _newChatRoom =
      HLA.AddChatRoomAndUpdateRegistry(ev.group);
      //_newChatRoomName.substring(1)
    } //synchronized(AcquireChatRoomRegistry())
    HLA._sem_ChatRoomRegistry.signal();
  } //if
  else {
    _newChatRoomName = HLA._name_general_ChatRoom;
    @wait_for ( HLA._sem_theChatRooms.planWait() );
    _newChatRoom = HLA._general_ChatRoom =
    HLA.SeekChatRoomById(_newChatRoomName);
    HLA._sem_theChatRooms.signal();
    extantGroup = (null != _newChatRoom);
//If not found, create it
if (!extantGroup)
{
    @subtask(evHLA_AcquireChatRoomRegistry.create(ev.user))
    //HLA.username
    //  synchronized(HLA.AcquireChatRoomRegistry())
    {
        //
        HLA._general_ChatRoom = HLA.new aChatRoom(
        HLA._name_general_ChatRoom, HLA._slot_general_ChatRoom);
        HLA._general_ChatRoom = HLA.getNewChatRoom(
        HLA._name_general_ChatRoom, HLA._slot_general_ChatRoom); //Awaiting bug fix
        HLA._general_ChatRoom.owned =
        HLA._general_ChatRoom.subscribed = true;
        //
        HLA._general_ChatRoom.divesting = false;
        @wait_for(HLA._sem_theChatRooms.planWait());
        //
        synchronized(_theChatRooms)
        {
            HLA._theChatRooms.put(HLA._general_ChatRoom.handle
            = HLA._rtiAmbassador.registerObjectInstance(HLA._och_ChatRoom),
            HLA._general_ChatRoom);
            //No need to add the General ChatRoom to the
            _ChatRoomRegistry, as it is already listed
            } //synchronized(_theChatRooms)
            HLA._sem_theChatRooms.signal();
        } //synchronized(AcquireChatRoomRegistry())
        HLA._sem_ChatRoomRegistry.signal();
        _newChatRoom = HLA._general_ChatRoom;
    } //if
} //if
//At this point, _newChatRoom exists

//Moving to the _newChatRoom
//The change of ChatRooms cannot be from nowhere or
//waiting_room; it has to be from an active ChatRoom, including
//the "<General>" ChatRoom.
HLA._rtiAmbassador.unassociateRegionsForUpdates(HLA._me.handle,
HLA._asrspl_Participant_current);
//At this point, _me.handle is only associated with
_asrspl_Participant_nowhere
//Unsubscribe from the current ChatRoom
HLA._ich_Communication.unsubscribeInteractionClassWithRegions(
    HLA._ich_Communication, HLA._rhs_current_ChatRoom);

//Look at the ChatRoom we're leaving to decide whether to delete it or not
int count = 0;
@wait_for{ HLA._sem_theParticipants.planWait() };

//synchronized(_theParticipants)
{
    Server.HLAchat.aParticipant _participant;
    for (java.util.Iterator i =
        HLA._theParticipants.values().iterator(); i.hasNext();)
    {
        _participant = (Server.HLAchat.aParticipant)i.next();
        //Out of scope Participants have unreliable attributes;
        //The only exceptions are owned Participants: any owned orphans can only be in the waiting_room
        //me isn't counted
        if (!(HLA._me.equals(_participant.handle)) &&
            (_participant.inscope) &&
            (HLA._me.chat_room_slot.equals(_participant.chat_room_slot)) )
            count++;
    } //for

    HLA._sem_theParticipants.signal();

    //This'll cause the Participants in our ChatRoom to go out of scope
    HLA._rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(
        HLA._och_Participant, HLA._asrspl_Participant_current);

    //What if someone logs into the ChatRoom after we've counted its Participants but before we actually delete it?

    Server.HLAchat.aChatRoom _previousChatRoom = HLA._myChatRoom;
    HLA._myChatRoom = _newChatRoom;
    //Modify asrspl and delete old region, unless it was the General ChatRoom
    RegionHandle _rh_previous_ChatRoom = HLA._rh_current_ChatRoom;
    // (RegionHandle)_rhs_current_ChatRoom.toArray()[0];
    HLA._asrspl_Participant_current.remove(0);
    HLA._rhs_current_ChatRoom.remove(_rh_previous_ChatRoom);
    if (!_rh_previous_ChatRoom.equals(HLA._rh_general_ChatRoom))
    {
        HLA._rtiAmbassador.deleteRegion(_rh_previous_ChatRoom);
    } //if
// New region
if (newChatRoomName.equals(HLA._name_general_ChatRoom))
{
    HLA._rh_current_ChatRoom = HLA._rh_general_ChatRoom;
} else {
    HLA._rh_current_ChatRoom = HLA._rtiAmbassador.createRegion(HLA._dhs_ChatRoomSlotSet);
    HLA._rtiAmbassador.setRangeBounds(HLA._rh_current_ChatRoom,
    HLA._dh_ChatRoomSlots, new RangeBounds(HLA._myChatRoom.slot.getValue(), 1 + HLA._myChatRoom.slot.getValue()));
    HLA._rhs_current_ChatRoom.add(HLA._rh_current_ChatRoom);
    HLA._rtiAmbassador.commitRegionModifications(HLA._rhs_current_ChatRoom); // Won't complain if there are no mods to commit
    // Regenerate asrspl
    HLA._asrspl_Participant_current.add(new AttributeRegionAssociation(HLA._oahs_Participant_forUpdate,
    HLA._rhs_current_ChatRoom));
    HLA._me.chat_room_slot.setValue(HLA._myChatRoom.slot.getValue());
    // Go back in
    // This'll reveal the participants in the new ChatRoom
    HLA._rtiAmbassador.subscribeObjectClassAttributesWithRegions(HLA._och_Participant, HLA._asrspl_Participant_current);
    // Re-open the communication channel for the new ChatRoom
    HLA._rtiAmbassador.subscribeInteractionClassWithRegions(HLA._ich_Communication, HLA._rhs_current_ChatRoom);
    // Reveal ourselves to the new ChatRoom
    HLA._rtiAmbassador.associateRegionsForUpdates(HLA._me.handle, HLA._asrspl_Participant_current);
/Delete old ChatRoom?
if (count <= 0) //Should be just ==0 but you never know (because of inscope, _me isn't counted)
{
  //We were the only Participant left in the old ChatRoom;
delete it as we leave (we must be the owner, obviously)
  @subtask( evHLA_AcquireChatRoomRegistry.create( ev.user ) );
  //HLA.username
  // synchronized(HLA.AcquireChatRoomRegistry())
  { 
    //Because we're the owner, we won't get a
    RemoveObjectInstance notification
    //The remove returns the removed value, which is
    _previousChatRoom
    @wait_for( HLA._sem_theChatRooms.planWait() );
    synchronized(_theChatRooms)
    { 
      HLA._rtiAmbassador.deleteObjectInstance(
          ((Server.HLAchat.aChatRoom)HLA._theChatRooms.remove(
            _previousChatRoom
          )) .handle, null );
      } //synchronized(_theChatRooms)
      HLA._sem_theChatRooms.signal();
      HLA.RemoveChatRoomAndUpdateRegistry(
        _previousChatRoom.slot.getValue() );
      } //synchronized(AcquireChatRoomRegistry())
      HLA._sem_theChatRooms.signal();
      } else if (_previousChatRoom.owned) {
      //There are other Participants but we were the owner:
      transfer ownership
      _previousChatRoom.divesting = true;
      HLA._rtiAmbassador.negotiatedAttributeOwnershipDivestiture(
        _previousChatRoom.handle, HLA._oaks_chatRoom, null );
      } //if
    // send an acknowledgement message event back to the user
    if (!newChatRoomName.equals(HLA._name_general_chatRoom))
    // if (!ev.group.equals(""))
    { 
      @send( ev.from, evJoinRes.result( ev.group, extantGroup ) );
      System.err.println( " (joining " + ev.user + ") to group "
        + ev.group + ")" ) ;
    } // When ev.group is ", this plan was invoked as a subtask of
    // plnLeaveGrp;
    // we must then defer to it to send the acknowledgement back
    // (since we've lost track of the ev.from)
  }
} //end plnJoinGrp

DRDC Valcartier TR 2007-412  1269
The `plnLeaveGrp` plan handles the overall process of removing a Client from its current group.

```java
// File: plnLeaveGrp.java
package server;

import client.evtJoinGrpi;
import client.evtLeaveGrpi;
import ca.gc.drdc_rddc.hla.rtl1516.jack.blfHLAi;

/**
 * This plan handles the `evtLeaveGrp` event, sent by the Client when it wants to Leave a Group (ChatRoom).
 * It removes the user from a specified group if the group exists. It also removes the group from the network if that group is empty.
 * Because of the HLA chat rules, the Client will always Leave its current group (because it can no longer belong to more than one group).
 */
public plan
plnLeaveGrp
extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    #handles event `evtLeaveGrp` ev;
    #posts event `evtJoinGrp` evJoinGrpi;
    #sends event `evtJoinRes` evJoinResi;
    #sends event `evtLeaveGrpRes` evLeaveGrpResi;
    #uses interface `Server` serveri;
    #reads data `blfHLA` datHLAi;

    static boolean relevant(evtLeaveGrp ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.user, _HLA ) &&
        ( HLA = (Server.HLAChat)_HLA.as_object() ) != null);
    }
```
#reasoning method

body()
{
    // We re-use the plnJoinGrp by sub-tasking
    // See if Leaving correct group (the current one)
    if (HLA._myChatRoom.name.toString().substring(1).equals(ev.group))
    {
        // Yes (treat as Joining the <General> ChatRoom)
        @subtask( evJoinGrp.join( "", ev.user ) );
        @send( ev.from, evLeaveGrpRes.result( true, ev.group ) );
        System.out.println( " (removing '" + ev.user + "' from group " + ev.group + ")" );
        @send( ev.from, evJoinRes.result( "<General>", true ) );
        System.out.println( " (joining '" + ev.user + "' to group '" + ev.group + ")" );
    }
    else {
        // No
        @send( ev.from, evLeaveGrpRes.result( false, ev.group ) );
        System.err.println( " (group '" + ev.group + ") incorrect" );
    }
}
//end plnLeaveGrp

The plnListGrp plan handles the overall process of giving the Client a list of all current groups.

// File: plnListGrp.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import client.evtListGrp;

/**
 * This plan handles the evtListGrp event, which occurs when the Client sends the "group" command.
 * Originally, it posted an Inference Goal Event (evtListGrpIGE) to retrieve all group names from blfdatGrpList.
 */
public plan
plnListGrp
extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    #handles event evtListGrp ev;
    #sends event evtGrpName evGrpName;
    #uses interface Server server;
    #reads data blfHLA datHLA;
static boolean relevant(evtListGrp ev)
{
    return true;
}

context()
{
    // Where most other events sent by the Client specify the
    // username (in the ev.user field, filled from ((Client)
    // self).getUserName()),
    // this evtListGrp does not. Luckily, we can recover it
    // from the ev.from, which will be of the form
    "username@portalname".
    datHLA.get( ev.from.substring(0, ev.from.indexOf("@")), _HLA )
    &
    ( _HLA = (Server.HLachat)_HLA.as_object() != null);
}

# reasoning method
body()
{
    System.out.println( " (retrieving group names) " );
    @wait_for{ HLA._sem_theChatRooms.planWait() ;
    {
        for (java.util.Iterator i =
            HLA._theChatRooms.values().iterator(); i.hasNext(); )
        {
            Server.HLachat.aChatRoom _ChatRoom =
                (Server.HLachat.aChatRoom)i.next();
            // Skip waiting room
            if ( !_ChatRoom.name.toString().equals( HL.
                name_waiting_room_chatRoom ) )
            {
                // Report General room differently
                if ( _ChatRoom.name.equals( HL.name_general_chatRoom
                    ) )
                {
                    @send{ ev.from, evGrpName.group( "General" ) };
                } else {
                    @send{ ev.from, evGrpName.group( _ChatRoom.name.toString().substring(1) } );
                }
                System.out.println( " (reporting group " +
                    _ChatRoom.name.toString() + ")" );
            }
        }
    } // synchronized (_theChatRooms)
    HLA._sem_theChatRooms.signal();
}
// end plnListGrp
The `plnLoginUser` plan handles the delicate process of logging a user in.

```java
// File: plnLoginUser.java
package server;

import java.io.*;
import java.util.StringTokenizer;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import client.evtUserLogin;

/**
 * This plan is responsible for adding a new user to the chat network.
 * It only adds the user if there is no user currently logged on with the
 * specified username.
 * */
public plan
plnLoginUser
extends Plan

{  
BufferedReader buf;
#handles event evtUserLogin ev;
#posts event evtHLA_ForceDivest evHLA_ForceDivest;
#sends event evtLoginRes evLoginRes;
#uses interface Server server;
#uses data blfHLA datHLA;
#uses data blfUsers blfdatUsers;

static boolean relevant(evtUserLogin ev)
{
  return true;
}

context()
{
  true;
}
```
# reasoning method
body()
{
    /* If another Client was created on the same host machine
       at the same time (to within 1 ms), it'll have the same
       portal name (ev.from) (see StartClient to see why).
       We could detect this rare occurrence but there's nothing we
       could do about it since any messages fired back at that
       portal would have only a 50% chance of reaching the right
       agent. This could be improved by using the username as a
discriminator, but what if they *also* try to log in
under the same name? */
    /*
    In the original code, the plan skipped straight to a
    vewdatUserData.addUser query, which decided whether or
    not the login was successful.
    With HLA, things are more complicated, so we must
    do things at the plan level before querying any
    Views or Belief Sets.
    */
    System.out.println("Logging user in as "+ev.username+"");

    //Create the HLAchat instance for the client
    Server.HLAchat HLA;
    try {
        // HLA = server.new HLAchat(ev.username);
        HLA = server.newHLAchat(ev.username);
    } catch (aos.jack.jak.beliefset.BeliefSetException e) {
        // Send a MessageEvent back to the client to indicate that
        // the login procedure was a failure.
        @send( ev.from, evLoginRes.result( ev.username, ev.from,
            false )
        );
        System.err.println("Couldn't log user in. HLA RTI connection
failed or username already logged in.");
        return true; //Plan succeeds (but login fails)
    }

    System.out.println(" HLA federate created");
    //HLAchat's constructor joins the federation and obtains the
    //various handles; it sets up its generic FederateAmbassador
    //callbacks and creates the ChatRoomRegistry Java object but
    //does not publish/subscribe anything.
    //
    //Note that if HLA isn't stored it'll go out of scope once this
    //plan concludes and will be garbage-collected. Its finalize()
    //method invokes disconnect() if it made it as far as joining
    //the federation.
    if (HLA.connected)
    {
        //We don't need to store a reference to the Server.HLAchat
        //instance because its constructor stores itself (at the
        //HLA1615. capHLA1516.HLAFederate level) in datHLA (and the
        //class of the stored instance is indeed HLAchat)
//Publish/Subscribe the interaction
HLA._ich_Communication.publishInteractionClass();
HLA._ich_Communication.subscribeInteractionClassWithRegions(
HLA._ich_Communication, HLA.rhs_waiting_room(ChatRoom));
//By subscribing through the waiting room only, we only
//trigger the advisories, without risking to get any
//Communications (because the waiting room is not a valid
//Communication "target").
//Note that at this point the waiting room ChatRoom may not
//exist yet.

