

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A SIMPLE SOFTWARE AGENTS FRAMEWORK FOR BUILDING DISTRIBUTED APPLICATIONS

by

Boon Kwang, Kin

March 2001

Thesis Advisor:
Co-Advisor:

Valdis Berzins
Jun Ge

Approved for public release; distribution is unlimited.

20010612 099

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) A Simple Software Agents Framework for Building Distributed Applications			5. FUNDING NUMBERS	
6. AUTHOR(S) Kin Boon Kwang			8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The development of distributed systems needs to consider multiple factors such as performance, scalability, resource sharing, and fault tolerance. This thesis proposes a simple agent-based framework to address these concerns when building distributed applications. Agents act as interfaces among processes to interact and to cooperate in a distributed environment. These agents encapsulate the implementation details and make the network transparent to running processes. The proposed framework is built on JINI infrastructure. It uses Linda TupleSpace model, a shared network-accessible repository, for different processes to exchange information. Processes are loosely coupled. Under the proposed model, the correspondent language wrappers such as Java, Ada, C++, C and Visual Basic support multiple programming languages. Information exchange among processes is not restricted to data only. Executable components, leveraging on Java code's portability features, can be sent over a heterogeneous environment and executed remotely. This framework can further address several important issues on formal specifications of the communication layer, such as partial failure, synchronization, coordination and heterogeneity, by offering properties in our design for operation timeout, and information and service leasing. This framework is to be used in the Distributed Computer Aided Prototyping System (DCAPS) to provide the inter-process communication layer. It simplifies the tasks of designing, binding and analyzing multiple processes of real-time, distributed prototype systems. The provided interface library shields the developer from working on the underlying dynamic and complex network environment. It supports a wide variety of programming languages and operating platforms. Important issues under distributed environment, such as partial failure, synchronization and coordination, have been taken into consideration.				
14. SUBJECT TERMS ActiveX, Agent, Distributed Systems, Framework, Interoperability, JavaSpace, JINI, Software, TupleSpace, Wrapper.			15. NUMBER OF PAGES	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

**A SIMPLE SOFTWARE AGENTS FRAMEWORK FOR BUILDING
DISTRIBUTED APPLICATIONS**

Boon Kwang, Kin
Ministry of Defence, Singapore
B.Eng.(Hons),
Nanyang Technological University, 1996

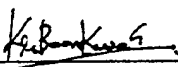
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

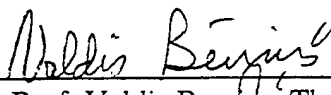
**NAVAL POSTGRADUATE SCHOOL
March 2001**

Author:



Boon Kwang, Kin

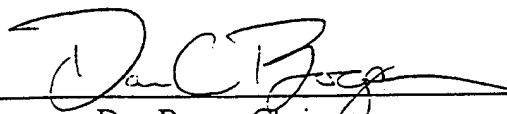
Approved by:



Prof. Valdis Berzins, Thesis Advisor



Dr. Jun Ge, Co-Advisor



Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The development of distributed systems needs to consider multiple factors such as performance, scalability, resource sharing, and fault tolerance. This thesis proposes a simple agent-based framework to address these concerns when building distributed applications. Agents act as interfaces among processes that interact and cooperate in a distributed environment. These agents encapsulate the implementation details and make the network transparent to running processes. The proposed framework is built on JINI infrastructure. It uses Linda TupleSpace model, a shared network-accessible repository, for different processes to exchange information. Processes are loosely coupled. They discover and linkup with one another by using services residing on JINI infrastructure. Under the proposed model, the correspondent language wrappers such as Java, Ada, C++, C and Visual Basic support multiple programming languages. Information exchange among processes is not restricted to data only. Executable components, leveraging on Java code's portability features, can be sent over a heterogeneous environment and executed remotely.

This framework can further address several important issues on formal specifications of the communication layer, such as partial failure, synchronization, coordination and heterogeneity, by offering properties in our design for operation timeout, and information and service leasing.

This framework is to be used in the Distributed Computer Aided Prototyping System (DCAPS) to provide the inter-process communication layer. It simplifies the

tasks of designing, binding and analyzing multiple processes of real-time, distributed prototype systems.

The provided interface library shields the developer from working on the underlying dynamic and complex network environment. It supports a wide variety of programming languages and operating platforms. Important issues under distributed environment, such as partial failure, synchronization and coordination, have been taken into consideration.

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
II.	BACKGROUND.....	5
A.	BENEFITS OF BUILDING DISTRIBUTED APPLICATIONS	5
1.	Performance.....	5
2.	Scalability.....	5
3.	Resource sharing	6
4.	Fault tolerance.....	6
B.	CHALLENGES OF BUILDING DISTRIBUTED APPLICATIONS	6
1.	Heterogeneity.....	7
2.	Latency	7
3.	Partial failure.....	7
4.	Synchronization.....	8
5.	Coordination	8
C.	MODELS FOR BUILDING DISTRIBUTED SYSTEMS.....	9
1.	Client/server model	9
a.	Sockets	10
b.	Remote Procedure Call.....	10
c.	Message Oriented Middleware (MOM).....	10
2.	Distributed Object Model.....	11
a.	Distributed Object standards	11
D.	JAVA LANGUAGE FOR DISTRIBUTED PROGRAMMING.....	12
E.	THE JINI TECHNOLOGIES.....	13
F.	TUPLE SPACE MODEL	14
III.	DESIGN.....	15
A.	OVERVIEW	15
B.	ENTRIES	19
1.	Shared variables	19
2.	Ordered structures.....	20
3.	Unordered structures.....	21
C.	AGENT SERVICE.....	22
D.	AGENT OPERATIONS	22
1.	Service registration operations	23
2.	Entry operations.....	24
3.	Transaction operations	25
4.	Event Handling.....	26
5.	Code Serialization	28
E.	AGENT ATTRIBUTES.....	28
1.	Leasing.....	28
2.	Timeout	30
F.	AGENT WRAPPERS	31
IV.	IMPLEMENTATION.....	33
A.	ARCHITECTURE	33

B.	AGENT SERVICE MODULE.....	34
C.	SERVICE REGISTRATION MODULE.....	34
1.	Service Accessor	35
2.	Service Finder	36
D.	EVENT MODULE	36
1.	Space Event Registration.....	36
2.	Space Event Listener.....	37
3.	Space Action handler	38
E.	INTERFACE MODULE	38
1.	Application Interface	39
a.	Agent configuration.....	39
b.	Entry declaration	40
c.	Transaction handling.....	41
2.	Entry Handlers	42
3.	Entries	45
F.	AGENT WRAPPER MODULES	46
V.	RESULTS.....	49
A.	SERVICE RESPONSE TIME	49
B.	DEVELOPMENT GUIDELINES	51
C.	TESTING JINI/SERVICES.....	52
D.	AN EXAMPLE: AN ELEVATOR CONTROL SYSTEM.....	55
1.	Declare and initialize entry.....	60
2.	Read and Write Entry.....	61
VI.	DISCUSSION	63
A.	JAVA PROGRAMMING LANGUAGE.....	63
B.	JINI TECHNOLOGY.....	64
C.	FUTURE WORKS	64
VII.	CONCLUSION.....	67
	LIST OF REFERENCES	69
	APPENDIX A. AGENT API.....	71
	TUPLESAPCE.CORE CLASS AGENT.....	71
	TUPLESAPCE.CORE CLASS SERVICEFINDER.....	74
	TUPLESAPCE.CORE CLASS SERVICEACCESSOR.....	75
	TUPLESAPCE.CORE INTERFACE SPACEEVENTREGISTRATION	76
	TUPLESAPCE.CORE INTERFACE SPACELISTENER.....	77
	TUPLESAPCE.CORE INTERFACE TSCONSTANTS	78
	TUPLESAPCE.CORE CLASS TSBASE.....	79
	TUPLESAPCE.CORE CLASS TSBOOLEAN	81
	TUPLESAPCE.CORE CLASS TSDOUBLE	84
	TUPLESAPCE.CORE CLASS TSHASH.....	87
	APPENDIX B. JINI/SERVICES SETUP SCRIPT	91