//Set up ChatRoomRegistry, creating the HLA object if
necessary
HLA._rtiAmbassador.publishObjectClassAttributes(
HLA._och_ChatRoomRegistry, HLA._oahs_ChatRoomRegistry_forUpdate );
// Reserve name
HLA.sem_name_reservation.clearChanged();
//We must create the semaphore before calling
//reserveObjectInstanceName in the (unlikely) case that the
//nameReservationOutcome callback occurs before this task
//reaches the wait_for statement.
//With a single-threaded Agent, this cannot occur anyway.
System.out.println("Looking up Chat Room Registry...");
HLA._rtiAmbassador.reserveObjectInstanceName(
HLA._ChatRoomRegistry.name );
//Wait for reservation succeeded/failed
//The name reservation handler for ChatRoomRegistry takes
//care of creating the ChatRoomRegistry (if the reservation
//was successful) or of waiting for its discovery (if not)
//The flag takes care of the case where the callback occurs
before we reach the wait_for
@wait_for( new aos.jack.util.cursor.Change(
HLA.sem_name_reservation, HLA.sem_name_reservation.hasChanged() ) );
//Assert: At this point, HLA._ChatRoomRegistry is known

//Set up ChatRooms, creating the _waiting_room HLA object if
necessary
HLA._rtiAmbassador.publishObjectClassAttributes(
HLA._och_ChatRoom, HLA._oahs_ChatRoom_forUpdate );
// Reserve _waiting_room name
HLA.sem_name_reservation.clearChanged(); //Alternative to -
new MyObservable();
HLA._rtiAmbassador.reserveObjectInstanceName(
HLA._name_waiting_room_ChatRoom );
//Wait for reservation succeeded/failed
//The flag takes care of the case where the callback occurs
before we reach the wait_for
System.out.println("Looking up Waiting Room...");
@wait_for( new aos.jack.util.cursor.Change(
HLA.sem_name_reservation, HLA.sem_name_reservation.hasChanged() ) );
//Assert: At this point, HLA._waiting_room_ChatRoom is known
//There is a small chance that the waiting room's attributes
have
//not been updated yet, but that should sort itself out
pretty quickly.
//Setup Participants
HLA._rtiAmbassador.publishObjectClassAttributes(
    HLA._och_Participant, HLA._oahs_Participant_forUpdate);
//Note that associateRegionsForUpdates can be done only on a
per-instance basis
//We publish federation-wide but will update and subscribe
through DDM
//No federate *ever* subscribes to the nowhere "ChatRoom"
HLA._rtiAmbassador.subscribeObjectClassAttributesWithRegions(
    HLA._och_Participant, HLA._asrspl_Participant_waiting_room);

//Log in proper
HLA._me_logging_in = true;
HLA._me.name.setValue( "p" + ev.username ); //HLA.username
//HLA._me.handle = null; //ObjectInstanceHandle of the
Participant object
//HLA._me.user_handle.setValue(0); //Same value, as an HLAInt32
for DDM purposes
// Reserve username
HLA.sem_name_reservation.clearChanged(); //Alternative to =
new MyObservable();
HLA._rtiAmbassador.reserveObjectInstanceName(
    HLA._me.name.ToString() );
// Wait for reservation succeeded/failed
System.out.println( "Looking up '" + ev.username + "'..." );
//HLA.username
@wait_for( new aos.ack.util.cursor.Change( HLA.sem_name_reservation, HLA.sem_name_reservation.hasChanged() ) );

if (HLA._me_logged_in)
    {
    HLA._rtiAmbassador.subscribeInteractionClassWithRegions(
        HLA._ich_Communication, HLA._rhs_current_ChatRoom);
    HLA._rtiAmbassador.subscribeInteractionClassWithRegions(
        HLA._ich_Communication, HLA._rhs_myParticipantRegion);
    } else {
    //Log in failed, remove ourselves from belief set
    datHLA.remove( ev.username, HLA );
    } //if
} //if
if (!HLA.connected)
{
    // If the constructor succeeded, ev.username has been registered, even if the connection later failed
    if (HLA.remove( ev.username, HLA ))
    { // Send a MessageEvent back to the client to indicate that the login procedure was a failure.
        System.err.println("Couldn't log user in. HLA RTI connection failed."");
        @send( ev.from, evLoginRes.result( ev.username, ev.from, false ) );
        System.err.println("Couldn't log user in. Username already exists.");
        if (!HLA.me_logged_in)
        {
            @send( ev.from, evLoginRes.result( ev.username, ev.from, false ) );
            System.err.println("Couldn't log user in. Username already exists.");
        }
        else if (!HLA.me_shutting_down)
        {
            HLA.doUnsubscribe();
            @subtask( evHLA_ForceDivest.create( ev.username ) );
            HLA.doUnpublish();
            HLA.resignFederationExecution();
            // Garbage collector should deal with HLA now
            System.gc();
        }
        else
        {
            // Add new user to the Users beliefset
            // CAREFUL: Sorting the fields of a beliefset changes the sequence of parameters in the add/remove methods!
            beliefset.add( ev.from, ev.username );
            System.out.println("Added "+ ev.username + ": " to Users belief set.");
            // Send a MessageEvent back to the client to indicate that the login procedure was a success.
            @send( ev.from, evLoginRes.result( ev.username, ev.from, true ) );
            System.out.println("User successfully logged in.");
        }
    } else {
        // Must disconnect now
        HLA.me_shutting_down = true;
        HLA.doUnsubscribe();
        @subtask( evHLA_ForceDivest.create( ev.username ) );
        HLA.doUnpublish();
        HLA.resignFederationExecution();
        // Garbage collector should deal with HLA now
        System.gc();
    }
} else {
    // Add new user to the Users beliefset
    // CAREFUL: Sorting the fields of a beliefset changes the sequence of parameters in the add/remove methods!
    beliefset.add( ev.from, ev.username );
    System.out.println("Added "+ ev.username + ": " to Users belief set.");
    // Send a MessageEvent back to the client to indicate that the login procedure was a success.
    @send( ev.from, evLoginRes.result( ev.username, ev.from, true ) );
    System.out.println("User successfully logged in.");
} if
} //end plnLoginUser
The `plnLogoutUsr` plan handles the process of logging a user out.

```java
// File: plnLogoutUsr.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import client.evtLogout;

/**
 * This plan handles the evtLogout event which occurs when the Client logs out.
 * After the considerable HLA clean up, it removes the username from the Users (and datHLA) belief set(s).
 */
public plan
plnLogoutUsr
extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtLogout ev;
    #posts event evtHLA_AcquireChatRoomRegistry
    evHLA_AcquireChatRoomRegistry;
    #posts event evtHLA_ForceDivest evHLA_ForceDivest;
    #uses interface Server server;
    #uses data blfHLA datHLA;
    #uses data blfUsers blfdatUsers;

    static boolean relevant (evtLogout ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.user, _HLA ) &&
        { (HLA = (Server.HLAchat)._HLA.as_object()) != null};
    }

    #reasoning method
    body()
    {
        //Log out procedure
        System.out.println("HLAchat shutting down...");
        HLA._me_shutting_down = true;
        HLA._me_logged_in = false;
        //Move from "current" ChatRoom to "waiting_room" ChatRoom
        //Unsubscribe from the current ChatRoom
        HLA._rtiAmbassador.unsubscribeInteractionClassWithRegions(HLA._ich_Communication, HLA._rhs_current_ChatRoom);
        //Unsubscribe from the user handle channel
        HLA._rtiAmbassador.unsubscribeInteractionClassWithRegions(HLA._ich_Communication, HLA._rhs_myParticipantRegion);
    }
```
//Look at the ChatRoom we're leaving to decide whether to delete it or not
int count = 0;
@wait_for( HLA._sem_theParticipants.planWait() );
[
    Server.HLAchat.aParticipant _iParticipant;
    for ( java.util.Iterator i =
        HLA._theParticipants.values().iterator(); i.hasNext(); )
        
        _iParticipant = (Server.HLAchat.aParticipant)i.next();
        //Out of scope Participants have unreliable attributes;
        they cannot be in our ChatRoom in any case
        //The only exceptions are owned Participants: any owned orphans can only be in the waiting_room
        //me isn't counted
        if( { !(HLA.me.handle.equals(_iParticipant.handle)) }&&
            { _iParticipant.inscope }&&
            (HLA.me.chat_room_slot.equals(
                _iParticipant.chat_room_slot ) ) ) count++;
    } //synchronized(_theParticipants)
    HLA._sem_theParticipants.signal();

    //This'll cause the Participants in our ChatRoom to go out of
    HLA._rtiAmbassador.unsubscribeObjectClassAttributesWithRegions(  
        HLA._ochParticipant, HLA._asrsplParticipant_current );

    //What if someone logs into the ChatRoom after we’ve counted its
    Participants but before we actually delete it?
    //Doesn't seem worth synchronizing on _myChatRoom
    if (count <= 0) /*Should be just ==0 but you never know (because
        of inscope, _me isn't counted)
        */
        
        //We were the only Participant left in the ChatRoom; delete it as we leave (we must be the owner, obviously)
        @subtask( evHLA_AcquireChatRoomRegistry.create( ev.user ) );
    //HLA.username
    
        //Because we're the owner, we won't get a RemoveObjectInstance
        notification
        HLA._rtiAmbassador.deleteObjectInstance(  
            HLA._myChatRoom.handle, null );
        //Because we're the owner, we won't get a
        RemoveObjectInstance notification
        //The remove returns the removed value, which is
        _myChatRoom
        HLA._rtiAmbassador.deleteObjectInstance(  
            (Server.HLAchat.aChatRoom)HLA._theChatRooms.remove(  
                HLA._myChatRoom.handle ) ).handle, null );
        HLA.RemoveChatRoomAndUpdateRegistry(
            HLA._myChatRoom.slot.getValue() );
    } //synchronized(AcquireChatRoomRegistry());
    HLA._sem_ChatRoomRegistry.signal();
}
/**
 * This is now done in doForceDivest:
 * else if (HLA._myChatRoom.owned)
 * {
 * //There are other Participants but we were the owner:
 * transfer ownership
 * HLA._myChatRoom.divesting = true;
 * HLA._rtiAmbassador.negotiatedAttributeOwnershipDivestiture( 
 * HLA._myChatRoom.handle, HLA._oahs_ChatRoom, null );
 * }
 * */
 * HLA._myChatRoom = null;

 //The _me Participant is the only one that can be elsewhere than
 the waiting_room and be owned;
 //by definition, an owned Participant stands in for the federate
 in whichever ChatRoom it is
 //Move it from current to waiting_room
 HLA._rtiAmbassador.unassociateRegionsForUpdates( HLA._me.handle,
 HLA._asrspl_Participant_current );
 @wait_for( HLA._sem_me.planWait() );
 {
 HLA._me.logged_in.setBoolean(false);
 HLA._me.chat_room_slot.setValue(
 HLA._slot_waiting_room_ChatRoom );
 HLA._rtiAmbassador.associateRegionsForUpdates( 
 HLA._me.handle, HLA._asrspl_Participant_waiting_room );
 //AutoProvide will fetch our attribute update (since all
 federates subscribe to the waiting_room ChatRoom at all times)
 } //synchronized(_me)
 HLA._sem_me.signal();
 HLA._me = HLA.getNewParticipant();
 // HLA._me = HLA.new aParticipant(); //awaiting bug fix

 //Delete the now unused regions
 HLA._rhs_myParticipantRegion.remove( HLA._rh_myParticipantRegion 
 );
 HLA._rtiAmbassador.deleteRegion( HLA._rh_myParticipantRegion );
 HLA._rh_myParticipantRegion = null;
 HLA._rhs_myParticipantRegion = null;
 if (!HLA._rh_current_ChatRoom.equals(HLA._rh_general_ChatRoom))
 {
 HLA._rtiAmbassador.deleteRegion( HLA._rh_current_ChatRoom );
 HLA._rh_current_ChatRoom = null;
 }

 //Here begins the exitForm equivalent
 HLA.doUnsubscribe();
 @subtask( evHLA_ForceDivest.create( ev.user ) ); //or
 HLA.username
 HLA.doUnpublish();
 HLA.resignFederationExecution();
IIRemove oursevles from datHLA
System.out.println("Logging '" + ev.user + "' out.");
datHLA.remove( ev.user, HLA );
// Remove user from the Users beliefset.
logical String location;
blfdatUsers.get( location, ev.user );
blfdatUsers.remove( location.as_string(), ev.user );
System.out.println( "Logged user '" + ev.user + "' out." );
//Garbage collector should deal with HLA now
System.gc();
}
//end plnLogoutUsr

The plnMessageGrp plan handles the sending of a message to the current group.

// File: plnMessageGrp.java
package server;

import client.evtMessageGrp;
import ca.gc.drdc.rddc.hla.rti1516.ont.*;
import ca.gc.drdc.rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/**
 * This plan handles the evtMessageGrp, which occurs when a Client sends "msg" to its current group.
 * Like plnRequestGrpWho, the original design posted an Inference Goal Event to unify over the various usernames associated to the current group.
 * Here we simply send the HLA interaction.
 */
public plan plnMessageGrp
extends Plan
{
    logical Object HLA;
    Server.HLAchat HLA;
    #handles event evtMessageGrp ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtMessageGrp ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.fromUsr, _HLA ) &&
        ( _HLA = (Server.HLAchat)_HLA.as_object() ) != null);
    }
# reasoning method

body()
{
    @post( msgGrpIGE.message( ev.group, ev.msg, ev.from, ev.fromUsr ) )
    {
        System.out.println( " (sending message to group " + ev.group + ")" )
    }
}

The `plnMessageUsr` plan handles the sending of a message to a specific user.

```java
// File: plnMessageUsr.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.omt.*;
import ca.gc.drdc_rddc.hla.rti1516.blfHLA;
import hla.rti1516.*;
import client.evtMessageUsr;

public plan
plnMessageUsr
    extends Plan
{
    // handles event evtMessageUsr ev;
    // sends event evtMessageUsrRes evMessageUsrRes;
    // uses interface Server server;
    // reads data blfHLA datHLA;
```
static boolean relevant(evtMessageUsr ev) 
{ 
    return true; 
} 

context() 
{ 
    datHLA.get( ev.fromUsr, _HLA ) && 
    ( (HLA = (Server.HLAchat)_HLA.as_object()) != null); 
}

#reasoning method
body() 
{ 
    //The message could be the zero-length String
    HLAunicodeString us_cmdline = new HLAunicodeString(ev.msg);
    //Note that there is no such thing as a ParameterHandleSet
    ParameterHandleValueMap phvm_parameters = 
    HLA._rtiAmbassador.getParameterHandleValueMapFactory().create(2);
    phvm_parameters.put(HLA._iph_Communication_message, us_cmdline.toByteArray());
    phvm_parameters.put(HLA._iph_Communication_sender, HLA._me.name.toByteArray());

    //We'll use DDM to send the message to a single user (within the
    //same chat room)
    //The Participant object has three attributes, all three of
    //which are bound
    //to the ChatRoomSlots dimension; in addition, user_handle is
    //bound to UserHandleSlots.

    //To allow user names and chat room names to be anything, the
    latter will be prefixed
    //by "p" and "c", respectively, whilst our reserved names will
    be prefixed by "."
    HLAunicodeString _us_TargetName = new HLAunicodeString("p" + ev.toUsr);
    @wait_for{ HLA._sem_theParticipants.planWait() };
    // synchronized(_theParticipants) 
    //Target may or may not be known or even extant, since the
    Client can specify any ID
    //ObjectInstanceHandle key = 
    HLA._rtiAmbassador.getObjectInstanceHandle("p" + ev.toUsr);
    Server.HLAchat.aParticipant _Participant = 
    HLA.SeekParticipantByName(_us_TargetName.toString());
    //me is usually not in scope (being owned), so you cannot
    talk to yourself
```java
if ((_Participant == null) || (!_Participant.inscope) ||
(!_Participant.logged_in.getBoolean()))
{
    @send( ev.from, evMessageUsrRes.result( false, ev.toUsr )
    System.err.println(" (target user " + ev.toUsr + ") not found");
    HLA._sem_theParticipants.signal();
    return true;
}

//The _ich_Communication interaction has two dimensions:
UserHandleSlots and ChatRoomSlots
RegionHandle _rh_aParticipant =
HLA._rtiAmbassador.createRegion( HLA._chs_UserHandleSlotsSet );
HLA._rtiAmbassador.setRangeBounds(_rh_aParticipant,
HLA._dh_UserHandleSlots, new RangeBounds(
_Participant.user_handle.getValue(),
_Participant.user_handle.getValue() + 1 ));
RegionHandleSet _rhs_aParticipant =
HLA._rtiAmbassador.getRegionHandleSetFactory().create();
(rhs_aParticipant.add(_rh_aParticipant);
HLA._rtiAmbassador.commitRegionModifications(
_rhs_aParticipant );
HLA._rtiAmbassador.sendInteractionWithRegions(
HLA._ich_Communication, _phvm_parameters, _rhs_aParticipant, null );
//Now throw the region away
_rhs_aParticipant.remove( _rh_aParticipant );
HLA._rtiAmbassador.deleteRegion( _rh_aParticipant );
//synchronized(_theParticipants)
HLA._sem_theParticipants.signal();
//Send an acknowledgement message back to the sending user.
@end( ev.from, evMessageUsrRes.result(true, ev.toUsr )
System.out.println(" (sending message to " + ev.toUsr + ")")
};
}
@end printlnMessageUsr
```
The `plnRequestGrpWho` plan handles a Client request for a list of other users within the same group.

// File: plnRequestGrpWho.java
package server;

import client.evtRequestGrpWho;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * This plan handles the `evtRequestGrpWho`, which occurs when a Client asks "who" belongs to a given Group (could be its current one).
 * It checks whether the group exists (is known); if it does, in the original design it posts an `evtGrpWhoIGE`; if it does not, it sends back an `evtNoGrp`.
 * Since `evtGrpWhoIGE` is an Inference Goal Event, it runs all applicable instances of Plans; the `plnGrpWho` then used its context() to unify over all usernames found associated with the requested group.
 * This rather clever use of an IGE is not applicable here, so we do away with the event and plan altogether (We can do this because they are internal to server). We only keep the original's outcome, the series of `evtGrpWhoRes` events it sends back to the Client.
 */
public plan
plnRequestGrpWho
extends Plan
{
    logical Object _HLA;
    Server.HLachat HLA;
    #handles event `evtRequestGrpWho` ev;
    #sends event `evtGrpWhoRes` evGrpWhoRes;
    #sends event `evtNoGrp` evNoGrp;
    #uses interface `Server` server;
    #reads data `blfHLA` datHLA;

    static boolean relevant(evtRequestGrpWho ev)
    |
    return true;
}
context()
{
   // Where the other events sent by the Client specify the
   // username
   // (in the ev.user field, filled from ((Client)
   // self).getUserName()),
   // this evtRequestGrpWho does not. Luckily, we can recover it
   // from the ev.from, which will be of the form
   "username@portalname".
   // datHLA.get( ev.owner, _HLA ) &&
   // server.println( Thread.currentThread() + "
   // plnRequestGrpWho.context: username is " + ev.from.substring(0,
   // ev.from.indexOf("@")) + "" ) &&
   // datHLA.get( ev.from.substring(0, ev.from.indexOf("@")), _HLA )
   &&
   ( _HLA = (Server.HLACHAT).HLA.as_object() != null );
}

// reasoning method
body()
{
   Server.HLACHAT.aChatRoom _ChatRoom;
   @wait_for( HLA._sem_theChatRooms.planWait() );
   {
      // The Client will ask for "General" when it means
      "<General>"
      // (it is allowed to "Join General", which creates
      "cGeneral");
      // the local name will hide the global.
      _ChatRoom = HLA.SeekChatRoomByName("c" + ev.group);
   }
   HLA._sem_theChatRooms.signal();
   if ( [ _ChatRoom == null ] && (ev.group.equals("General")) )
      _ChatRoom = HLA._general_chatRoom;
if (_ChatRoom != null)
{
    // group does exist
    // Original design was:
    // @post{ evGrpWhoIGE.who( ev.group, ev.from ) };
    // System.out.println( " (retrieving members of group " +
    // ev.group + ")" );
    // @wait_for( HLA._sem_theParticipants.planWait() );
    
    for (java.util.Iterator i =
        HLA._theParticipants.values().iterator(); i.hasNext(); )
    {
        Server.HLAchat.aParticipant _Participant =
            (Server.HLAchat.aParticipant)i.next();
        // _me is filtered out because it is not in scope (being
        // owned)
        if (_Participant.chatter_slot.equals(_ChatRoom.slot))
        {
            @send( ev.from, evGrpWhoRes.result(
                _Participant.name.toString().substring(1) ) );
            System.out.println( " (reporting group member " +
                _Participant.name.toString() + ")) " );
        } //if
    } //for
    // Add me
    if (HLA._me.chatter_slot.equals(_ChatRoom.slot))
    {
        // HLA.username is:
        // ev.from.substring(0, ev.from.indexOf("@"))
        // or equivalently
        // HLA._me.name.toString().substring(1)
        @send( ev.from, evGrpWhoRes.result(
            HLA._me.name.toString().substring(1) ) );
        System.out.println( " (reporting group member " +
            HLA._me.name + ") ");
    }
} //synchronized(_theParticipants)
HLA._sem_theParticipants.signal();
}
else {
    // group doesn't exist
    @send( ev.from, evNoGrp.group{ ev.group } );
    System.err.println( " (group " + ev.group + ") not
    extant) ");
} //if

// end plnRequestGrpWho
The plnHLA_AcquireChatRoomRegistry plan handles the (frequent) sub-task of securing ownership of the ChatRoomRegistry.

// File: plnHLA_AcquireChatRoomRegistry.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * This subtask takes care of acquiring the ChatRoomRegistry for a specific HLAchat instance. It is meant to be called as a @subtask()
 */

public plan plnHLA_AcquireChatRoomRegistry extends Plan {

    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLA_AcquireChatRoomRegistry ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLA_AcquireChatRoomRegistry ev) {
        return true;
    }

    context() {
        datHLA.get( ev.identifier, _HLA ) &
            ( (HLA == (Server.HLAchat)_HLA.as_object()) || null );
    }

    #reasoning method
    body() {
        @wait_for( HLA._sem_ChatRoomRegistry.planWait() );
        //Presumably the Semaphore will remain grabbed by this thread
        //and will therefore pass from this plan to the invoking one
        if (HLA._sem_ChatRoomRegistry.owned) return true;
        HLA.sem_ChatRoomRegistry_acquisition.clearChanged();
        HLA._rtiAmbassador.attributeOwnershipAcquisition(HLA._ChatRoomRegistry.handle, HLA._oahs_ChatRoomRegistry, null);
        @wait_for( new aos.jack.util.cursor.Change( HLA.sem_ChatRoomRegistry_acquisition,
                                                    HLA.sem_ChatRoomRegistry_acquisition.hasChanged() ) );
        //Failure to acquire is not an option
    }
}
//end plnHLA_AcquireChatRoomRegistry
The `plnHLA_ForceDivest` plan handles the sub-task of aggressively divesting ownership of any still-owned objects, in order to complete a Client’s withdrawal from the Chat federation.

```java
// File: plnHLA_ForceDivest.java
package server;

import ca.gc.drdc_rddc.hla.rtl1516.jackson.blfHLA;

/**
 * This subtask takes care of divesting any owned objects during shutdown. It is meant to be called as a @subtask()
 */
public plan plnHLA_ForceDivest
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLA_ForceDivest ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLA_ForceDivest ev)
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
            ( (HLA = (Server.HLAchat)_HLA.as_object()) != null);
    }

    #reasoning method
divestChatRoomRegistry()
    {
        //This method is invoked when the federate wants to shut down but still owns some instance attributes
        if (!HLA._ChatRoomRegistry.owned) return true;
        HLA._semChatRoomRegistry_divestiture.clearChanged();
        HLA._ChatRoomRegistry.divesting = true;
        HLA._rtiAmbassador.negotiatedAttributeOwnershipDivestiture(
            HLA._ChatRoomRegistry.handle, HLA._oahs_ChartRoomRegistry, null );
        @wait_for( new aos.jack.util.cursor.Change( HLA._semChatRoomRegistry_divestiture,
            HLA._semChatRoomRegistry_divestiture.hasChanged() ) )
        //Other federates will receive requestAttributeOwnershipAssumption and
        //respond with attributeOwnershipAcquisitionIfAvailable.
        //We'll get requestDivestitureConfirmation, to which we'll confirmDivestiture
        //Once that is done, we no longer own _ChatRoomRegistry and can resign normally.
    } //divestChatRoomRegistry
```
```java
public void divestChatRooms()
{
    // This method is invoked when the federate wants to shut down
    // but still owns some instance attributes
    // Any federate callbacks requesting ownership assumption will be
turned down right away;
    // however, ownership acquisition notifications could conceivably
squeak through
    // (if another federate started shutting down just before this
one does) and would lock
    // up on the synchronized(_theChatRooms), so we take a snapshot
and release the monitor right away.
    Server.HLAchat.aChatRoom[] _ChatRooms;
    @wait_for( HLA._sem_theChatRooms.planWait() );
    _ChatRooms = (Server.HLAchat.aChatRoom[])
    HLA._theChatRooms.values().toArray( new Server.HLAchat.aChatRoom[0] );
    // synchronized(_theChatRooms)
    HLA._sem_theChatRooms.signal();
    for (Iterator i = HLA._theChatRooms.entrySet().iterator();
i.hasNext();)
    for (int i = 0; i < _ChatRooms.length; i++)
    {
        _ChatRoom = (Server.HLAchat.aChatRoom)
        (java.util.Map.Entry)i.next().getValue();
        if (_ChatRoom.owned)
        {
            HLA._sem_ChartRoom_divestiture.clearChanged();
            _ChatRoom.divesting = true;
            //...
        }
    }
    HLA._rtiAmbassador.negotiatedAttributeOwnershipDivestiture;
    HLA._oahs_ChartRoom = _ChatRoom.handle;
    @wait_for( new aos.jack.util.cursor.Change(;
    HLA._sem_ChartRoom_divestiture,
    HLA._sem_ChartRoom_divestiture.hasChanged() ) );
    } //if
} //divestChatRooms
```
divestParticipants()
{
    // This method is invoked when the federate wants to shut down
    // but still owns some instance attributes
    // Any federate callbacks requesting ownership assumption will be
    // turned down right away;
    // however, ownership acquisition notifications could conceivably
    // squeak through
    // (if another federate started shutting down just before this
    // one does) and would lock
    // up on the synchronized(_theParticipants), so we take a
    // snapshot and release the monitor right away.
    synchronized(_theParticipants)
    { _Participants = (Server.HLAChat.aParticipant[])
      HLA._theParticipants.values().toArray( new
      Server.HLAChat.aParticipant[0] );
      _Participants = (Server.HLAChat.aParticipant[])
      HLA._theParticipants.values().toArray( new
      Server.HLAChat.aParticipant[0] );
      } // synchronized(_theParticipants)
      
      for (Iterator i = HLA._theParticipants.entrySet().iterator();
            i.hasNext())
      for (int i = 0; i < _Participants.length; i++)
      {
          _Participant = (Server.HLAChat.aParticipant)(
          java.util.Map.Entry)i.next().getValue();
          HLA._theParticipants.entrySet().iterator();
          } // synchronized(_theParticipants)
          
          HLA._rtiAmbassador.negotiatedAttributeOwnershipDivestiture(
          _Participant.handle, HLA._oahs_Participant, null );
          @wait_for( new aos.jack.util.cursor.Change(
          HLA._Participant_divestiture, HLA._Participant_divestiture.hasChanged() ) )
          if (_Participant.owned)
          {
              HLA._Participant_divestiture.clearChanged();
              _Participant.divesting = true;
          }
          } //for
          } //divestParticipants

    // No action required if last federate out
    if (HLA._alone) return true;
    // This method is invoked when the federate wants to shut down
    // but still owns some instance attributes
    // ChatRoomRegistry object
    divestChatRoomRegistry();
    // ChatRoom objects
    divestChatRooms();
    // Participant objects
    divestParticipants();
}
The plnHLA_ChatRoomRegistry_Discovery plan handles the discovery of the ChatRoomRegistry object.

```java
// File: plnHLA_ChatRoomRegistry_Discovery.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAdiscOeveObjectInstance;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * Handles HLA ChatRoomRegistry Discovery
 */
public plan plnHLA_ChatRoomRegistry_Discovery
    extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    #handles event evtHLAdiscOeveObjectInstance ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAdiscOeveObjectInstance ev)
    {
        return ev.objectName.equals(
            Server.HLAChat._name_ChatRoomRegistry );
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        ( (HLA = (Server.HLAChat)_HLA.as_object()) != null ) &&
        ev.theObjectClass.equals( HLA._ocl_ChatRoomRegistry );
    }

    #reasoning method
    body()
    {
        // Server.HLAChat HLA = (Server.HLAChat)_HLA.as_object();
        HLA._ChatRoomRegistry.handle = ev.theObject;
        //Signal the waiting plnHLA_ChatRoomRegistryNameReservation that
        we’re done
        HLA.sem_ChatRoomRegistry_discovery.setChanged();
    }
}
//end plnHLA_ChatRoomRegistry_Discovery
```
The `plnHLA_ChatRoomRegistry_NameReservation_Failed` plan handles the failed outcome of the ChatRoomRegistry name reservation.

```java
package server;

import ca.gc.drdc_rddc.hla.rti1516.ack.evtHLAobjectInstanceNameReservationFailed;
import ca.gc.drdc_rddc.hla.rti1516.ack.blfHLA;

/**
 * Handles the evtHLAobjectInstanceNameReservationFailed for ChatRoomRegistry
 */
public plan plnHLA_ChatRoomRegistry_NameReservationFailed
extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAobjectInstanceNameReservationFailed ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAobjectInstanceNameReservationFailed ev)
    {
        return ev.objectName.equals(Server.HLAchat._name_ChatRoomRegistry);
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &
        ( (HLA = (Server.HLAchat)_HLA.as_object()) != null);
    }

    #reasoning method
    body()
    {
        //Name reservation failed, which means there is an already
        extant instance
        HLA._sem_ChatRoomRegistry_discovery.clearChanged();
        HLA._ChatRoomRegistry.subscribed = true;
        HLA._rtiAmbassador.subscribeObjectClassAttributes(
            HLA._och_ChatRoomRegistry, HLA._oahs_ChatRoomRegistry_forUpdate );
        @wait_for( new aos.jack.util.cursor.Change(
            HLA._sem_ChatRoomRegistry_discovery,
            HLA._sem_ChatRoomRegistry_discovery.hasChanged() ) );
        //Signal the waiting plnLoginUser that we're done
        HLA._sem_name_reservation.setChanged();
    }

    //end plnHLA_ChatRoomRegistry_NameReservation_Failed
```
The `plnHLA_ChatRoomRegistry_NameReservation_Succeeded` plan handles the successful outcome of the ChatRoomRegistry name reservation.

```java
// File: plnHLA_ChatRoomRegistry_NameReservation_Succeeded.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.ack.evtHLAobjectInstanceNameReservationSucceeded;
import ca.gc.drdc_rddc.hla.rti1516.ack.blfHLA;

/**
 * Handles the evtHLAobjectInstanceNameReservationSucceeded for ChatRoomRegistry
 */
public plan plnHLA_ChatRoomRegistry_NameReservation_Succeeded
    extends Plan
{
    logical Object _HLA;
    Server.HLACHat _HLA;
    // handles event evtHLAobjectInstanceNameReservationSucceeded ev;
    // uses interface Server server;
    // reads data blfHLA datHLA;
    static boolean relevant(evtHLAobjectInstanceNameReservationSucceeded ev)
    {
        return ev.objectName.equals(Server.HLACHat._name_ChatRoomRegistry);
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        ( _HLA = (Server.HLACHat)_HLA.as_object() ) != null);
    }

    // reasoning method
    body()
    {
        // Having reserved the name, we know we're the first federate
        // to reach this point, so we must create the ChatRoomRegistry.
        HLA._ChatRoomRegistry.owned = HLA._ChatRoomRegistry.subscribed = true;
        // HLA._ChatRoomRegistry.divesting = false;
        HLA._ChatRoomRegistry.handle =
            HLA._rtiAmbassador.registerObjectInstance( HLA._och_ChatRoomRegistry,
            HLA._ChatRoomRegistry.name );
        HLA._rtiAmbassador.subscribeObjectClassAttributes(
            HLA._och_ChatRoomRegistry, HLA._oahs_ChatRoomRegistry_forUpdate );
        // Signal the waiting plnLoginUser that we're done
        HLA.sem_name_reservation.setChanged();
    }
}
// end plnHLA_ChatRoomRegistry_NameReservation_Succeeded
```
**The plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed plan handles**
the failed outcome of a ChatRoomRegistry ownership acquisition attempt.

```java
// File: plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLA.attributeOwnershipUnavailable;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
* Handles the AttributeOwnershipUnavailable notification for
* ChatRoomRegistry
*/
public plan plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed
  extends Plan
{
  logical Object _HLA;
  Server.HLACHAT HLA;
  #handles event evtHLA.attributeOwnershipUnavailable ev;
  #uses interface Server server;
  #reads data blfHLA datHLA;

  static boolean relevant (evtHLA.attributeOwnershipUnavailable ev)
  |
  return true;
  |

  context()
  |
  datHLA.get( ev.identifier, _HLA ) &&
  ( (HLA = (Server.HLACHAT) HLA.as_object()) != null ) &&
  //
  (Server.HLACHAT) HLA.as_object().och_ChatRoomRegistry.equals( HLA._och_ChatRoomRegistry );

  #reasoning method
  body()
  |
  // Server.HLACHAT HLA = (Server.HLACHAT) HLA.as_object();
  HLA.sem_ChatRoomRegistry_acquisition.setChanged();
}
//end plnHLA_ChatRoomRegistry_OwnershipAcquisitionFailed
```
The `plnHLA_ChatRoomRegistry_OwnershlpAcqulsltlonNotlflcatlon` plan handles the successful outcome of a ChatRoomRegistry ownership acquisition attempt.

```java
// File: plnHLA_ChatRoomRegistry_OwnershlpAcqulsltlonNotlflcatlon.java
package server;

import ca.gc.drdc.rddc.hla.rtl1516.ack.evtHLAattributeOwnershipAcquisitionNotification;
import ca.gc.drdc.rddc.hla.rtl1516.ack.blfHLA;

/**
Handles the attribute ownership notification for ChatRoomRegistry
*/
public plan
plnHLA_ChatRoomRegistry_OwnershlpAcqulsltlonNotlflcatlon
extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAattributeOwnershipAcquisitionNotification ev;
    #posts event evtHLA_ForceDivest evHLA_ForceDivesti;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean
relevant(evtHLAattributeOwnershipAcquisitionNotification ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &
        ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &
        HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
        ).equals( HLA._och_ChatRoomRegistry );
    }
```
#reasoning method