APPENDIX C. AGENT TEST BENCH LISTING	95
A. JAVA VERSION	95
1. JavaTestBench.java.....	95
2. PerformActions.java	109
B. VISUAL BASIC VERSION	121
1. VBTestBench.vb	121
C. C VERSION.....	127
1. CtestBench.C	127
APPENDIX D. AGENT WRAPPER LISTING	137
1. AGENT.H.....	137
2. AGENT.C.....	141
APPENDIX E. AGENT API LISTING.....	169
A. SERVICE PACKAGE	169
1. AgentServiceInterface.java	169
2. AgentService.java.....	169
B. CORE PACKAGE	173
1. Agent.java	173
2. ServiceFinder.java.....	185
3. ServiceAccessor.java	187
4. SpaceEventRegistration.java	189
5. TSBase.java.....	190
6. TSConstants.....	192
7. TSDouble.java	199
8. TSLong.java.....	205
9. TSHash.java.....	211
10. TSQueue.java.....	220
11. TSString.java.....	226
C. ENTRIES PACKAGE	232
1. EntryBoolean.java.....	232
2. EntryBytes.java	232
3. EntryClass.java	233
4. EntryDouble.java	233
5. EntryFloat.java.....	233
6. EntryHash.java.....	234
7. EntryInteger.java	234
8. EntryLong.java.....	235
9. EntryListItem.java	235
10. EntryListStatus.java	236
11. EntryQueueItem.java	237
12. EntryQueueStatus.java.....	237
12. EntryStackItem.java	239
14. EntryStackStatus.java	239
15. EntryString.java.....	240
16. SpaceActionHandle.java.....	241
17. SpaceEventListener.java	241

INITIAL DISTRIBUTION LIST	243
---------------------------------	-----

LIST OF FIGURES

Figure 1 A distributed application using our agent framework	15
Figure 2 Share Variables Entries.....	20
Figure 3 Ordered Structure Entries	21
Figure 4 Unordered Structure Entries	22
Figure 5 Agent Service Registration Process.....	23
Figure 6a Entry Read Operation.....	24
Figure 6b Entry Take Operation.....	24
Figure 6c Entry Write Operation.....	25
Figure 6d Entry Update Operation	25
Figure 6e Entry Notify Operation	25
Figure 7 Entry Operations with Transaction.....	26
Figure 8 Event Handling	27
Figure 9 Code Serialization.....	28
Figure 10 Renewing Service Lease.....	29
Figure 11 Entry Lease Expired.....	29
Figure 12 Transaction Lease Expired.....	30
Figure 13 Service Timeout.....	31
Figure 14 Entry Read/Take timeout	31
Figure 15 Agent Wrappers	32
Figure 16 Agent Architecture.....	33
Figure 17 Service Registration Module	35
Figure 18 Event Module.....	36
Figure 19 Agent Interface Module	39
Figure 20 Entry Handlers	43
Figure 21 Entries	46
Figure 22 Agent Wrapper Module	46
Figure 23 Chart 1 JINI/JavaSpace Response Time.....	50
Figure 24 Chart 2 JINI/JavaSpace Response Time.....	50
Figure 25a Agent Test Bench (Java Language version)	53
Figure 25b Agent Test Bench (Visual Basic Language version)	54
Figure 25c Agent Test Bench (C Language version).....	54
Figure 26 ECS State Chart	57
Figure 27 ECS processes.....	58
Figure 28A Elevator A & B Control Panel	59
Figure 28B Floor Control Panel	59

LIST OF TABLES

Table 1 ECS Requirements	56
Table 2 ECS Shared Entries	59

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My sincerest appreciation and thanks to my thesis advisor, Professor Valdis Berzins. His knowledge of this field is extraordinary and he provided me with many insight ideas, assistance, and support.

I would also like to give special thanks to Doctor Jun Ge for enlightening me and working diligent hours to help me in the thesis.

- Boon Kwang Kin

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In the past few years, the computing landscape has changed dramatically. More and more devices, such as hand-phones, Personal Device Assistances (PDAs) and Internet terminals, etc., have been enhanced with network capabilities to leverage on the benefits that new communication technologies have brought to us. Industrial companies and military services often operate and communicate via the Internet in a broad area to streamline operations and cut down on expenditure. Devices and software components have become more tightly coupled, cooperating together in a distributed system to accomplish a common goal. A distributed computer system is, therefore, defined to be a system of multiple autonomous processing elements, cooperating for a common purpose or to achieve a common goal ([Robert98]). Hence, challenges and concerns for constructing a distributed computer system have become a growing field that is being intensively studied worldwide in recent years.

Distributed systems offer more benefits than standalone systems do. The computational ability of a standalone system is usually limited by the available power and resources in a single computer. Distributed applications can grow more easily to meet new demands by introducing more CPUs and memory. Expensive resources such as supercomputers and color printers can be shared and utilized by many different users. Distributed systems could reach a better fault tolerance and availability because both problems have to be carefully taken care of from the beginning of system design. Hence, unlike a standalone program, which could be totally terminated if any component in its system fails, a distributed application can still continue to operate even if some processes

get disrupted because of system failure. Distributed systems can also reuse a lot of legacy software components through re-engineering.

Rewriting legacy software to run in a distributed environment tends to be prohibitively expensive and complexed. Many legacy software systems are expensive investments that have been developed over many years. Replacing them with new designs or implementations is usually not easy to justify in terms of costs and resource allocation. Although the only way to keep such legacy software useful is to incorporate them into a wider cooperating community in which they can be exploited by other pieces of software, in practice, this could be very complex in terms of design.

Despite the benefits, building distributed applications is difficult. There are many different requirements and specifications for distributed systems. Developers have to face many problems that do not arise when building standalone applications. Besides some common problems like heterogeneity, latency, partial failure and synchronization, developers have to consider issues about how to make legacy software components “co-exist” with each other and how to “glue” multiple processes together running independently on different machines.

Today, the technique to “glue” multiple processes running in a heterogeneous environment ranges from low-level sockets and messaging techniques to more sophisticated technology Object Resource Broker (ORB) [ORB91], such as CORBA and DCOM [DCOM96]. Many of these techniques require developers either to perform significant low-level coding work in constructing the communication mechanism or to

have a good knowledge of the interface details before designing. Hence, “gluing” pieces of processes is still a difficult task and requires skilful designers with expertise.

Wrapper and Glue technology was proposed for constructing distributed systems. Wrapping is an approach to protect legacy software systems and commercial off-the-shelf (COTS) software products that requires no modification of those products. A wrapper consists of two parts, an adapter that provides some additional functionalities for an application program at key external interfaces, and an encapsulation mechanism that binds the adapter to the application and protects the combined components. Wrapping should require no changes to the existing application program. [MEESON97] Candidate mechanisms for implementation via wrappers: authentication, logging and auditing, constraint checking, encryption, access control, fault detection and recovery, redundancy. The glue plays another role by providing a uniformed, supporting layer for individual wrapped components based on their specific hardware and software configurations so that distributed systems are built upon the glue structure through which components communicate to each other. The developer does not have to go down to low-level communication details when developing the overall paradigm.