body()
{
    Server.HLAchat HLA = (Server.HLAchat) HLA.as_object();
    @wait_for{ HLA._sem_ChatRoomRegistry.planWait(); }
    // synchronized(HLA._ChatRoomRegistry)
    {
        HLA._ChatRoomRegistry.owned = true || (HLA._ChatRoomRegistry.divesting = HLA._me_shutting_down);
        //synchronized(HLA._ChatRoomRegistry)
        HLA._sem_ChatRoomRegistry.signal();
        HLA._sem_ChatRoomRegistry_acquisition.setChanged();
        //Just in case we somehow managed to acquire whilst switching to
        //shutting down mode
        if (HLA._me_shutting_down) server.postEvent(
            evHLA_ForceDivest.create( ev.identifier ) );
    }
    //end plnHLA_ChatRoomRegistry_OwnershipAcquisitionNotification
}

The plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest plan handles
ownership assumption requests for the ChatRoomRegistry object.

// File: plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest.java
package server;

import ca.gc.drdd_rddc.hla.rti1516.jack.evtHLArequestAttributeOwnershipAssumption;
import ca.gc.drdd_rddc.hla.rti1516.jack.blfHLA;

/**
Handles OwnershipAssumptionRequests for ChatRoomRegistry
 */
public plan
plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest
extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLArequestAttributeOwnershipAssumption ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLArequestAttributeOwnershipAssumption ev)
    {
        return true;
    }
context()
{
    datHLA.get( ev.identifier, _HLA ) &&
( (HLA = (Server.HLAdat) _HLA.as_object() ) != null ) &&
HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_ChatRoomRegistry );
}

#reasoning method
body()
{
    // Server.HLAdat HLA = (Server.HLAdat) _HLA.as_object();
    ///Decline if we're in the process of shutting down (and thus
divesting)
    //Note that we return true because the plan was a success
    //even though we simply decided to ignore the request
    if (HLA.me_shutting_down) return true;
    //Acquiesce if able
    //This condition is similar to !_me_logged_in but better
    //If not subscribed, probably not up to date --decline
    if (!HLA._ChatRoomRegistry.subscribed) return true;
    //Note that the divesting federate will have offered ownership
to all federates, and that even if they all acquiesce,
    //only one will receive an ownershipAcquisitionNotification; if
    we use attributeOwnershipAcquisition, there won't be any
    //negative feedback if we fail to get ownership --and we'll have
    an outstanding acquisition request.
    //This is why it is preferable to use
    attributeOwnershipAcquisitionIfAvailable
    HLA.rtiAmbassador.attributeOwnershipAcquisitionIfAvailable(
    HLA._ChatRoomRegistry.handle, HLA._oahs_ChatRoomRegistry);
}

//end plnHLA_ChatRoomRegistry_OwnershipAssumptionRequest
The plan handles requests for confirmation of divestiture of the ChatRoomRegistry object.

// File: plnHLA_ChatRoomRegistry_OwnershlpDivestltureConflrmatlonRequest.java
package server;

import ca.gc.drdc_rddc.hla.rtl1516.jack.evtHLArequestDivestitureConfirmation;
import ca.gc.drdc_rddc.hla.rtl1516.jack.blfHLA;

/**
 * Handles requests for confirmation of ownership divestiture of ChatRoomRegistry
 */
public plan plnHLA_ChatRoomRegistry_OwnershlpDivestltureConflrmatlonRequest
  extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLArequestDivestitureConfirmation ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLArequestDivestitureConfirmation ev)
    {
      return true;
    }

    context()
    {
      datHLA.get( ev.identifier, _HLA ) &&
      ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
      // ev.theObjectClass.equals( ((Server.HLAchat)_HLA.as_object()).och_ChatRoomRegistry );
      HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
        ).equals( HLA._och_ChatRoomRegistry );
    }
The plnHLA_ChatRoomRegistry_OwnershipReleaseRequest plan handles ownership release requests for the ChatRoomRegistry object.
context()
{
    datHLA.get( ev.identifier, _HLA ) &&
    ( HLA = (Server.HLAchat)_HLA.as_object() ) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_ChatRoomRegistry );
}

#reasoning method
body()
{
    Server.HLAchat HLA = (Server.HLAchat)_HLA.as_object();
    //Acquiesce
    @wait_for( HLA._sem_ChatRoomRegistry.planWait() );
    //synchronized(HLA._ChatRoomRegistry)
    {
        //Plan succeeds even though divestiture fails
        if ( !(HLA._oahs_ChatRoomRegistry.containsAll( HLA._rtiAmbassador.attributeOwnershipDivestitureIfWanted( HLA._ChatRoomRegistry.handle, HLA._oahs_ChatRoomRegistry ))) return true;
        HLA._ChatRoomRegistry.owned = HLA._ChatRoomRegistry.divesting = false;
    } //synchronized(_ChatRoomRegistry)
    HLA._sem_ChatRoomRegistry.signal();
    HLA._sem_ChatRoomRegistry_divestiture.setChanged();
}
//end plnHLA_ChatRoomRegistry_OwnershipReleaseRequest

The plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate plan handles requests for ChatRoomRegistry attribute value updates.

// File: plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAprovideAttributeValueUpdate;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/** Handles attribute value update requests received from the RTI for the ChatRoomRegistry. */
public plan
plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAprovideAttributeValueUpdate ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;
static boolean relevant(evtHLAprovideAttributeValueUpdate ev) {
    return true;
}

context() {
    datHLA.get( ev.identifier, HLA ) &&
    ( (HLA = (Server.HLAsync).as_object()) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._oah_ChatRoomRegistry );
}

#reasoning method
body() {
    AttributeHandleValueMap _ahvm_attributeValues =
    HLA._rtiAmbassador.getAttributeHandleValueMapFactory().create(
    HLA._oah_ChatRoomRegistry_forUpdate.size() );
    //HLAprivilegeToDeleteObject is of undefined type (it has no
    //content)
    _ahvm_attributeValues.put(
    HLA._oah_ChatRoomRegistry_DeletePrivilege, null);
    _ahvm_attributeValues.put( HLA._oah_ChatRoomRegistry_list,
    HLA._ChatRoomRegistry.list.toByteArray() );
    HLA._rtiAmbassador.updateAttributeValues( 
    HLA._ChatRoomRegistry.handle, _ahvm_attributeValues, null );
}
//end plnHLA_ChatRoomRegistry_ProvideAttributeValueUpdate

The plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate plan handles
ChatRoomRegistry attribute value update reflections.

// File: plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAreflectAttributeValues;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 Handles attribute value updates received from the RTI for the
 ChatRoomRegistry.
 */
public plan
plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate
extends Plan{
    logical Object _HLA;
    Server.HLAsync HLA;
    #handles event evtHLAreflectAttributeValues ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;
static boolean relevant(evtHLAreflectAttributeValues ev) {
    return true;
}

custom() {
    datHLA.get( ev.identifier, _HLA ) &&
        ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
        HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
        ).equals( HLA._och_ChatRoomRegistry );
}

#reasoning method
body() {
    //HLA.privilegeToDeleteObject has no content, so we won’t decode it
    HLA._ChatRoomRegistry.list.decode( (byte[])ev.theAttributes.get( HLA._och_ChatRoomRegistry_list ) );
}

//end plnHLA_ChatRoomRegistry_ReflectAttributeValueUpdate

The plnHLA_ChatRoom_Discovery plan handles the discovery of ChatRoom objects (including the Waiting Room).

// File: plnHLA_ChatRoom_Discovery.java
package server;

import ca.gc.drdc_rddc.hla.rtl1516.jack.evtHLAdiscoverObjectInstance;
import ca.gc.drdc_rddc.hla.rtl1516.jack.blfHLA;

/**
 Handles discovery of ChatRooms, including the "_waiting_room".
 */
public plan plnHLA_ChatRoom_Discovery extends Plan {
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAdiscoverObjectInstance ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAdiscoverObjectInstance ev) {
        return true;
    }
context()
{
    datHLA.get( ev.identifier, _HLA ) &
    (!HLA = (Server.HLAchat).HLA.as_object() != null) &&
    ev.theObjectClass.equals('HLA._och.ChatRoom');
}

#if reasoning method
#body()
{
    Server.HLAchat HLA = (Server.HLAchat).HLA.as_object();
    // Assuming AutoProvide, this'll be immediately followed by an
    Update
    Server.HLAchat.aChatRoom _ChatRoom;
    if (ev.objectName.equals('Server.HLAchat._name_waiting_room.ChatRoom'))
    {
        _ChatRoom = HLA._waiting_room.ChatRoom;
    } else {
        _ChatRoom = HLA.getNewChatRoom(); // Awaiting bug fix
        _ChatRoom = HLA.new aChatRoom();
    }
    _ChatRoom.handle = ev.theObject;
    _ChatRoom.subscribed = true;
    // _ChatRoom.name and slot will come from Update; we must wait
    until then before updating the GUI
    @wait_for( HLA._sem_theChatRooms.planWait() );
    HLA._theChatRooms.put( ev.theObject, _ChatRoom );
    HLA._sem_theChatRooms.signal();
    // It is premature to add the ChatRoom to the blfdatGrpList,
    since
    // all we know at this point is the HLA name (ev.objectName),
    *not* // the ChatRoom's "name" attribute.
    // Special case: waiting room
    // The discovery of this never-deleted ChatRoom is always waited
    upon
    // We do not want to trip the discovery semaphore (waiting for a
    Participant) with a ChatRoom discovery
    if (!ev.objectName.equals('Server.HLAchat._name_waiting_room.ChatRoom')) return true;
    // Signal the waiting plnHLA.WaitingRoomNameReservation that
    we're done
    HLA._sem_ChatRoom_discovery.setChanged();
}
//#end plnHLA_ChatRoom_Discovery
The `plnHLA_ChatRoom_OwnershipAcquisitionFailed` plan handles the failed outcome of ChatRoom attribute ownership requests.

```java
// File: plnHLA_ChatRoom_OwnershipAcquisitionFailed.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAattributeOwnershipUnavailable;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/** Handles the AttributeOwnershipUnavailable notification for ChatRooms */
public plan plnHLA_ChatRoom_OwnershipAcquisitionFailed
    extends Plan {
    logical Object _HLA;
    Server.HLAChat HLA;
    // handles event evtHLAattributeOwnershipUnavailable ev;
    // uses interface Server server;
    // reads data blfHLA datHLA;
    
    static boolean relevant(evtHLAattributeOwnershipUnavailable ev) {
        return true;
    }

    context() {
        datHLA.get( ev.identifier, _HLA ) &&
          ( (HLA = (Server.HLAChat)_HLA.as_object()) != null ) &&
          // ev.theObjectClass.equals( (Server.HLAChat)_HLA.as_object() ).och_ChatRoomRegistry );
          HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA.och_ChatRoom );
    }

    #reasoning method
    body() {
        HLA.sem_ChatRoom_acquisition.setChanged();
    }
}
//end plnHLA_ChatRoom_OwnershipAcquisitionFailed
```
The `plnHLA_ChatRoom_OwnershipAcquisitionNotification` plan handles the successful outcome of ChatRoom attribute ownership requests.

```java
// File: plnHLA_ChatRoom_OwnershipAcquisitionNotification.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack_evtHLA_attributeOwnershipAcquisitionNotification;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * Handles the attribute ownership notification for ChatRooms
 */
public plan plnHLA_ChatRoom_OwnershipAcquisitionNotification
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    // handles event evtHLA_attributeOwnershipAcquisitionNotification ev;
    // posts event evtHLA_ForceDivest evHLA_ForceDivest;
    // uses interface Server server;
    // reads data blfHLA datHLA;

    static boolean relevant(evtHLA_attributeOwnershipAcquisitionNotification ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
            ( _HLA - (Server.HLAchat)_HLA.as_object() ) != null ) &&
            HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_ChatRoom );
    }
```
The plnHLA_ChatRoom_OwnershipAssumptionRequest plan handles ownership assumption requests for ChatRoom objects.

// File: plnHLA_ChatRoom_OwnershipAssumptionRequest.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.ack.evtHLArequestAttributeOwnershipAssumption;
import ca.gc.drdc_rddc.hla.rti1516.ack.blfHLA;

/**
Handles OwnershipAssumptionRequests for ChatRooms
*/
public plan
plnHLA_ChatRoom_OwnershipAssumptionRequest
extends Plan
{
    logical Object _HLA;
    Server.HLChat HLA;
    //handles event evtHLArequestAttributeOwnershipAssumption ev;
    //uses interface Server server;
    //reads data blfHLA datHLA;

    static boolean relevant(evtHLArequestAttributeOwnershipAssumption ev)
    {
        return true;
    }
}
context()
{
  datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAchat) HLA.as_object()) != null) &&
  // ev.theObjectClass.equals(
    ((Server.HLAchat) HLA.as_object())._och_ChatRoom);
  HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
  ).equals( HLA._och_ChatRoom );
}

// reasoning method
body()
{
  Server.HLAchat HLA = (Server.HLAchat) HLA.as_object();
  // Note that we return true because the plan was a success
  // even though we simply decided to ignore the request
  if (HLA._me.shutting_down) return true;
  // Acquiesce if able
  @wait_for( HLA._sem_theChatRooms.planWait() );
  Server.HLAchat.aChatRoom _ChatRoom = HLA.SeekChatRoom(ev.theObject);
  HLA._sem_theChatRooms.signal();
  // if (_ChatRoom == null) throw new ObjectInstanceNotKnown("Unknown
  ChatRoom offered");
  // Also decline if:
  // We're not subscribed, thus the values are probably not up to date
  if (![_ChatRoom.subscribed]) return true;
  // The HLA federates also check to see if they're logged out and
  // the ChatRoom slot is other than the waiting_room, because they
  // can remain joined whilst logged out --not so the JACK
  // federates, which resign and shut down when they log out.
  // Since ownership is offered only when the owner leaves a
  // ChatRoom and knows there are other federates in that ChatRoom,
  // accept only if a) it's the waiting room (which is persistent)
  // or b) we're in the room being offered
  // Could use _me.chat_room_slot instead of _myChatRoom.slot
  if (([_ChatRoom.slot.equals( HLA.waiting_room_ChatRoom )) &&
      ([_ChatRoom.slot.equals( HLA._myChatRoom.slot ))
  return true;

  // Note that the divesting federate will have offered ownership
  // to all federates, and that even if they all acquiesce, only
  // one will receive an ownershipAcquisitionNotification; if we
  // use attributeOwnershipAcquisition, there won't be any negative
  // feedback if we fail to get ownership --and we'll have an
  // outstanding acquisition request.
  // This is why it is preferable to use
  attributeOwnershipAcquisitionIfAvailable
  HLA._rtiAmbassador.attributeOwnershipAcquisitionIfAvailable(
    _ChatRoom.handle, HLA._oahs_ChatRoom); // or theObject
}
// end pInHLA_ChatRoom_OwnershipAssumptionRequest
The `plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest` plan handles requests for confirmation of divestiture of ChatRoom objects.

```java
// File: plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest.java
package server;

import ca.gc.drdc.rddc.hla.rti1516.jack.evtHLArequestDivestitureConfirmation;
import ca.gc.drdc.rddc.hla.rti1516.jack.blfHLA;

/**
Handles requests for confirmation of ownership divestiture of ChatRooms
*/
public plan
plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest
extends Plan
{
  logical Object _HLA;
  Server.HLAchat HLA;
  #handles event evtHLArequestDivestitureConfirmation ev;
  #uses interface Server server;
  #reads data blfHLA datHLA;

  static boolean relevant(evtHLArequestDivestitureConfirmation ev)
  |
  return true;

  context()
  |
  datHLA.get( ev.identifier, _HLA ) &&
  ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
  //
  ev.theObjectClass.equals(
  ((Server.HLAchat)_HLA.as_object())._och.ChatRoom ),
  HLA._rtiRammbassador.getKnownObjectClassHandle( ev.theObject
  ).equals( HLA._och.ChatRoom );
}
```
#reasoning method

body()
{
    Server.HLAchat HLA = (Server.HLAchat)_HLA.as_object();
    // synchronized(HLA._theChatRooms)
    @wait_for( HLA._sem_theChatRooms.planWait() );
    Server.HLAchat aChatRoom _ChatRoom = HLA.SeekChatRoom(ev.theObject);
    // } //end of synchronized(HLA._theChatRooms)
    HLA._sem_theChatRooms.signal();
    HLA._rtiAmbassador.confirmDivestiture(_ChatRoom.handle,
    HLA._cags_ChatRoom, null );
    _ChatRoom.owned = _ChatRoom.divesting = false;
    //There'll be a divestiture semaphore of some sort when logging
    off
    HLA._sem_ChatRoom_divestiture.setChanged();
}

//end plnHLA_ChatRoom_OwnershipDivestitureConfirmationRequest

The plnHLA_ChatRoom_OwnershipReleaseRequest plan handles ownership
release requests for ChatRoom objects.

// File: plnHLA_ChatRoom_OwnershipReleaseRequest.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import ca.gc.drdc_rddc.hla.rti1516.jack(evtHLArequestAttributeOwnershipRelease;
/**
Handles requests for ownership release of ChatRooms
*/
public plan
plnHLA_ChatRoom_OwnershipReleaseRequest
extends Plan
{
    logical object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLArequestAttributeOwnershipRelease ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLArequestAttributeOwnershipRelease ev)
    {
        return true;
    }
context()
{
    datHLA.get( ev.identifier, _HLA )
    &&
        ( HLA = (Server.HLAchat) HLA.as_object() )
    &&
        HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_ChatRoom );
}

#reasoning method
body()
{
    //Acquiesce
    @wait_for( HLA._sem_theChatRooms.planWait() );
    //
    synchronized(HLA._theChatRooms) {
        Server.HLAchat.aChatRoom _ChatRoom =
            HLA.SeekChatRoom(ev.theObject);
        //Plan succeeds even though divestiture fails
        if ( !HLA._oahs_ChatRoom.containsAll( _ChatRoom.handle, HLA._oahs_ChatRoom ))
            return true;
        _ChatRoom.owned = _ChatRoom.divesting = false;
        //synchronized(_theChatRooms)
        HLA._sem_theChatRooms.signal();
        HLA._sem_ChatRoom_divestiture.setChanged();
    }
}
//end plnHLA_ChatRoom_OwnershipReleaseRequest

The plnHLA_ChatRoom_ProvideAttributeValueUpdate plan handles requests for ChatRoom instance attribute value updates.

// File: plnHLA_ChatRoom_ProvideAttributeValueUpdate.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAprovideAttributeValueUpdate;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/**
 * Handles attribute value update requests received from the RTI for the ChatRooms.
 */
public plan
plnHLA_ChatRoom_ProvideAttributeValueUpdate
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAprovideAttributeValueUpdate ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;
static boolean relevant(evtHLAprovideAttributeValueUpdate ev)
{
    return true;
}

context()
{
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAcchat_HLA.as_object()) != null) &&
        _HLA.equals(_HLA._och_ChatRoom ));
}

#reasoning method

body()
{
    @wait_for( HLA._och_theChatRooms.planWait() );
    Server.HLAcchat._ChatRoom _ChatRoom =
        HLA.SeekChatRoom(ev.theObject);
    HLA._och_theChatRooms.signal();
    AttributeHandleValueMap _ahvm_attributeValues =
        HLA._rtiAmbassador.getAttributeHandleValueMapFactory().create(
            HLA._ochs_ChatRoom_forUpdate.size() );
    _ahvm_attributeValues.put( HLA._och_ChatRoom_name,
        _ChatRoom.name.toByteArray() );
    _ahvm_attributeValues.put( HLA._och_ChatRoom_slot,
        _ChatRoom.slot.toByteArray() );
    HLA._rtiAmbassador.updateAttributeValues( _ChatRoom.handle,
        _ahvm_attributeValues, null );
}

//end plnHLA_ChatRoom_ProvideAttributeValueUpdate

The plnHLA_ChatRoom_ReflectAttributeValueUpdate plan handles
ChatRoom instance attribute value update reflections.

// File: plnHLA_ChatRoom_ReflectAttributeValueUpdate.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAreflectAttributeValues;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
Handles attribute value updates received from the RTI for the
ChatRooms.
*/
public plan
plnHLA_ChatRoom_ReflectAttributeValueUpdate
    extends Plan
{
    logical Object _HLA;
    Server.HLAcchat _HLA;
    #handles event evtHLAreflectAttributeValues ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

static boolean relevant(evtHLAreflectAttributeValues ev) {
    return true;
}

context() {
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAchat).HLA.as_object()) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_ChatRoom );
}

#reasoning method
body() {
    @wait_for{ HLA._sem_theChatRooms.planWait() };
    Server.HLAchat.aChatRoom _ChatRoom = HLA.SeekChatRoom( ev.theObject );
    HLA._sem_theChatRooms.signal();
    { //Name is a true attribute, not the object's HLA name
        _ChatRoom.name.decode( (byte[]) ev.theAttributes.get( HLA._oah_ChatRoom_name ) );
        _ChatRoom.slot.decode( (byte[]) ev.theAttributes.get( HLA._oah_ChatRoom_slot ) );

        //ChatRooms (Groups) will be stored using their full HLA
        //names; the leading character will be added/stripped at the
        //client/server agent interface

        //Adding the ChatRoom's name to the list every time we get an
        //update could be a waste of time; fortunately, _ChatRoom
        //attributes never change so we'll only get this event after
        //a discovery.
        //We will, however, get it once for each Client...
        if ( _ChatRoom.name.toString().equals( HLA._name_waiting_room_ChatRoom ) )
            blfdatGrpList.add( _ChatRoom.name.toString().substring(1) );
    }
}

//end plnHLA_ChatRoom_ReflectAttributeValueUpdate
The `plnHLA_ChatRoom_Removal` plan handles ChatRoom object instance removals.

```java
// File: plnHLA_ChatRoom_Removal.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAremoveObjectInstance;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * Handles the removal of ChatRoom objects.
 */
public plan plnHLA_ChatRoom_Removal
  extends Plan
{
  logical Object _HLA;
  Server.HLACHat HLA;
  #handles event evtHLAremoveObjectInstance ev;
  #uses interface Server server;
  #reads data blfHLA datHLA;

  static boolean relevant(evtHLAremoveObjectInstance ev)
  {
    return true;
  }

  context()
  {
    datHLA.get( ev.identifier, _HLA ) &&
      (! (HLA = (Server.HLACHat)_HLA.as_object()) != null) &&
      HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject.equals( HLA._ocl_ChatRoom ));
  }

  #reasoning method
  body()
  {
    // Server.HLACHat HLA = (Server.HLACHat)_HLA.as_object();
    Server.HLACHat.aChatRoom _ChatRoom =
      (Server.HLACHat.aChatRoom)HLA._theChatRooms.remove( ev.theObject );
    //In the HLA GUI Chat, the <General> ChatRoom remains on the list
    //of ChatRooms even whilst non-existent. There is no such list here.
    //  if (_ChatRoom.name.toString().equals( HLA._name_general_ChatRoom )) return true;
    //  This must be true
    //  remove group from groupList
    //  blfdatGrpList.remove( _ChatRoom.name.toString() );
  }
}
//end plnHLA_ChatRoom_Removal
```
The `plnHLA_WaitingRoom_NameReservation_Failed` plan handles the failed outcome of the Waiting Room name reservation.

// File: plnHLA_WaitingRoom_NameReservation_Failed.java
package server;

import ca.gc.drdc.rddc.hla.rti1516.jack.evtHLAobjectInstanceNameReservationFailed;
import ca.gc.drdc.rddc.hla.rti1516.jack.blfHLA;

/**
Handles reservation of the "waiting_room" ChatRoom name.
*/
public plan plnHLA_WaitingRoom_NameReservation_Failed extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    //handles event evtHLAobjectInstanceNameReservationFailed ev;
    //uses interface Server server;
    //reads data blfHLA datHLA;

    static boolean relevant(evtHLAobjectInstanceNameReservationFailed ev)
    {
        return
        ev.objectName.equals(Server.HLAChat._name_waiting_room_ChatRoom);
    }

    context()
    {
        datHLA.get(ev.identifier, _HLA) &&
        ( (HLA = (Server.HLAChat)._HLA.as_object()) != null);
    }

    //reasoning method
    body()
    {
        //Name reservation failed, which means there is an already extant instance
        HLA.sem_chatRoom_discovery.clearChanged();
        HLA.waiting_room_chatRoom.subscribed = true;
        HLA._rtiAmbassador.subscribeObjectClassAttributes(
            HLA.ocl_chatRoom, HLA.oahs_chatRoom_forUpdate );
        @wait_for( new aos.jack.util.cursor.Change(
            HLA.sem_chatRoom_discovery, HLA.sem_chatRoom_discovery.hasChanged() )
        );

        //Signal the waiting plnLoginUser that we're done
        HLA.sem_name_reservation.setChanged();
    }
}
//end plnHLA_WaitingRoom_NameReservation_Failed
The plan handles the successful outcome of the Waiting Room name reservation.

```java
// File: plnHLA_WaitingRoom_NameReservation_Succeeded.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAobjectInstanceNameReservationSucceeded;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
Handles reservation of the "waiting_room" ChatRoom name.
*/
public plan plnHLA_WaitingRoom_NameReservation_Succeeded extends Plan
{
    logical object _HLA;
    Server.HLAChat HLA;
    #handles event evtHLAobjectInstanceNameReservationSucceeded ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAobjectInstanceNameReservationSucceeded ev)
    {
        return ev.objectName.equals(Server.HLAChat._name_waiting_roomChatRoom);
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        ( _HLA = (Server.HLAChat)_HLA.as_object() ) != null;
    }
```
# reasoning method

body()
{
    // Having reserved the name, we know we're the first federate
    // to reach this point, so we must create the _waiting_room
    ChatRoom.
    HLA._waiting_room_ChatRoom.owned = true;
    HLA._waiting_room_ChatRoom.subscribed = true;
    // HLA._waiting_room_ChatRoom.divestng = false;
    @wait_for { HLA._sem_theChatRooms.planWait() }
    HLA._theChatRooms.put( HLA._waiting_room_ChatRoom.handle = HLA._rtiAmbassador.registerObjectInstance( HLA._och_ChatRoom,
    HLA._waiting_room_ChatRoom.name.toString() ),
    HLA._waiting_room_ChatRoom );
    HLA._sem_theChatRooms.signal();
    HLA._rtiAmbassador.subscribeObjectClassAttributes( HLA._och_ChatRoom, HLA._och_ChatRoom_forUpdate );
    // Signal the waiting plnLoginUser that we're done
    HLA._sem_name_reservation.setChanged();
}
} // end plnHLA_WaitingRoom_NameReservation_Succeeded

The plnHLA_ParticipantEntersGeneralChatRoom plan handles the subtask of “entering” a Client’s Participant avatar into the General ChatRoom.

// File: plnHLA_ParticipantEntersGeneralChatRoom.java
package server;

import ca.gc.drdc.rddc.hla.rti1516.*;
import hla.rti1516.*;

/** Handles the subtask of entering the (owned) Participant into the
    General ChatRoom. */
public plan
plnHLA_ParticipantEntersGeneralChatRoom
    extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    #handles event evtHLA_ParticipantEntersGeneralChatRoom ev;
    #posts event evtHLA_AcquireChatRoomRegistry ev_CRR;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant( evtHLA_ParticipantEntersGeneralChatRoom ev
    )
    {
        return true;
    }
context() {
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLachat)_HLA.as_object()) != null);
}

#reasoning method
body() {
    //Complete log-in process by entering owned Participant into the General ChatRoom
    //Does the General ChatRoom exist?
    @wait_for( HLA._sem_theChatRooms.planWait() );
    {
        HLA._general_chatRoom = HLA.SeekChatRoomBySlot( HLA._slot_general_chatRoom );
        if (null == HLA._general_chatRoom) {
            @subtask( ev_CRR.create( ev.identifier ) );
            synchronized(AcquireChatRoomRegistry()) {
                HLA._general_chatRoom = HLA.newChatRoom( HLA._name_general_chatRoom, HLA._slot_general_chatRoom );
                HLA._general_chatRoom = HLA.getNewChatRoom( HLA._name_general_chatRoom, HLA._slot_general_chatRoom );
                HLA._general_chatRoom.owned = HLA._general_chatRoom.subscribed = true;
                // HLA._general_chatRoom.divesting = false;
                HLA._chatRooms.put( HLA._general_chatRoom.handle = HLA._rtiAmbassador.registerObjectInstance( HLA._och_chatRoom ), HLA._general_chatRoom );
                //No need to add the General ChatRoom to the _ChatRoomRegistry, as it is already listed
            } //synchronized(AcquireChatRoomRegistry())
            HLA._sem_ChatRoomRegistry.signal();
        } else {
            } //if
        } //synchronized(_theChatRooms)
    HLA._sem_theChatRooms.signal();

    //This association will trigger Attribute Scope Advisories, Discovery:
    HLA._myChatRoom = HLA._general_chatRoom;
    HLA._me.chat_room_slot.setValue( HLA._slot_general_chatRoom );
    HLA._me.logged_in = !(HLA._me_logging_in = false);
    HLA._rtiAmbassador.associateRegionsForUpdates( HLA._me.handle, HLA_.asrpl_Participant_current );
    HLA._rtiAmbassador.subscribeObjectClassAttributesWithRegions( HLA._och_Participant, HLA._asrpl_Participant_current );

    //Subscription will refresh subscribed Participants;
    //since we use Auto-Provide, an explicit Update request is not needed
}
//Subscribe to the interaction
//We'll be listening to interactions that come in on our user-handle channel in addition to the chat-room-slot channel
//It is the interaction sender's responsibility to pick the channels to use
HLA._rh_myParticipantRegion = HLA._rtiAmbassador.createRegion(HLA._dhs_UserHandleSlotsSet);
//setRangeBounds must be invoked at least once for each dimension specified.
//Only then can commitRegionModifications be invoked to turn the region template into a region specification.
HLA._rtiAmbassador.setRangeBounds(HLA._rh_myParticipantRegion, HLA._dhs_UserHandleSlots, new RangeBounds(HLA._me.user_handle.getValue(), HLA._me.user_handle.getValue() + 1));
HLA._rhs_myParticipantRegion = HLA._rtiAmbassador.getRegionHandleSetFactory().create();
HLA._rhs_myParticipantRegion.add(HLA._rh_myParticipantRegion);
HLA._rtiAmbassador.commitRegionModifications(HLA._rhs_myParticipantRegion);
//subscribeInteractionClassWithRegions done by plnLoginUser

The plnHLA_Participant_AttributeScopeAdvisory_In plan handles the attribute scope advisory for a Participant instance coming into scope.