This thesis proposes a simple framework using agents to act as interfaces among various processes that interact and cooperate in a distributed heterogeneous environment. It shields developers from the underlying dynamic and complex network environment, and offers developers a simple set of Application Program Interfaces (API) to build distributed applications. Developers, therefore, do not need to worry about their operating platforms and programming languages. It also provides an easier way for legacy software to share information with other applications in a heterogeneous

environment. The glue structure provided in this thesis not only gives APIs to realize the communication function across the platform, but also explores the real-time constraints and monitors for distributed system communications.

This thesis is structured as follows. Chapter II introduces the benefits and challenges of distributed computing in detail and presents our survey of current technologies that are being used for constructing distributed applications. Chapter III gives an overview of the proposed agent framework. The features and underlying design details of the agent framework are presented with respect to the requirements of real-time distributed systems. Chapter IV discusses the agent architecture and its implementation. In Chapter V, a test-bed application implemented using various language wrappers is presented to demonstrate the agent's features. A more complicated application of an elevator control system has been constructed on the proposed agent structure. Future research directions are discussed in Chapter VI.

II. BACKGROUND

This session discusses the benefits and challenges of building distributed applications. It also presents a list of technologies that are useful for designing and creating distributed systems.

A. BENEFITS OF BUILDING DISTRIBUTED APPLICATIONS

Distributed applications offer more benefits than standalone applications do in terms of performance, scalability, resource sharing, fault tolerance and availability.

1. Performance

Distributed applications can achieve better performance than standalone applications. Unlike standalone applications, which rely heavily on newer hardware to improve performance, distributed applications utilize the combination of systems connected together by a network to boost performance. This performance gain is often achieved through splitting problems into smaller pieces and then delivering them across the network, where they can be executed in parallel. Nevertheless, this category of problems is restricted to tasks that have a low communication-to-computation ratio, i.e. processes that spend much less time in communicating than in computing. Otherwise, any performance gain would be overwhelmed by the network latency.

2. Scalability

Distributed applications can scale more easily. Unlike standalone applications, where their scalabilities are limited by how much resource each system has, distributed

applications can grow without such constraint and new resources can be added to match the problem's difficulties.

3. Resource sharing

Distributed applications allow organizations to fully utilize their available physical resources. For instance, expensive resources like supercomputers and color printers are difficult to redistribute, allowing each end user a local access might not be feasible. However, by making them as services more users can remotely access them and resource can be better utilized.

4. Fault tolerance

Non-distributed systems typically have little tolerance for failure. If a system that hosts standalone applications fails, these applications will terminate and remain unavailable until they are restarted. On the contrary, distributed applications can tolerate a limited amount of failure since they are built on multiple, independent processes that reside on many systems -- if some processes fail, others can continue.

B. CHALLENGES OF BUILDING DISTRIBUTED APPLICATIONS

Despite their benefits, it is difficult to build distributed applications. The distributed environment introduces many problems that have not been taken care of in a standalone application's development. These problems include heterogeneity, latency, partial failure, synchronization and coordination.

1. Heterogeneity

Typically, a distributed application may have to reside on different platforms and use multiple technologies from various vendors to meet all requirements. Mixing them together usually creates new dimensions of difficulties in terms of integration and system management. Integrating multiple technologies can often result in incompatibilities that are beyond the control of designers.

2. Latency

Communication latency among processes over network is longer and less predictable than in a local system. The time lag is typically in several orders of magnitude compared to the communication among processes in the same system or to the speed of processors. Unlike a local system where user has better control over how resources are utilized, the user in a distributed system usually has no control on some resources, such as network usages. Hence, predicting the communication latency among processes that belong to a distributed application is much more complex and difficult.

3. Partial failure

Partial failure is the greatest challenge for designers building distributed systems. Unlike a standalone system, which is subjected to total failure, a distributed system is subjected to partial failure. If any of the components in the standalone system fails, the entire computation terminates. This type of failure is easy to detect and correct. Correction can be done simply by rebooting the system and follow by restarting the application. On the other hand, partial failure, if a component in the distributed system fails, some other processes of a distributed application can continues to run; very often

this is difficult to detect and control because resources like network are also being shared by many other applications.

4. Synchronization

In a distributed application, processes need to synchronize with one another to succeed a common task. For example, a distributed application often needs to mediate access to a limited set of shared resources, guarantee fair access to those resources, or prevent processes from terminating as they wait for resources that may never become available. Synchronization is not a problem for standalone applications because most of them are executed in sequence, and has a central resource manager, the operating system, which mediates access to critical resources. Designers building distributed applications have to build their own synchronization mechanism, which is difficult because processes are run independently on many machines that might have different architectures -- if not done properly, it can easily result in a deadlock situation.

5. Coordination

Coordinating distributed processes can be difficult. An application that runs on a single machine has an operating system to act as a centralized manager to synchronize multiple threads. But in a network environment, a single point of control does not necessarily exist. Processes are executed on different machines and at their own pace. Unless we build a centralized controller to manage those processes, which might introduce an unwelcome bottleneck to the environment, we must build a distributed means of managing their interactions.

C. MODELS FOR BUILDING DISTRIBUTED SYSTEMS

Distributed systems can be implemented using these models; namely, client/server model and distributed object model.

1. Client/server model

The Client/server model contains a set of server processes and client processes. A server process act as a resource manger for a collection of resources of a particular type such as database server, file server, print server and etc. A client process communicates with the server for the purpose of exchanging or retrieving information. Communication between the client and server can be achieved through sets of protocols agreed by both parties.

The major drawback of the client/server model is that the control of individual resource is centralized at the server and this could create a potential bottleneck and a single point of failure. Although many implementations have tried to overcome this drawback by replicating storage data and functions across multiple servers, thus making duplicate server to either act as backup or serve different cluster of clients, this has introduced new problems in terms of maintaining data consistency in the servers.

Despite the drawback, centralizing of resources at a few locations greatly simplifies the management of resources. Software update and maintenance are is much simple; administrators just need to concentrate on a few locations.

The client/server model can be implemented in various ways. Typically, it is done using low-level sockets, remote procedure calls or high-level message oriented middleware such as message queues.

a. *Sockets*

Sockets are low-level inter-process communication similar to file input/output. It requires developers to implement their own protocol through which the client and server will use to communicate with each other.

b. *Remote Procedure Call*

A Remote Procedure Call (RPC) is a high-level communication paradigm that allows network applications to be developed by way of specialized procedure calls. Client invokes a procedure call that sends requests to a remote server. When these requests arrive, the server calls a dispatch routine, executes the requested procedures, and returns the results to the client. Program control is returned to the client immediately after the RPC is completed.

The major limitation of RPC is that it only offers synchronous data exchange between the calling program and called procedure. Developers must employ operating system features such as threads or subtasks to force the RPC to process in an asynchronous manner. Using RPCs to integrate applications also limits portability because the application code will become very dependent on the operating system.

c. *Message Oriented Middleware (MOM)*

MOM is primarily middleware that facilitates communication between distributed applications. While MOM supports both synchronous and asynchronous messaging, it is most closely identified with asynchronous messaging using queuing. MOM sends messages from one application to another using a queue as an interim step. Client messages are sent to a queue and remain there until they are retrieved by the server application.

The advantage of this system is that the server application does not need to be available when the message is sent. Instead, the server can retrieve the message at any time. In addition, since messages can be retrieved off the queue in any order, MOM can also facilitate retrieval of messages using priority or load-balancing schemes. MOM can also provide a level of fault-tolerance using persistent queues that allow messages to be recovered when the system fails.