// File: plnHLA_Participant_AttributeScopeAdvisory_In.java
package server;

import ca.gc.drdc.rddc.hla.rti1516.jack.blfHLA;
import ca.gc.drdc.rddc.hla.rti1516.jack.evtHLAattributesInScope;

/**
 * Handles the Attribute Scope Advisories for Participants.
 */
public plan
plnHLA_Participant_AttributeScopeAdvisory_In
extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    // handles event evtHLAattributesInScope ev;
    // uses interface Server server;
    // reads data blfHLA datHLA;

    static boolean relevant(evtHLAattributesInScope ev)
    {
        return true;
    }
}
context()
{
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLACHat)_HLA.as_object()) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject )
    .equals( HLA._och_Participant );
}

#reasoning method
body()
{
    //The Participant has just come into or gone out of scope,
    //which means either we or it has changed ChatRooms
    //it cannot have been truly deleted
    @wait_for( HLA._sem_theParticipants.planWait() );
    Server.HLACHIcChat.aParticipant _Participant =
        HLA.SeekParticipant( ev.theObject );
    HLA._sem_theParticipants.signal();
    //If we could localDelete Participants as they go out of scope,
    //we would always get a Discovery right before the InScope
    //callback; but we can’t do that because of the pRTI
    //RTIambassador-Federate Service Thread synch bug (putting the
    //Federate Service Thread in any kind of synch-wait state (a
    //Semaphore, say) causes a federate-triggering RTIambassador
    //call by *any* thread to hang, even if the latter
    //RTIambassador-caller isn’t owning any monitors).
    //So we use the inScope property of aParticipant to work around
    //We know we do not own theObject, since we wouldn’t get any
    advisories in that case
    _Participant.inScope = true;
    //The Java Chat federate uses this event to add/remove
    //Participants to its drop-down list of private message targets;
    //the JACK Chat federates resolve that list only when a Client
    //sends an evtRequestGrpWho, so there is no further processing
    //required here.
}

//end plnHLA_Participant_AttributeScopeAdvisory_In
The `plnHLA_Participant_AttributeScopeAdvisory_Out` plan handles the attribute scope advisory for a Participant instance going out of scope.

```java
// File: plnHLA_Participant_AttributeScopeAdvisory_Out.java
package server;

import ca.gc.drdc.rddc.hla.rtl1516.
import ca.gc.drdc.rddc.hla.rtl1516.
/**
Handles the Attribute Scope Advisories for Participants.
*/
public plan plnHLA_Participant_AttributeScopeAdvisory_Out extends Plan {
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAattributesOutOfScope ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant (evtHLAattributesOutOfScope ev) {
        return true;
    }

    context () {
        datHLA.get( ev.identifier, _HLA ) &&
        ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
        HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
        ).equals( HLA._och_Participant );
    }
```
#reasoning method

body()
{
    // The Participant has just gone out of scope,
    // which means either we or it has changed ChatRooms
    // (it cannot have been truly deleted)
    @wait_for( HLA._sem_theParticipants.planWait() );
    Server.HLAChat.aParticipant _Participant =
    HLA.SeekParticipant(ev.theObject);
    HLA._sem_theParticipants.signal();
    // If we could localDelete Participants as they go out of scope,
    // we would always get a Discovery right before the InScope
    // callback, but we can't do that because of the pRTI
    // RTIambassador-Pederate Service Thread synth bug (putting the
    // Pederate Service Thread in any kind of synth-wait state (a
    // Semaphores, say) causes a federate-triggering RTIambassador
    // call by *any* thread to hang, even if the latter
    // RTIambassador-caller isn't owning any monitors).
    // So we use the inScope property of aParticipant to work around
    // it.
    // We know we do not own theObject, since we wouldn't get any
    advisories in that case
    _Participant.inscope = false;
    // The Java Chat federate uses this event to add/remove
    // Participants to its drop-down list of private message targets;
    // the JACK Chat federates resolve that list only when a Client
    // sends an evtRequestGrpWho, so there is no further processing
    // required here.
}

// end plnHLA_Participant_AttributeScopeAdvisory_Out

The plnHLA_Participant_Discovery plan handles the discovery of other Participant instances.

// File: plnHLA_Participant_Discovery.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAdiscoverObjectInstance;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * Handles the discovery of Participant objects
 */
public plan
plnHLA_Participant_Discovery
extends Plan
{
    logical Object _HLA;
    Server.HLAChat HLA;
    #handles event evtHLAdiscoverObjectInstance ev;
    #uses interface Server server;
    #uses data blfHLA datHLA;
static boolean relevant(evtHLAdiscoverObjectInstance ev) {
    return true;
}

context()
{
    datHLA.get( ev.identifier, _HLA )
    ( (HLA = (Server.HLAchat) HLA.as_object()) != null )
    ev.theObjectClass.equals( HLA._och_Participant );
}

#reasoning method
body()
{
    logical Object $HLA2;

    Server.HLAchat.aParticipant _Participant =
    HLA.getNewParticipant();
    // Server.HLAchat.aParticipant _Participant = HLA.new
    Participant();
    _Participant.subscribed = true;
    // _Participant.inscope = false; //Default; the InScope advisory
    will follow discovery in any case
    _Participant.handle = ev.theObject;
    // _Participant.name.setValue( HLA._rtiAmbassador.getObjectInstanceName( ev.theObject ) );
    _Participant.name.setValue( ev.objectName );

    @wait_for{ HLA._sem_theParticipants.planWait() };
    HLA._TheParticipants.put( ev.theObject, _Participant);
    HLA._sem_theParticipants.signal();
    //When we first tried to maintain the old JACK belief sets, this
    //got complicated here.
    //At this point we don't know which ChatRoom slot the
    //Participant is in, so we couldn't add to blfdatGroups (mapping
    //of usernames to groups); we'd have to await an Update for
    //that.
    //Whether the Participant discovered by this JACK client is
    //another JACK client or not we can tell because a JACK client
    //first adds itself to blfdatHLA before logging in. Not so with
    //blfdatUsers, which is added to after the login is complete.
    //
    //Now, in using only the blfdatHLA, we don't need to bother.
    //What the Participant objects map to becomes irrelevant;
    //interaction send and receive is done through the RTI instead
    //of through direct Server intervention. Basically, the Server
    //relays the send to the RTI, which then callbacks the pertinent
    //JACK clients; this goes through the Server again, which then
    //delivers the interaction instances to each Client instance.

    //if (HLA._me_shutting_down) return true;
    //we use the Auto-Provide switch, so an explicit Update request
    //is not needed
}
The plnHLA_Participant_NameReservation_Failed plan handles the failed outcome of Participant name reservations.

```java
// File: plnHLA_Participant_NameReservation_Failed.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAobjectInstanceNameReservationFailed;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/**
 * Handles outcome of Participant (username) name reservation request.
 */
public plan
plnHLA_Participant_NameReservation_Failed
extends Plan
{
  logical object _HLA;
  Server.HLAchat HLA;
  #handles event evtHLAobjectInstanceNameReservationFailed ev;
  #posts event evtHLA_ParticipantEntersGeneralChatRoom ev_PEGCR;
  #uses interface Server server;
  #reads data blfHLA dstHLA;

  static boolean relevant( evtHLAobjectInstanceNameReservationFailed ev)
  {
    return ( ev.objectName.charAt(0) == 'p' );
  }

  context()
  {
    datHLA.get( ev.identifier, _HLA ) &
    ( _HLA = (Server.HLAchat)_HLA.as_object() != null);
  }
```
### reasoning method

```java
body()
{
/**
  * blfdatUsers does not list non-JACK clients (usernames)
*/
//Prepare the general chat room association
HLA._rhs_current_ChatRoom =
  HLA._rtiAmbassador.getRegionHandleSetFactory().create();
//Arrival ( <General> ) ChatRoom will be slot 2, so we set up
"current" accordingly
HLA._rh_current_ChatRoom = HLA._rh_general_ChatRoom;
HLA._rh_current_ChatRoom.add( HLA._rh_current_ChatRoom );
HLA._rtiAmbassador.commitRegionModifications( HLA._rhs_current_ChatRoom ); //Won't complain if there are no mods to commit
HLA._asrspl_Participant_current =
  HLA._rtiAmbassador.getAttributeSetRegionSetPairListFactory().create();
  HLA._asrspl_Participant_current.add( new AttributeRegionAssociation( HLA._oahs_Participant_forUpdate, HLA._rhs_current_ChatRoom ) );
  //Note that the association has not been applied yet.
  //Name reservation failed, which means there is an already extant instance.
  //At this point, we know the desired Participant object exists -
we just don't know where it is.
  //If it is logged-out, it'll be in the waiting_room and we will
  //discover it there and can then acquire it.
  //It is logged-in by someone else, it'll be in some other
  //ChatRoom (2+) and we won't discover it and cannot use it.
  //There is no "fail-fast" in that situation, however.
  //It is reasonably safe to assume that if we don't know of the
  //sought Participant, it must be logged in by someone else.
  //Is the object known?
try
{
  @wait_for( HLA._sem_theParticipants.planWait() );
  //getObectInstanceHandle will throw an exception if the
  //object is unknown.
  //We put a try-finally around it to avoid having to signal()
  within the outer catch block
  try
  {
    _Participant = HLA.SeekParticipant( HLA._rtiAmbassador.getObjectInstanceHandle( HLA._me.name.toString() ) );
  }
  finally
  {
    HLA._sem_theParticipants.signal();
  }
  //No exception occurred, therefore the object is known; could
  //be anywhere, however
```
// If freshly discovered, we could wait for its user_handle to
// be updated by reflectAttributeValues (this is the only
// field guaranteed to change) so that we can then safely
// look up its logged_in or chat_room_slot_values.
// Instead, we'll request ownership right away; a logged-in
// Participant will fail.

// It could be already owned (if it was an unused Participant
we just happened to have the custody of)
if (!Participant.owned)
{
    // For a Participant to be eligible for acquisition, it
    // must be in the _waiting_room and therefore inscope
    if (Participant.inscope)
    {
        HLA.sem_Participant_acquisition.clearChanged();
        // Before acquiring, set up update regions so they do
        not lapse during ownership transfer
        HLA._rtiAmbassador.associateRegionsForUpdates(
            Participant.handle, HLA._arspl_Participant_nowhere );
        HLA._rtiAmbassador.associateRegionsForUpdates(
            Participant.handle, HLA._arspl_Participant_waiting_room );
        HLA._rtiAmbassador.attributeOwnershipAcquisition(
            Participant.handle, HLA._oahs_Participant, null );
        @wait_for( new aos.jack.util.cursor.Change(
            HLA.sem_Participant_acquisition,
            HLA.sem_Participant_acquisition.hasChanged() ) );

        // The plan handling the acquisition request outcome
        // will set HLA._me_logging_in to false (and
        // HLA._me.name.setValue("") in case of failure (which
        // happens if the object was discovered in the
        // _waiting_room and then logged-in by someone else
        // before we could get to it).
    } else {
        HLA._me_logging_in = false;
        HLA._me.name.setValue("");
    } // if
} // if

if (HLA._me_logging_in)
{
//Acquisition succeeded: that Participant was available
  @wait_for( HLA._sem_me.planWait() )
  {
    HLA._me = _Participant;
    //Enter the <General> ChatRoom
    HLA._me.divesting = false; //could be still true: will
    // be fixed later
    HLA._me.logged_in.setBoolean( HLA._me.owned = true );
    HLA._me.user_handle.setValue( HLA._me.handle.hashCode() )
  }
}
    //Removing _me from the waiting_room will trip
Attribute Scope Advisories
  HLA._rtiAmbassador.unassociateRegionsForUpdates( HLA._me.handle, HLA._asrsplParticipant_waiting_room);
    //At this point, _me.handle is only associated with
  _asrspl_Participant_nowhere
  HLA._me.chat_room_slot.setValue( HLA._slot_nowhere_chatRoom )
  } //synchronized(_me)
  } //if
} catch (ObjectInstanceNotKnown ex) {
  //The Participant was unknown: it cannot be in the
  waiting_room
  HLA._me_logging_in = false;
  HLA._me.name.setValue(""");
} //try

    //Note that a created Participant goes only _nowhere; it does
  not pass through _waiting_room.
    //At this point the _Participant is only _nowhere.
  if (HLA._me_logging_in)
    {
    @subtask( ev_PEGCR.create( ev.identifier ) )
    }
    //Signal the waiting plnLoginUser that we're done
  HLA._sem_name_reservation.setChanged( );
}
//end plnHLA_Participant_NameReservation_Failed
The `plnHLA_Participant_NameReservation_Succeeded` plan handles the successful outcome of Participant name reservations.

```java
// File: plnHLA_Participant_NameReservation_Succeeded.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAobjectInstanceNameReservationSucceeded;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/**
Handles outcome of Participant (username) name reservation request.
*/
public plan plnHLA_Participant_NameReservation_Succeeded
  extends Plan
{ logical object _HLA;
  Server.HLAChat HLA;
  #handles event evtHLAobjectInstanceNameReservationSucceeded ev;
  #posts event evtHLA_ParticipantEntersGeneralChatRoom ev_PEGCR;
  #uses interface Server server;
  #reads data blfHLA dstHLA;

  static boolean relevant(evtHLAobjectInstanceNameReservationSucceeded ev )
  { return ( ev.objectName.charAt(0) == 'p' ); }

  context()
  { dstHLA.get( ev.identifier, _HLA ) &&
      ( _HLA = (Server.HLAChat)_HLA.as_object() ) != null );
  }
```
#reasoning method

body()

/**
 * blfdatUsers does not list non-JACK clients (usernames)
 */

//Prepare the general chat room association
HLA._rtiAmbassador.getRegionHandleSetFactory().create();

//Arrival (<General>) ChatRoom will be slot 2, so we set
"current" accordingly
HLA._rh_current_ChatRoom = HLA._rtiAmbassador.getRegionHandleSetFactory().create();
HLA._rh_current_ChatRoom.add(HLA._rh_current_ChatRoom);
HLA._rtiAmbassador.commitRegionModifications(HLA._rh_current_ChatRoom);

HLA._asrspl_Participant_current = HLA._rtiAmbassador.getAttributeSetRegionSetPairListFactory().create();
HLA._asrspl_Participant_current.add(new AttributeRegionAssociation(HLA._oahe_Participant_forUpdate, HLA._rh_current_ChatRoom));

//Note that the association has not been applied yet.
//For chat_room_slot filtering, we associate all "forUpdate"
attributes to _ChatRooms regions.
//We'd get InvalidRegionContext when subscribing if we included
the DeletePrivilege.
@wait_for(HLA._sem_me.planWait());
{
    HLA._me.logged_in.setBoolean(HLA._me.owned = HLA._me.subscribed = !(HLA._me.inscope = HLA._me.divesting = false));
    HLA._me.chat_room_slot.set_value(HLA._slot_nowhere_ChatRoom);
    //The <General> ChatRoom must have been "waiting_room"
    // (slot 1 ) previously. If there were any federates
    // subscribed to the creation region (nowhere), there would be
    // Discovery and Auto-Update would trigger our
    // InstanceAttributeResponder and all that;
    // Hence the atomic register-and-put; the synchronized(_me)
    // would delay a bit until the _me.user_handle is set.
    // But since no-one subscribes to nowhere, there is no rush
    @wait_for(HLA._sem_theParticipants.planWait());
    {
        HLA._theParticipants.put(HLA._me.handle = HLA._rtiAmbassador.registerObjectInstanceWithRegions(HLA._och_Participant, HLA._asrspl_Participant_nowhere, HLA._me.name.toString()), HLA._me);
    } //synchronized(_theParticipants)
    HLA._sem_theParticipants.signal();
    HLA._me.user_handle.setValue(HLA._me.handle.hashCode());
    } //synchronized(_me)
    HLA._sem_me.signal();
// Note that a created Participant goes only _nowhere_; it does
// not pass through _waiting_room_.
// At this point the _Participant is only _nowhere_.
if (HLA._me_logging_in)
{
  @subtask( ev_PEGCR.create( ev.identifier ) );
} //if
// Signal the waiting plnLoginUser that we're done
HLA.sem_name_reservation.setChanged();
}

The plnHLA_Participant_NameReservation_Succeeded plan handles the
failed outcome of a Participant attribute ownership acquisition attempt.