2. Distributed Object Model

A distributed object-based system is a collection of objects that isolates requestors of services from providers of services (servers) by a well-defined encapsulating interface. Clients are isolated from the implementation of services as data representations and executable code.

In a distributed object model, a client sends a message to an object that in turn interprets the message to decide what service to perform. This service could be performed either through the object or a broker.

a. Distributed Object standards

Distributed object systems such as CORBA, DCOM, and Java RMI provide the infrastructure for supporting remote object activation and remote method invocation in a client-transparent way. A client program obtains a pointer (or a reference) from a remote object, and invokes methods through that pointer as if the object resides in the client's own address space. The infrastructure takes care of all low-level issues such as packing the data in a standard format for heterogeneous environments (i.e., marshaling

and unmarshaling), maintaining the communication endpoints for message sending and receiving, and dispatching each method invocation to the target object.

Among all different vendors for distributed object systems, CORBA is the most widely supported standard. Its main advantages are platform independence and open industry standard that contains over 750 industry members.

D. JAVA LANGUAGE FOR DISTRIBUTED PROGRAMMING

The major advantages of using Java programming language to write distributed applications are their platform independent and networking support. Unlike applications written in platform dependent programming languages such as C++ and Ada languages, which must be compiled to their native platforms in order to run; Java applications need not be compiled and they can run on varieties of system architectures and software platforms as long as they have JVMs (Java virtual machines) [JVM96] installed to interpret the byte code during runtime. Java language is also the first general programming language that is designed specifically to work over the network, in particular the Internet. Java extensive library of routines for coping with TCP/IP protocols like HTTP and FTP make creating network connections much easier than in C or C++. Java applications can open and access objects across the net via URLs with the same ease that programmers are used to when accessing a local file system.

E. THE JINI TECHNOLOGIES

JINI [KEI99] [JOY99] is one of a large number of distributed systems architectures including industry-pervasive system such as CORBA and DCOM. It is distinguished by being based on Java programming language and deriving many features that leverage on capabilities that this language provides, like object-oriented programming, code portability, RMI (Remote Method Invocation) [RMI00], network support and security.

Some of the features Jini Technologies offers are

- ✓ Enable users to share services and resources over a network
- ✓ Provide users easy access to resources anywhere on the network while allowing the network location of the user to change
- ✓ Simplify the task of building, maintaining, and altering a network of devices, software, and users

Jini technology consists of a programming model and a runtime infrastructure. The programming model helps designers build reliable distributed systems as a federation of services and client programs. The runtime infrastructure resides on the network and provides mechanisms for adding, subtracting, locating, and accessing services as the system is used. Services use the runtime infrastructure to make themselves available when they join the network. A client uses the runtime infrastructure to locate and contact desired services. Once the services have been contacted, the client can use the programming model to enlist the help of the services in achieving the client's goals.

F. TUPLE SPACE MODEL

Tuple Space model was first conceived in the mid-1980 at Yale University by professor David Gelernter under a project called Linda. Tuples are typed data structures. Collections of tuple exist in a shared repository called a tuple space. Coordination is achieved through communication taking place in a tuple space globally shared among several processes. Each process can access the tuple space by inserting, reading or withdrawing tuples.

In this model, the programmer never has to be concerned with or program explicit message passing constructs and never has to manage the relatively rigid, point-to-point process topology induced by message passing. In contrast, coordination in Linda is *uncoupled* and *anonymous*. The first means that the acts of sending (producing) and receiving (consuming) data are independent (akin to buffered message passing). The second means that process identities are unimportant and, in particular, there is no need to "hard wire" them into the code.

There are a few similar implementations like JINI/JavaSpaces, IBM's Tspaces [TSPACES00] and Cloudscape's Java database [CLOUD00], which are built based on TupleSpace model. All of them offer a simple mechanism for dynamic communication, coordination, and sharing of objects between clients and servers.

III. DESIGN

This session gives an overview of the agent framework and describes its features and underlying design.

A. OVERVIEW

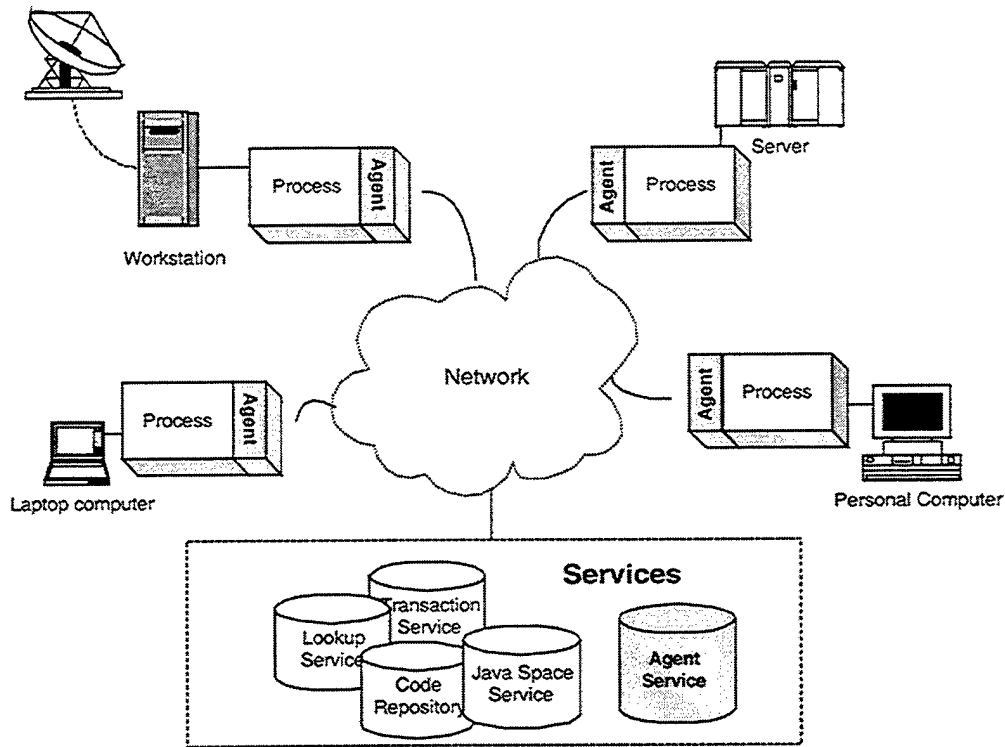


Figure 1 A distributed application using our agent framework

This thesis proposes a simple framework using agents to act as interfaces among processes interacting and cooperating in a distributed environment. These agents encapsulate the implementation details and make the network transparent to processes. Hence, by using these agents, developers can reduce significantly their efforts in implementing the process interfaces and concentrate on building the logic portions of a distributed application.

Our framework is built on JINI infrastructure and uses JINI network technology to simplify the task of building and maintaining reliable distributed systems. This technology consists of a well-defined programming model, which allows us to create our own services and leverage on services that have already been built to support JINI infrastructure. Using this programming model, we do not have to worry about the low-level communication protocol. Client processes can dynamically locate and access services held in the JINI community using its runtime infrastructure, even if they do not know their host URL addresses.

Our framework uses Linda [GEL85] TupleSpace model type of communication mechanism for inter-process communications. Processes are loosely coupled, rather than through direct communication, they interact in a globally shared space - *repository service* provided by JavaSpace Service [JS99], through share variables - *entries*. Being loosely coupled, processes need not be physically connected all the time and do not have to worry about the point-to-point topology induced by message passing. Several processes residing on same machine or on different machines can access the repository simultaneously. They interact among themselves by means of reading, writing or consuming entries stored in the repository service.

Repository service is a shared, network-accessible depot for entries storage. It behaves like a lightweight relational database, where agents acting on behalf of their processes can store, retrieve and query entries stored in it. Unlike database where users construct Structured Query Language (SQL) statements to query records, agents use pre-

constructed templates defined in our framework to match entries stored in the repository and only entries that match exactly the data types and fields defined in the template are returned by the repository service.

Entries are a collection of values or objects placed in repository service by coordinating processes for information sharing. Before a process can start operating on an entry, it first has to declare the entry, which is identified by a unique name and an entry type, with its agent; just like variables declaration in programming techniques. Entry type varies from simple primitive type like integer, float, double, etc to more complex type like queue, stack, list, etc where entries are managed as group. Each entry, upon declaration, is assigned an entry handler to serve operations for accessing the repository service.

Entry handler is responsible for carrying out operations pertaining to a declared entry. There are many kinds of entry handlers. Each one is associated to an entry type and has methods designed specifically to handle a particular entry type. Methods that are common in all handlers are: read, write, take, update and notify. Processes mainly use them for manipulating entries stored in the repository service. Every entry handler consists of a set of attributes that determine how it carries out its operations. Many of them can be overwritten after entry declaration by processes to meet different application needs. For instance, an entry-leasing attribute, which determines the validity of the entry process stored in the repository, can be used in a real-time application to specify the deadline of information to prevent recipients from accessing obsolete information, which sometimes can be more damaging than not have any of them.

Establishing a session with agent service is done in two simple steps: firstly, locate the service and then perform a login registration. If the process knows the network location where agent service is held, the process can bypass the search procedures. Searching for services in JINI network is achieved using JINI's discovery protocol - agent inserts a package into the network and wait for lookup services to respond. A lookup service provides the facility for services to publish their services. Upon receiving the request package, lookup services respond by returning a list of service items. Each item describes its service properties and functions. The agent then searches through the list, comparing their service attributes with those of the agent service. After it has found a matching service, it will proceed to establish a connection follow by a service registration providing a valid login ID and a password to the agent service.

Below is a summary of features the framework provides,

- ✓ A simple and yet comprehensive interface that allow multiple processes to get connected and interact with one another in a distributed environment.
- ✓ Processes can be implemented in Java, Visual Basic, C/C++, or Ada; two agent wrappers are included, ActiveX wrapper and a C library wrapper.
- ✓ Processes are loosely coupled; they need not be physically connected all the time and do not have to worry about the point-to-point topology induced by message passing.

- ✓ Several processes residing on the same machine or the different machines can access the repository service and retrieve data simultaneously in a reliable manner.
- ✓ Agent Service provides authentication and controls mechanism to manage processes using its services.
- ✓ Avoid the needs to create and manage remote/virtual classes (e.g. stubs and skeletons in RMI and CORBA implementations)
- ✓ Provide callback mechanisms that invoke user-defined methods when conditions are met.
- ✓ Support transaction, which enforce consistency over a set of entry operations
- ✓ Support leasing, which prevent resources from growing out of bound.

B. ENTRIES

Entries are a collection of values or objects placed in the repository service by coordinating agents for information sharing. Every entry has a unique entry name and an entry type. Unique name differentiates an entry from those stored in the repository and entry type determines the contents placed inside the entry. We have grouped entries into three categories: shared variables, ordered structures and unordered structures.

1. Shared variables

Like variables in programming techniques, they provide storage for some values or objects; however, shared variables allow multiple processes operating in the virtual

space to access and modify them simultaneously in an atomic manner. Atomic operation prevents these entries from corruption caused by race conditions. Shared variable type can either be defined as Integer, Boolean, Float, Long, Double, String or User-defined. User-defined as the name implies allow developers to define their own data structure.

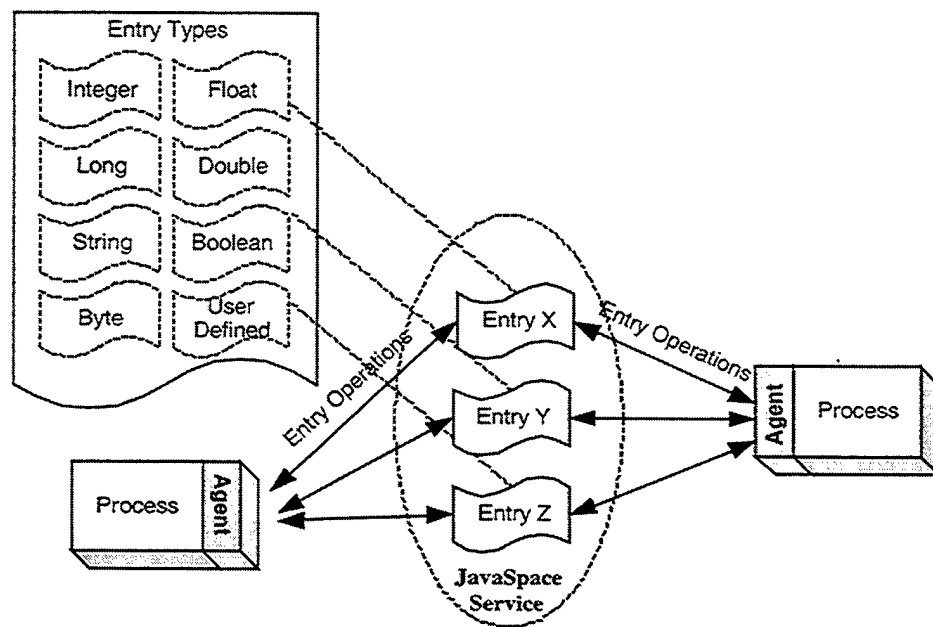


Figure 2 Share Variables Entries

2. Ordered structures

Ordered structure entries are like containers, collecting of similar values or objects, in object-oriented languages, where process can iterate through them. There are three types of ordered structures defined, i.e., queue, stack, and list. Queue type operates in First In First Out (FIFO); a *take* operation consumes the first entry in its group and a *put* operation deposits an entry to the last entry in its group. Stack type is similar to queue

type, except that the *take* operation consumes the most recent entry deposited. List type allows processes to search and iterate through the list of entries by using indexes.

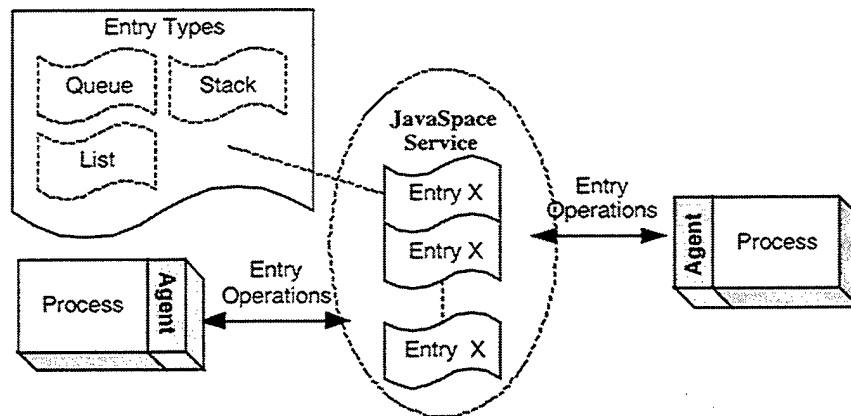


Figure 3 Ordered Structure Entries

3. Unordered structures

An unordered structure behaves like a pack of entries with the same entry ID but without any internal order. Unlike ordered structures where they are read or written in a predetermined order, unordered structures are arbitrary read or written into the repository. A typical example is task-result bags; a process places many small pieces of computational tasks in the task bag, waits for other processes to pick them up for computation and retrieves the results from the result bag placed after they have been computed.