// File: plnHLA_Participant_OwnershipAcquisitionFailed.java
package server;

import ca.gc.drdc.rddc.hla.rti1516.jack.evtHLAattributeOwnershipUnavailable;
import ca.gc.drdc.rddc.hla.rti1516.jack.blfHLA;

/**
Handles the AttributeOwnershipUnavailable notification for
Participants
*/
public plan
plnHLA_Participant_OwnershipAcquisitionFailed
  extends Plan
{
  logical Object _HLA;
  Server.HLAchat HLA;
  //Handles event evtHLAattributeOwnershipUnavailable ev;
  //uses interface Server server;
  //reads data blfHLA datHLA;
  static boolean relevant(evtHLAattributeOwnershipUnavailable ev)
  {
    return true;
  }
  context()
  {
    datHLA.get( ev.identifier, _HLA ) &
      ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &
      //
      ((Server.HLAchat)_HLA.as_object()).och_ChatRoomRegistry );
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject
      ).equals( HLA._och_Participant );
  }
body()
{
    HLA.sem_Participant_acquisition.setChanged();
}

//end plnHLA_Participant_OwnershipAcquisitionFailed

The plnHLA_Participant_OwnershipAcquisitionNotification plan handles the successful outcome of a Participant attribute ownership acquisition attempt.

// File: plnHLA_Participant_OwnershipAcquisitionNotification.java
package server;
import ca.gc.drdoc.rddc.hla.rti1516.jack.evtHLAattributeOwnershipAcquisitionNotification;
import ca.gc.drdoc.rddc.hla.rti1516.jack.blfHLA;

/** Handles the attribute ownership notification for Participants */
public plan plnHLA_Participant_OwnershipAcquisitionNotification
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAattributeOwnershipAcquisitionNotification ev;
    #posts event evtHLA_ForceDivest eVHLA_ForceDivest;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAattributeOwnershipAcquisitionNotification ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        ( (HLA = (Server.HLAchat) _HLA.as_object()) != null ) &&
        HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_Partner);
    }
}
# reasoning method
body()
{
    @wait_for( HLA._sem_theParticipants.planWait() );
    synchronized(HLA._theParticipants)
    Server.HLachat.aParticipant _Participant = HLA.SeekParticipant(ev.theObject);
    //end of synchronized(HLA._theParticipants)
    HLA._sem_theParticipants.signal();
    _Participant.owned = !_Participant.inscope =
    //There are two circumstances where we acquire Participants:
    //When we want to log in, and when another federate logs out and
    shuts down.
    //In both cases we set up the ForUpdate Regions such that the
    //Participant arrives associated with nowhere and
    //waiting room. In the first circumstance, the
    //planHLAParticipant_NameReservation waits on the
    //sem_Participant_acquisition and will move the Participant to
    //the_general ChatRoom.
    HLA.sem_Participant_acquisition.setChanged();
    //Just in case we somehow managed to acquire whilst switching to
    shutting down mode
    if (HLA._me_shutting_down) server.postEvent(
        eVHLA_ForceDivest.create( ev.identifier ) );
}
//end planHLA_Participant_OwnershipAcquisitionNotification

The planHLA_Participant_OwnershipAssumptionRequest plan handles
ownership assumption requests for Participant object attributes.

// File: planHLA_Participant_OwnershipAssumptionRequest.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLA_requestAttributeOwnershipAssump
tion;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
Handles OwnershipAssumptionRequests for Participants
*/
public plan
planHLA_Participant_OwnershipAssumptionRequest
extends Plan
{
    logical Object _HLA;
    Server.HLachat HLA;
    //handles event evtHLA_requestAttributeOwnershipAssumption ev;
    //uses interface Server server;
    //reads data blfHLA datHLA;
static boolean relevant(evtHLARequestAttributeOwnershipAssumption ev) {
    return true;
}

class context() {
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
    HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_Participant );
}

# reasoning method
body() {
    // Server.HLAchat HLA = (Server.HLAchat)_HLA.as_object();
    // Decline if we're in the process of shutting down (and thus
divesting).
    // Note that we return true because the plan was a success
    // even though we simply decided to ignore the request.
    if (HLA.me.shutting_down) return true;
    @wait_for( HLA.sem_theParticipants.planWait() );
    Server.HLAchat.aParticipant _Participant = HLA.SeekParticipant( ev.theObject );
    HLA._sem_theParticipants.signal();
    // Acquiesce if able
    // If not subscribed, probably not up to date -- decline
    if (! _Participant.subscribed ) return true;
    // Note that the divesting federate will have offered ownership
    // to all federates, and that even if they all acquiesce, only
    // one will receive an ownershipAcquisitionNotification; if we
    // use attributeOwnershipAcquisition, there won't be any negative
    // feedback if we fail to get ownership -- and we'll have an
    // outstanding acquisition request.
    // This is why it is preferable to use
    attributeOwnershipAcquisitionIfAvailable
    //
    // Before acquiring, set up update regions so they do not lapse
during ownership transfer
    HLA.rtiAmbassador.associateRegionsForUpdates( _Participant.handle, HLA._asrspl_Participant_nowhere );
    HLA.rtiAmbassador.associateRegionsForUpdates( _Participant.handle, HLA._asrspl_Participant_waiting_room );
    HLA.rtiAmbassador.attributeOwnershipAcquisitionIfAvailable( _Participant.handle, HLA._oahs_Participant ); // or theObject
}
The `plnHLA_Participant_OwnershipDivestitureConfirmationRequest` plan handles requests for confirmation of divestiture of Participant object attributes.

```java
// File: plnHLA_Participant_OwnershipDivestitureConfirmationRequest.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLArequestDivestitureConfirmation;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;

/**
 * Handles requests for confirmation of ownership divestiture of Participants
 */
public plan plnHLA_Participant_OwnershipDivestitureConfirmationRequest extends Plan
{
    logical object _HLA;
    Server.HLAChat HLA;
    // handles event evtHLArequestDivestitureConfirmation ev;
    // uses interface Server server;
    // reads data blfHLA dstHLA;

    static boolean relevant(evtHLArequestDivestitureConfirmation ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
            ( (HLA = (Server.HLAChat)_HLA.as_object()) != null ) &&
            HLA.rtiAmbassador.getKnownObjectClassHandle( ev.theObject).
                .equals( HLA._och_Participant );
    }
}
```
# reasoning method

body()
{
    @wait for( HLA._sem_theParticipants.planWait() );
    Server.HLachat.aParticipant _Participant =
    HLA.SeekParticipant(ev.theObject);
    // synchronized(HLA._theParticipants)
    {
        HLA._rtiAmbassador.confirmDivestiture( _Participant.handle,
        HLA._oahs_Participant, null );
        _Participant.owned = _Participant.divesting = false;
        //synchronized(HLA._theParticipants)
        HLA._sem_theParticipants.signal();
        //There'll be a divestiture semaphore of some sort when logging
        off
        HLA._sem_Participant_divestiture.setChanged();
    }
}
//end plnHLA_Participant_OwnershipDivestitureConfirmationRequest

The plnHLA_Participant_OwnershipReleaseRequest plan handles ownership release requests for Participant object attributes.

// File: plnHLA_Participant_OwnershipReleaseRequest.java
package server;

import ca.gc.drdc.rddc.hla.rtl1516.blfHLA;
import ca.gc.drdc.rddc.hla.rtl1516.evtHLA_requestAttributeOwnershipRelease;

/**<p handles requests for ownership release of Participants */
public plan
plnHLA_Participant_OwnershipReleaseRequest
extends Plan
{
    logical Object _HLA;
    Server.HLachat HLA;
    //handles event evtHLA_requestAttributeOwnershipRelease ev;
    //uses interface Server server;
    //reads data blfHLA datHLA;
    static boolean relevant(evtHLA_requestAttributeOwnershipRelease ev)
    {
        return true;
    }
}
context()
{
    datHLA.get(ev.identifier, HLA) &&
    (!HLA = (Server.HLACHat).HLA.as_object()) != null) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle(ev.theObject)
    .equals(HLA._och_Participant);
}

#reasoning method
body()
{
    //Acquiesce
    @wait_for(HLA._sem_theParticipants.planWait());
    //synchronized(HLA._theParticipants)
    |
    Server.HLACHat.aParticipant _Participant =
    HLA.SeekParticipant(ev.theObject);
    //If me, refuse (there is no RTI call to tell the requester
to piss off).
    //Here we fail the plan as a means of signalling our refusal.
    //Note that the JACK federates resign and shut down when not
    //logged-in, so the Java test on _me_logged_in becomes
    //redundant here.
    _Participant.equals(HLA._me)) return false;
    //Otherwise, acquiesce
    //Plan succeeds even though divestiture fails
    if (HLA._oahs_Participant.containsAll(
        HLA._rtiAmbassador.attributeOwnershipDivestitureIfWanted(
            _Participant.handle, HLA._oahs_Participant))) return true;
    _Participant.owned = _Participant.divesting =
    _Participant.inscope = false;
    //Region associations are automatically lost along with
    ownership
    //synchronized(_theParticipants)
    HLA._sem_theParticipants.signal();
    HLA._sem_Participant_divestiture.setChanged();
}

//end plnHLA_Participant_OwnershipReleaseRequest
The `plnHLA_Participant_ProvideAttributeValueUpdate` plan handles requests for Participant instance attribute value updates.

// File: plnHLA_Participant_ProvideAttributeValueUpdate.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.evthLAprovideAttributeValueUpdate;
import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import hla.rti1516.*;

/**
 * Handles attribute value update requests received from the RTI for the Participants.
 */
public plan plnHLA_Participant_ProvideAttributeValueUpdate
  extends Plan
{
  logical Object _HLA;
  Server.HLAChat HLA;
  #handles event evthLAprovideAttributeValueUpdate ev;
  #uses interface Server server;
  #reads data blfHLA datHLA;

  static boolean relevant(evthLAprovideAttributeValueUpdate ev)
  {
    return true;
  }

  context()
  {
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAChat)HLA.as_object()) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject ).equals( HLA._och_Participant );
  }
}
# reasoning method

```java
body()
{
    @wait_for( HLA._sem_theParticipants.planWait() );
    Server.HLAchat.aParticipant _Participant =
    HLA.SeekParticipant(ev.theObject);
    HLA._sem_theParticipants.signal();
    AttributeHandleValueMap _ahvm_attributeValues =
    HLA._rtiAmbassador.getAttributeHandleValueMapFactory().create(
    HLA._cahs_Participant_forUpdate.size() );
    // In the HLA Chat federates, it is crucial to synchronize on
    // _Participant here, since the object's registration may trigger
    // this handler right away (because of Auto-Provide) --and the
    // object cannot fill its user_handle field in the same atomic
    // operation.
    // In JACK Clients, on the other hand, the callbacks are queued
    // up as posts on a single agent (the Server), hence this is not
    // a problem.
    _ahvm_attributeValues.put( HLA._cah_Participant_logged_in,
        _Participant.logged_in.toByteArray() );
    _ahvm_attributeValues.put( HLA._cah_Participant_user_handle,
        _Participant.user_handle.toByteArray() );
    _ahvm_attributeValues.put( HLA._cah_Participant_chat_room_slot,
        _Participant.chat_room_slot.toByteArray() );
    HLA._rtiAmbassador.updateAttributeValues(_Participant.handle,
        _ahvm_attributeValues, null );
}
@end
```

The `plnHLA_Participant_ProvideAttributeValueUpdate` plan handles Participant instance attribute value update reflections.

```java
package server;

import ca.gc.drdc_rddc.hla.rtl1516.jack.evtHLAreflectAttributeValueValues;
import ca.gc.drdc_rddc.hla.rtl1516.jack.blfHLA;

/**
 * Handles attribute value updates received from the RTI for the
 * Participants.
 */
public plan
plnHLA_Participant_ReflectAttributeValueUpdate
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat _HLA;
    #handles event evtHLAreflectAttributeValueValues ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;
```
static boolean relevant(evtHLAreflectAttributeValues ev)  
{
    return true;
}

class()
{
    datHLA.get( ev.identifier, _HLA ) &&
    ( (HLA = (Server.HLAchat)_HLA.as_object()) != null ) &&
    HLA._rtiAmbassador.getKnownObjectClassHandle( ev.theObject )
    .equals( HLA._och_Participant );
}

(reasoning method body)
{
    //HLA.privilegeToDeleteObject has no content, so we won't decode it
    @wait_for( HLA._sem_theParticipants.planWait() );
    Server.HLAchat.aParticipant _Participant =
    HLA.SeekParticipant( ev.theObject);
    HLA._sem_theParticipants.signal();
    {
        //The attribute Name matches the object's HLA name and is updated at Discovery time
        _Participant.name.setValue( HLA._rtiAmbassador.getObjectInstanceName( ev.theObject ) );
        _Participant.logged_in.decode( (byte[]) ev.theAttributes.get( HLA._och_Participant_logged_in ) );
        _Participant.user_handle.decode( (byte[]) ev.theAttributes.get( HLA._och_Participant_user_handle ) );
        _Participant.chat_room_slot.decode((byte[]) ev.theAttributes.get( HLA._och_Participant_chat_room_slot ) );
        
        //This is where the JACK belief set model would need to put the Participant in the proper ChatRoom.
        //Using either the _ChatRoomRegistry.list or HLA.SeekChatRoomBySlot, one can find which ChatRoom name
        //corresponds to the Participant's chat_room_slot, then one can add to blfdatGroups.
    }
    //Unlike the HLA Chat federates, a JACK Client that is not logged in cannot subscribe
    //if (!HLA._me_logged_in) return true;
}

//end plnHLA_Participant_ReflectAttributeValueUpdate
The plnHLA_Interaction plan handles reception of messages.