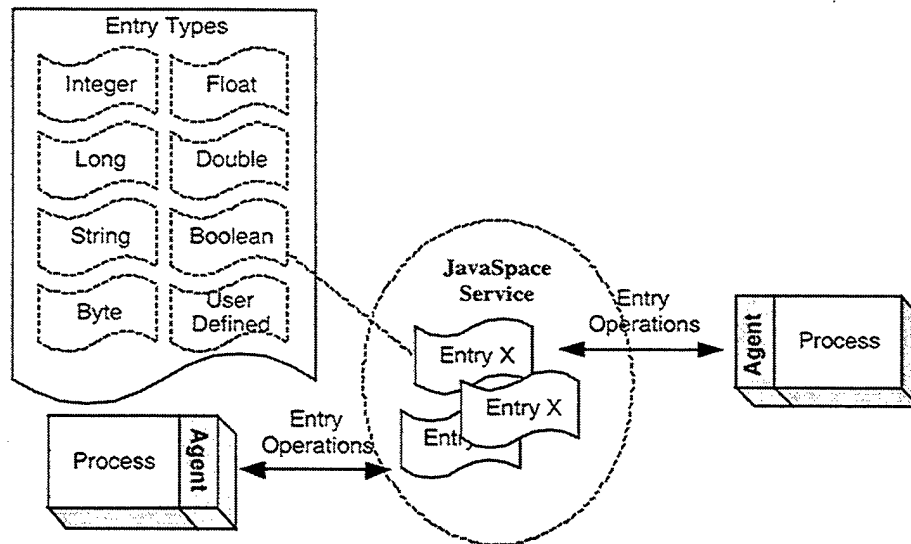


Figure 4 Unordered Structure Entries

C. AGENT SERVICE

Agent service provides authentication and controls mechanism for processes running in the network environment. It forces every process using its service to go through an authentication process, verifying their identifications against those held inside its service database. Identifications can be added or removed from its database to control process accesses to restricted resources. It includes a control mechanism that helps itself recover resources held by non-existing processes. This mechanism will periodically check the status of all processes using the agent service. When it discovers a process that is no longer existing (leasing expired), it frees up the resources and assigns them to other processes.

D. AGENT OPERATIONS

Agent interfaces offer processes a simple and yet comprehensive set of operations to interact with one another in a distributed environment, from establishing of connections to handling of entries. Generally, we have grouped them into 5 types of

operations: service registration operations, entry operations, transaction operations, event-handling operations and advance operations.

1. Service registration operations

Service registration operations provide processes the means to discover and register with agent service as well as to retrieve the handles of common services. Figure 5 shows a service registration process. To begin, a process first has to locate a lookup service that agent service publishes itself. Lookup service is like a name directory, providing facility for services to publish themselves whereby clients can easily locate them. If the process knows exactly where the agent service is located, it can proceed directly to service registration, otherwise, it has to perform a network search using JINI discovery protocol – place a request package onto the network and wait for lookup services to response. After registration the process can retrieve the handles of services needed for inter-process communication such as transaction service, JavaSpace service and etc.

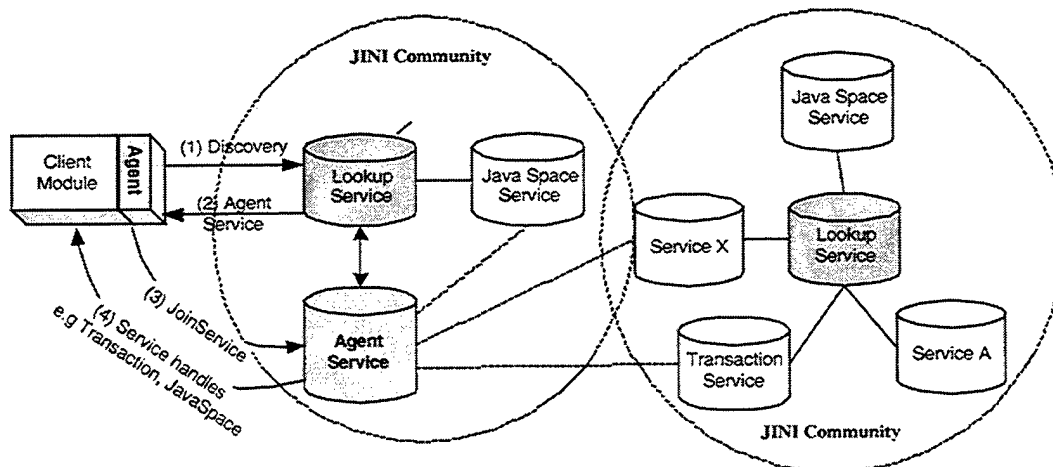


Figure 5 Agent Service Registration Process

2. Entry operations

Entry operations allow processes to manipulate entries store in the repository service. There are five kinds of entry operations that the agent interface provides, namely: read, write, update, take and notify. A read operation and a take operation, as depicted in figure 6a & 6b, allow processes to retrieve a copy of an entry that matches a given template stored in the repository. Unlike a read operation, a take operation will remove the entry from the repository after retrieving. On the other hand, a write operation places a new entry in the repository while an update operation overwrites any existing entries having the same entry ID. The Notify operation, shown in figure 6e, allows process to monitor for incoming entries to the repository - when a new entry is added and matches the entry ID being monitored, process would be notified by an event.

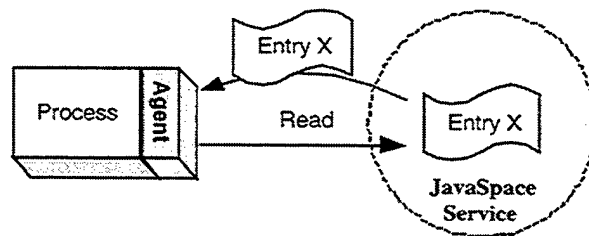


Figure 6a Entry Read Operation

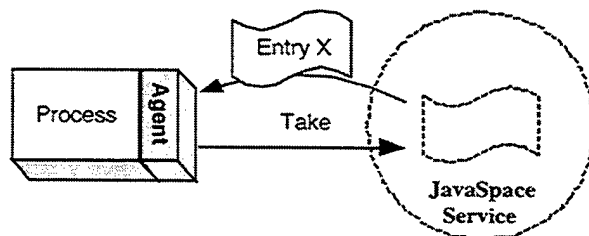


Figure 6b Entry Take Operation

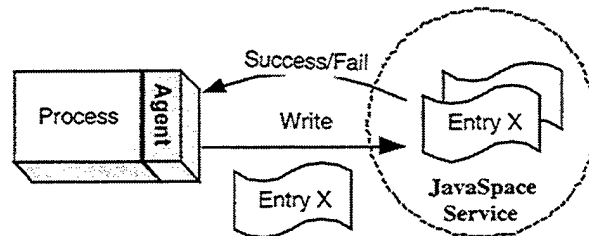


Figure 6c Entry Write Operation

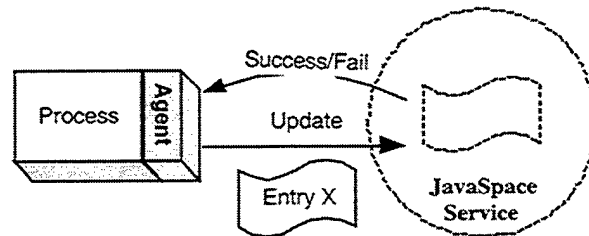


Figure 6d Entry Update Operation

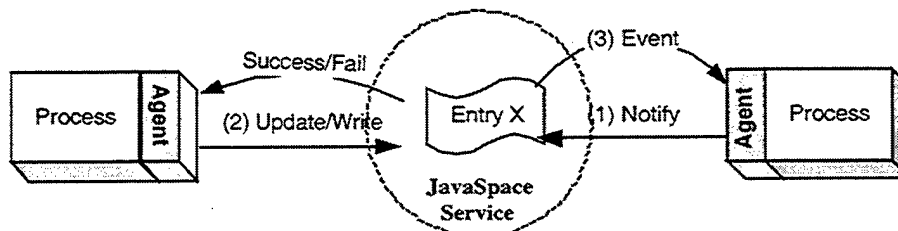


Figure 6e Entry Notify Operation

Every entry operation that process performs is constrained by a set of entry attributes that is described in details in session III-E. For instance, these attributes limit how long a read operation should wait for an entry if it is not available in the repository, how long an entry should remain valid before being removed away from the repository and etc. Although these attributes are preloaded during declaration, designers can change them to meet individual application needs.

3. Transaction operations

Transaction operations allow processes to enforce consistency over a set of entry operations. When a process starts a transaction, entry operations carry out under this

transaction will be considered as a group operation. If the transaction completes successfully, then all entry operations associated to it are considered completed. However, if any problem arises in one of the operations, then the transaction would be aborted and all other operations are cancelled. Though using transaction would decrease the response time of entry operations, it helps to address one of the most difficult issues in distributed environment - partial failure.

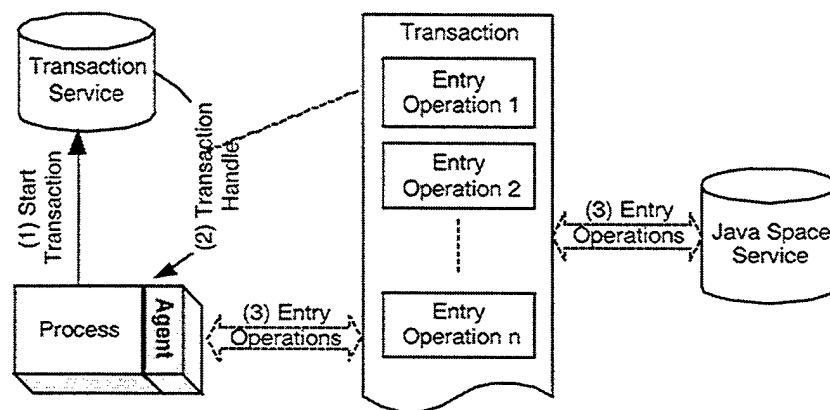


Figure 7 Entry Operations with Transaction

As depicted in Figure 7, in order to bind entry operations with a transaction, the agent first has to request for a handle from the transaction service. Subsequently, it passes this handle along with other required parameters to carry out the entry operations. Every handle comes with an expiring time - it automatically terminates the transaction if operations issued are not completed within a given time. When the agent completes its entry operations, it has to inform the transaction service to close the transaction.

4. Event Handling

When a new entry arrives, the repository service creates a notification event. The event Handling allows processes to be triggered by notification event. For a process to be

triggered, it first has to invoke a notify operation, specifying the entry it wishes to monitor and the callback routine it defines to handle the trigger. The agent, upon receiving a notification event, performs some internal filtering and sorting to determine the owner and then triggers the callback routine.

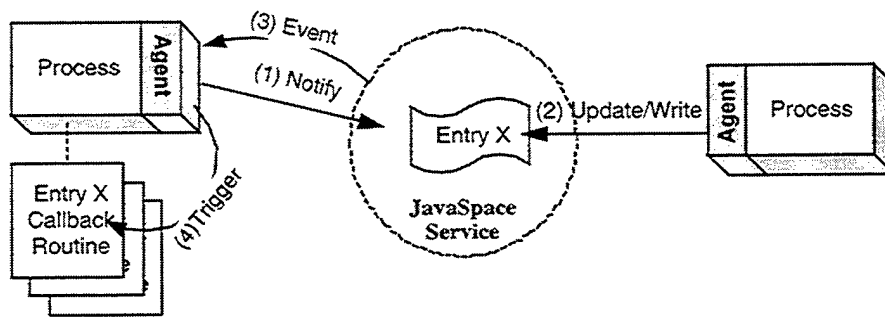


Figure 8 Event Handling

Event notification is especially important in a distributed system. Rather than “busy waiting” for an entry to arrive in the repository service through polling, which is a burden to the entire network load, processes can make use of notification operation whereby a user-defined routine would be called when conditions are satisfied.

However, in our current design we do not guarantee the delivery of all notification events raised by the service. In the worst case, where system fails or network congestion occurs, a notification event package might not reach its destination. Since no acknowledgment is returned to the sender, the notification event is considered lost. One way to overcome this problem is to use an event mailbox service, which keeps a copy of all outgoing events and keep delivering the event until the recipient acknowledges.

5. Code Serialization

Leveraging on the Java code portability, our agent contains some operations that allow processes to embed codes into entries for executing in other processes. It performs the function of packing codes into entries and resembles the code in the agent for execution, as shown Figure 9.

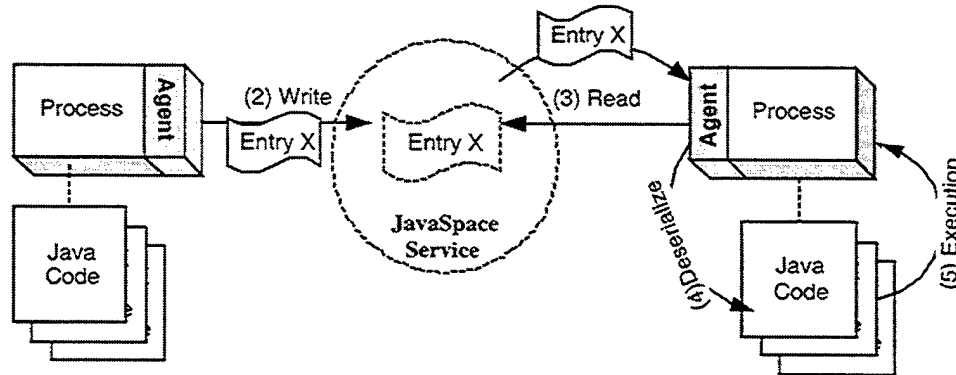


Figure 9 Code Serialization

E. AGENT ATTRIBUTES

Agent Attributes affect how the agent carries out its operations. Two of the attributes are lease time and time out, and they are applied in many areas. The designer can change the default attribute setting after initializing the agent to meet individual application needs.

1. Leasing

Leasing ensures that the resources used will not grow out of bound. Especially, in an environment where partial failure is frequent and processes are disconnected from the resources and cannot explicitly free them. A lease must be renewed before it expires, otherwise, the resources that have been requested under this lease will be released. Transparent to developers, the agent service has to periodically renew its lease with

JINI/lookup service to maintain the interest in using its facilities to publish its service. It is similar for agents to renew its lease in the agent service.

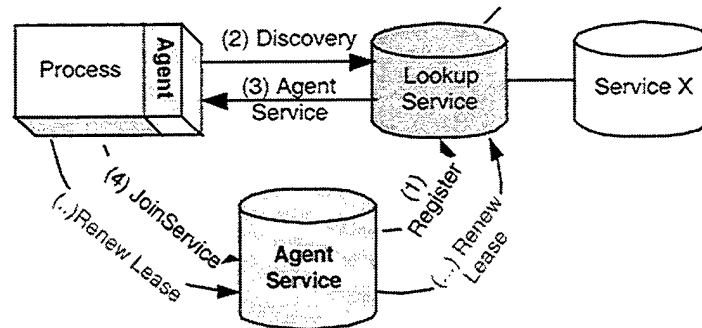


Figure 10 Renewing Service Lease

Every entry posted to space must be leased. By default it is given a lease of one minute. After the lease time expired, the service will free the resource occupied by the entry. Similarly, the transaction service will abort all transaction requests right after the lease expires.