// File: plnHLA_Interaction.java
package server;
import hla.rti1516.*;
import ca.gc.drdc_rddc.hla.rti1516.omt.*;
import java.util.Iterator;
import ca.gc.drdc_rddc.hla.rti1516.blfHLA;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAreceiveInteraction;
import client.evtMessageUsr;

/**
Handles the evtHLAinteraction event
*/
public plan
plnHLA_Interaction
extends Plan
{
    logical Object _HLA;
    Server.HLachat HLA;
    #handles event evtHLAreceiveInteraction ev;
    #sends event evtRelayMessg evRelayMessg;
    #uses interface Server server;
    #reads data blfUsers blfdatUsers;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAreceiveInteraction ev)
    {
        //We could check the interaction class handle, but there is only
        one
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        (HLA = (Server.HLachat)_HLA.as_object() != null);
    }
}
# reasoning method
body()
{
    HLAunicodeString _message = new HLAunicodeString();
    HLAunicodeString _sender = new HLAunicodeString();
    for (Iterator i = ev.theParameters.keySet().iterator();
        i.hasNext(); )
    {
        ParameterHandle _parameterHandle = (ParameterHandle)i.next();
        if (_parameterHandle.equals(HLA._iph_Communication_message))
        {
            _message.decode((byte[]) ev.theParameters.get(_parameterHandle));
            if (_parameterHandle.equals(HLA._iph_Communication_sender))
            {
                _sender.decode((byte[]) ev.theParameters.get(_parameterHandle));
                server.postEvent( evMessageUsr.message(_sender.toString().substring(1), _message.toString(), HLA.username ) );
            }
        }
        //for
        // server.postEvent( evMessageUsr.message( _sender.toString().substring(1), _message.toString(), HLA.username ) );
        logical String location = hlfDatUsers.get( location, ev.identifier ); //HLA.username
        @send( location.as_string(), evRelayMsg.relay( _message.toString(), _sender.toString().substring(1) ) );
        System.out.println( "(relaying message to " + ev.identifier + ")" ); //HLA.username
    }
    //end plnHLA_Interaction
The plnHLA_InteractionScopeAdvisory_Off plan handles the interaction scope advisory in the “turn off” case; this simply means the Client is “alone” in the Chat federation.

```java
// File: plnHLA_InteractionScopeAdvisory_Off.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.ack.evtHLAturnInteractionsOff;
import ca.gc.drdc_rddc.hla.rti1516.ack.blfHLA;

/**
 * Handles the HLA interactionScopeAdvisory event.
 */
public plan plnHLA_InteractionScopeAdvisory_Off
extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAturnInteractionsOff ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAturnInteractionsOff ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
          ( (HLA = (Server.HLAchat)_HLA.as_object()) != null);
        // No need to test (InteractionClassHandle)ev.theHandle
        // since we only subscribe to one interaction
    }

    #reasoning method
    body()
    {
        HLA._alone = true;
    }
}
// end plnHLA_InteractionScopeAdvisory_Off
```
The `plnHLA_InteractionScopeAdvisory_On` plan handles the interaction scope advisory in the “turn on” case; this simply means the Client is no longer “alone” in the Chat federation.

```java
// File: plnHLA_InteractionScopeAdvisory_On.java
package server;

import ca.gc.drdc_rddc.hla.rti1516.jack.blfHLA;
import ca.gc.drdc_rddc.hla.rti1516.jack.evtHLAturnInteractionsOn;

/**<*
Handles the HLAInteractionScopeAdvisory event.
*/
public plan plnHLA_InteractionScopeAdvisory_On
    extends Plan
{
    logical Object _HLA;
    Server.HLAchat HLA;
    #handles event evtHLAturnInteractionsOn ev;
    #uses interface Server server;
    #reads data blfHLA datHLA;

    static boolean relevant(evtHLAturnInteractionsOn ev)
    {
        return true;
    }

    context()
    {
        datHLA.get( ev.identifier, _HLA ) &&
        ( (HLA = (Server.HLAchat)_HLA.as_object()) != null);
        //No need to test (InteractionClassHandle)ev.theHandle
        //since we only subscribe to one interaction
    }

    #reasoning method
    body()
    {
        HLA._alone = false;
    }
}
//end plnHLA_InteractionScopeAdvisory_On
```
This page intentionally left blank.
References


[6] Jaillet, Christophe; Krajecki, Michaël; and Fugère, Dr. Benoît Jean (2002). *Parallélisation en mémoire partagée d'un moteur de simulation du commandement militaire basé sur une modélisation par automates cellulaires*, RenPar 14 (Rencontres francophones du parallélisme), Hammamet (Tunisia), April 2002


[40] Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve; Yergeau, François; Cowan, John; editors, Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation 16 August 2006, World Wide Web Consortium (W3C) http://www.w3.org/TR/2006/REC-xml11-20060816/ (accessed 2007 Jan 22)
Bibliography

Distributed Interactive Simulation (DIS) Protocol:


IEEE 1278 Distributed Interactive Simulation (DIS) Protocol:


IEEE (1997). IEEE 1278.4-1997 - Recommended Practice for Distributed Interactive Simulation - Verification Validation & Accreditation

IEEE (unpublished). IEEE 1278.5-XXXX - Fidelity Description Requirements (never published)

Aggregate Level Simulation Protocol (ALSP):


High Level Architecture (HLA) 1.3:


HLA (2001c). HLA 1.3NG Appendix C - Classes and Supporting Types (2001 Jun 21)


High Level Architecture (HLA) IEEE Proposal (P1516):


IEEE 1516 High Level Architecture (HLA):


JACK™ Intelligent Agents:

Other Documents:

http://www.cc.gate.edu/computing/papers/PAPERS/HLA-TM-1.0.pdf


## List of symbols/abbreviations/acronyms/initialisms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Three-Dimensional</td>
</tr>
<tr>
<td>ALSP</td>
<td>Aggregate Level Simulation Protocol</td>
</tr>
<tr>
<td>AMLCD</td>
<td>Active Matrix Liquid Crystal Display</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute <a href="http://www.ansi.org">www.ansi.org</a></td>
</tr>
<tr>
<td>AOS</td>
<td>Agent-Oriented Software Pty. Ltd. <a href="http://www.agent-software.com">www.agent-software.com</a></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ASIP</td>
<td>All-Source Intelligence Prototype/Producer</td>
</tr>
<tr>
<td>BDI</td>
<td>Belief Desire Intention</td>
</tr>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>C3IS</td>
<td>Command and Control Information System</td>
</tr>
<tr>
<td>CCIS</td>
<td>see C3IS</td>
</tr>
<tr>
<td>CD</td>
<td>Compact Disc</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>Compact Disc – Read-Only Memory</td>
</tr>
<tr>
<td>CGF</td>
<td>Computer-Generated Forces</td>
</tr>
<tr>
<td>CLI</td>
<td>Command-Line user-Interface</td>
</tr>
<tr>
<td>CMF</td>
<td>Collaborative Metaprogramming Framework</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture <a href="http://www.corba.org">www.corba.org</a></td>
</tr>
<tr>
<td>DDM</td>
<td>Data Distribution Management</td>
</tr>
<tr>
<td>DIF</td>
<td>Data Interchange Format</td>
</tr>
<tr>
<td>DIR</td>
<td>Defence Industrial Research</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Interactive Simulation</td>
</tr>
<tr>
<td>DISTI</td>
<td>Distributed Simulation Technology Inc. <a href="http://www.simulation.com">www.simulation.com</a></td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>DND</td>
<td>Department of National Defence <a href="http://www.dnd.ca">www.dnd.ca</a></td>
</tr>
<tr>
<td>DRDC</td>
<td>Defence R&amp;D Canada <a href="http://www.drde-rrdc.gc.ca">www.drde-rrdc.gc.ca</a></td>
</tr>
<tr>
<td>DRP</td>
<td>Document Review Panel</td>
</tr>
<tr>
<td>DSP</td>
<td>Defense Standardization Program <a href="http://www.dsp.dla.mil">www.dsp.dla.mil</a></td>
</tr>
<tr>
<td>DSS</td>
<td>Decision Support Systems</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>DUB</td>
<td>Dimension Upper Bound</td>
</tr>
<tr>
<td>FDD</td>
<td>FOM Document Data</td>
</tr>
<tr>
<td>FED</td>
<td>Field Emission Display</td>
</tr>
<tr>
<td>FEDEP</td>
<td>Federation Development Process</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In, First-Out</td>
</tr>
<tr>
<td>FOM</td>
<td>Federation Object Model</td>
</tr>
<tr>
<td>GALT</td>
<td>Greatest Available Logical Time</td>
</tr>
<tr>
<td>GEO-TIDE</td>
<td>Geospatial Technologies for Information Decisions</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User-Interface</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>HPCVL</td>
<td>High Performance Computing Virtual Laboratory</td>
</tr>
<tr>
<td>ICM</td>
<td>International Congress of Mathematicians</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
</tbody>
</table>
IEC  International Electrotechnical Commission  www.iec.ch
IEEE  Institute of Electrical and Electronics Engineers  www.ieee.org
INCITS  InterNational Committee for Information Technology Standards  www.incits.org
IRC  Internet Relay Chat
ISBN  International Standard Book Number
ISO  International Organization for Standardization  www.iso.ch
IST  (University of Central Florida) Institute for Simulation and Training  www.ist.ucf.edu
JACK  Originally stood for Java Agent Compiler and Kernel
JAL  JACK Agent Language
JDE  JACK Development Environment
JNI  Java Native Interface
JTC  Joint Training Confederation
LFCS  Land Forces Command System
LITS  Least Incoming Time Stamp
LRC  Local RTI Component
M&S  Modelling and Simulation
MAGNETAR  Metaprogrammable AGent NETwork ARchitecture  www.magnetar.org
MOM  Management Object Model
NATO  North Atlantic Treaty Organization  www.nato.int
OLED  Organic Light-Emitting Diode
OMG  Object Management Group  www.omg.org
OMT  Object Model Template
OO  Object-Oriented
OOAD  Object-Oriented Analysis and Design
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOTW</td>
<td>Operations Other Than War</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>PDG</td>
<td>Product Development Group</td>
</tr>
<tr>
<td>R&amp;D</td>
<td>Research and Development</td>
</tr>
<tr>
<td>RMC</td>
<td>Royal Military College <a href="http://www.rmc.ca">www.rmc.ca</a></td>
</tr>
<tr>
<td>RPR-FOM</td>
<td>Realtime Platform Reference FOM</td>
</tr>
<tr>
<td>RTI</td>
<td>Run-Time Infrastructure</td>
</tr>
<tr>
<td>SIMNET</td>
<td>SIMulator NETwork, <em>later became</em> SIMulation NETwork</td>
</tr>
<tr>
<td>SIREQ</td>
<td>Soldier Information Requirements</td>
</tr>
<tr>
<td>SISO</td>
<td>Simulation Interoperability Standards Organization <a href="http://www.sisostds.org">www.sisostds.org</a></td>
</tr>
<tr>
<td>SO</td>
<td>Shared Object</td>
</tr>
<tr>
<td>SOM</td>
<td>Simulation Object Model</td>
</tr>
<tr>
<td>SPIE</td>
<td>Society for Photo-Optical Instrumentation Engineers <a href="http://www.spie.org">www.spie.org</a></td>
</tr>
<tr>
<td>STANAG</td>
<td>Standardisation Agreement <a href="http://www.cri.ensmp.fr/OTAN/Serveur2.html">www.cri.ensmp.fr/OTAN/Serveur2.html</a></td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>TD</td>
<td>Technology Demonstration</td>
</tr>
<tr>
<td>TFT-LCD</td>
<td>Thin Film Transistor – Liquid Crystal Display</td>
</tr>
<tr>
<td>ToMaDi</td>
<td>Topographic Map Display</td>
</tr>
<tr>
<td>TSO</td>
<td>Time-Stamp Order</td>
</tr>
<tr>
<td>UCF</td>
<td>University of Central Florida <a href="http://www.ucf.edu">www.ucf.edu</a></td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>USD (A&amp;T)</td>
<td>U.S. Under Secretary of Defense for Acquisition and Technology <a href="http://www.acq.osd.mil">www.acq.osd.mil</a></td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium <a href="http://www.w3.org">www.w3.org</a></td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language <a href="http://www.xml.org">www.xml.org</a></td>
</tr>
</tbody>
</table>
XRTI  eXtensible Run-Time Infrastructure  www.npsnet.org/~npsnet/xrti
Distribution list

Internal distribution (DRDC Valcartier)

2 Thibault, D. U. (1 CD and 1 printed)

4 Library (1 CD and 3 printed)

Force Protection & Weapon Systems:

1 Electro-optical Warfare (EOW; Nathalie Harrison) (1 CD)

C4ISR Sector:

1 C² Decision Support Systems (C²DSS) (1 CD)

1 Intelligence and Information (I) (1 CD)

1 System of Systems (SoS) (1 CD)

10 TOTAL LIST PART I (6 CDs and 4 printed)

External distribution

1 DAD (LColls Hall & McPhearson) (1 CD)
Directorate of Army Doctrine
Canadian Forces Base Kingston
P.O. Box 17000 Stn Forces
Kingston ON K7K 7B4

1 DLR 5 (LCol Bodner) (1 CD)
Directorate of Land Requirements
Canadian Forces Base Kingston
P.O. Box 17000 Stn Forces
Kingston ON K7K 7B4

1 DLR 8 (LCol Lefebvre) (1 CD)
Directorate Land Requirements
National Defence Headquarters
MGen George R. Pearkes Building
101 Colonel By Drive
Ottawa ON K1A 0K2

1 DRDKIM (1 CD)
1 DLSC (LCols Maurer & Williams) (1 CD)
Directorate of Land Strategic Concepts
Canadian Forces Base Kingston
P.O. Box 17000 Stn Forces
Kingston ON K7K 7B4

1 DLSE/ASECO (Maj James S. Denford & Mr. Paul Roman) (1 CD)
Directorate Land Synthetic Environment
Army Synthetic Environment Coordination Office
Land Force Doctrine and Training System
P.O. Box 17000 Station Forces
Canadian Forces Base Kingston
Kingston, ON K7K 7B4

1 DND SECO (Mr. D. Robert Elliott) (1 CD)
Canadian Forces Experiment Centre
Synthetic Environment Coordination Office
DREO/CRC Shirley’s Bay Campus
3701 Carling Avenue
Ottawa, ON K1A 0K2

1 DRDC Atlantic - VCS Group (Mr. Brad Dillman) (1 CD)
Defence R&D Canada – Atlantic
9 Grove Street
Dartmouth, NS B2Y 3Z7

1 DRDC Ottawa FFSE (Dr. Andrew L. Vallerand & Dr. Paul Pace) (1 CD)
Defence R&D Canada – Ottawa
Future Forces Synthetic Environments
3701 Carling Avenue
Ottawa, Ontario, K1A 0Z4

1 DSTL 8 (Mr. Benoit Cantin, Thrust Co-ordinator 12o “Command”) (1 CD)
Directorate Science and Technology Land
National Defence Headquarters
MGen George R. Pearkes Building
101 Colonel By Drive
Ottawa ON K1A 0K2

1 LSEC (Capt Chris Taft) (1 CD)
Land Software Engineering Centre
National Defence Headquarters
MGen George R. Pearkes Building
101 Colonel By Drive
Ottawa ON K1A 0K2

11 TOTAL LIST PART 2 (11 CDs)

21 TOTAL COPIES REQUIRED
**DOCUMENT CONTROL DATA**

1. **ORIGINATOR** (name and address)
   DRDC Valcartier
   2459 boul. Pie-XI nord
   Quebec QC G3J 1X5

2. **SECURITY CLASSIFICATION**
   (Including special warning terms if applicable)
   UNCLASSIFIED

3. **TITLE** (Its classification should be indicated by the appropriate abbreviation (S, C, R or U))
   (U) Simulation technologies for CAS development & training - Final report

4. **AUTHORS** (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)
   Thibault, Daniel U.

5. **DATE OF PUBLICATION** (month and year)
   February 2008

6a. **NO. OF PAGES**
   1380

6b. **NO. OF REFERENCES**
   40

7. **DESCRIPTIVE NOTES** (the category of the document, e.g. technical report, technical note or memorandum. Give the inclusive dates when a specific reporting period is covered.)
   Project final report

8. **SPONSORING ACTIVITY** (name and address)
   DLSC (Directorate of Land Strategic Concepts)
   Box 17000 Stn Forces
   Kingston ON K7K 7B4

9a. **PROJECT OR GRANT NO.** (Please specify whether project or grant)
   12sd (12kr)

9b. **CONTRACT NO.**

10a. **ORIGINATOR'S DOCUMENT NUMBER**
    DRDC Valcartier TR 2007-412

10b. **OTHER DOCUMENT NOS**
    N/A

11. **DOCUMENT AVAILABILITY** (any limitations on further dissemination of the document, other than those imposed by security classification)

   - Unlimited distribution
   - Restricted to contractors in approved countries (specify)
   - Restricted to Canadian contractors (with need-to-know)
   - Restricted to Government (with need-to-know)
   - Restricted to Defense departments
   - Others

12. **DOCUMENT ANNOUNCEMENT** (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)
13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself.

   It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin
   with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified)
   represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is
   bilingual).

   A functional overview of the High Level Architecture (HLA) distributed simulation standard Institute of Electrical and Electronics
   Engineers (IEEE) 1516 is provided. The Agent Oriented Software (AOS) intelligent software agent development environment,
   JACK, is described. Problems with the IEEE 1516 specification are identified and possible improvements outlined. A
   demonstration Internet Relay Chat (IRC)-like application is designed and implemented, which allows a Java application to
   interoperate with a JACK application through the HLA. A re-usable JACK capability is described which allows any JACK agent to
   participate in an HLA federation. A re-usable framework of Java classes implementing the HLA Object Model Template (OMT)
   encoding and decoding facilities is described. Finally, bugs are documented in JACK and the HLA Run-Time Infrastructure (RTI)
   used, and various design considerations and lessons learned are discussed.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document
    and could be helpful in cataloguing the document. They should be selected so that no security classification is required.
    Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be
    included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific
    Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the
    classification of each should be indicated as with the title.)

   Simulation; Modelling; M&S; Distributed simulation; High Level Architecture; HLA; Pitch pRT1516; IEEE 1516; Object Model
   Template; OMT; Java; Agent Oriented Software; AOS; JACK; Intelligent Software Agents; Internet Relay Chat; IRC.