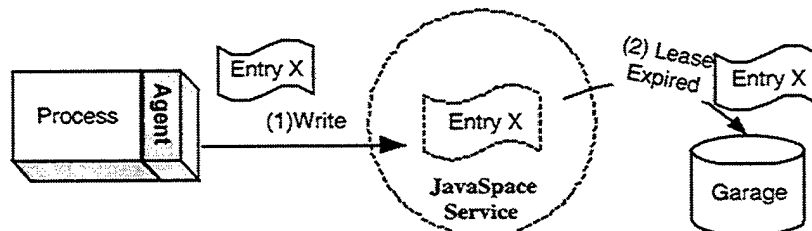


Figure 11 Entry Lease Expired

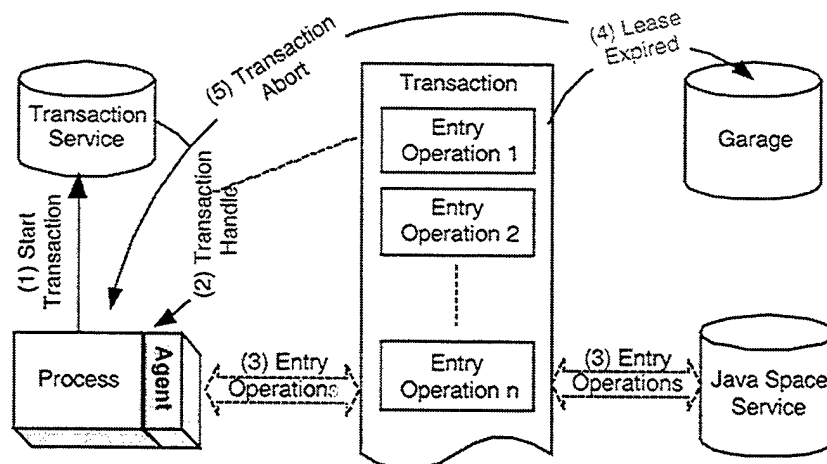


Figure 12 Transaction Lease Expired

Developers must provide due consideration towards leasing while developing the application. Although the developer can set a longer lease time that would not expire, this could put some constraints on services that still have to maintain the resource used when the owner has been disconnected from the network and cannot explicitly free them.

2. Timeout

In a distributed environment, it is difficult to determine whether a process becomes unavailable due to a system failure or because the CPU is busy with some other jobs. Timeout offers the means for the process to give up its operations and turn to other jobs. Without timeout, process could be deadlocked to an operation when a certain system failure occurs, i.e., waiting endlessly until someone intervenes. Figure 13 and Figure 14 show diagrams of service timeout and Entry Read/Write operation timeout respectively.

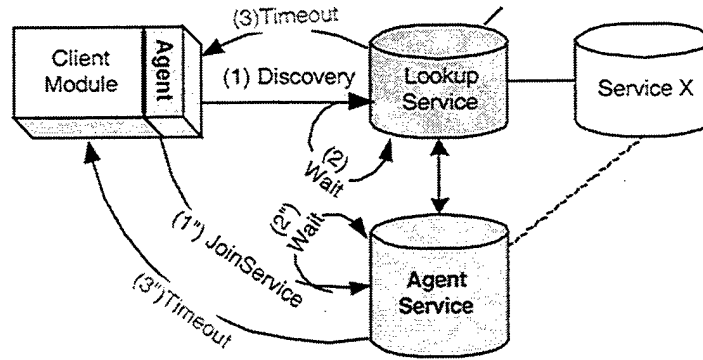


Figure 13 Service Timeout

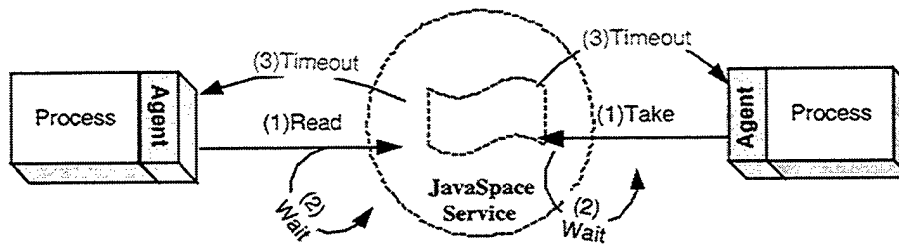


Figure 14 Entry Read/Take timeout

F. AGENT WRAPPERS

There are many compelling reasons for the agent to support a wider variety of programming languages. Some of these reasons are software re-use, integration with legacy code, leveraging on tools that are not available for a particular programming language, and performing low-level activities such as hardware interface.

In the thesis, we provide two types of agent wrappers, ActiveX Component Wrapper and Native C Library wrapper. ActiveX Component Wrapper allows our agents to be encapsulated as objects in Visual Basic, Visual C++ or Microsoft Office applications running in Microsoft Window platform, whereas Native C Library Wrapper allows our agent to be bound together with native languages such as Ada, C and C++.

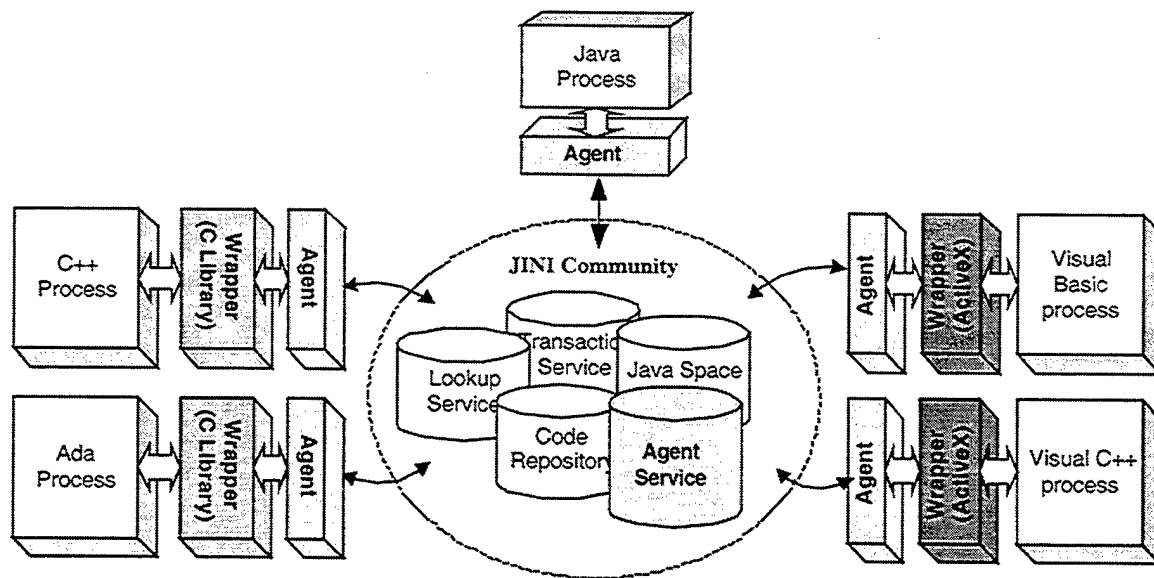


Figure 15 Agent Wrappers

IV. IMPLEMENTATION

This session describes our framework architecture and its implementation details.

A. ARCHITECTURE

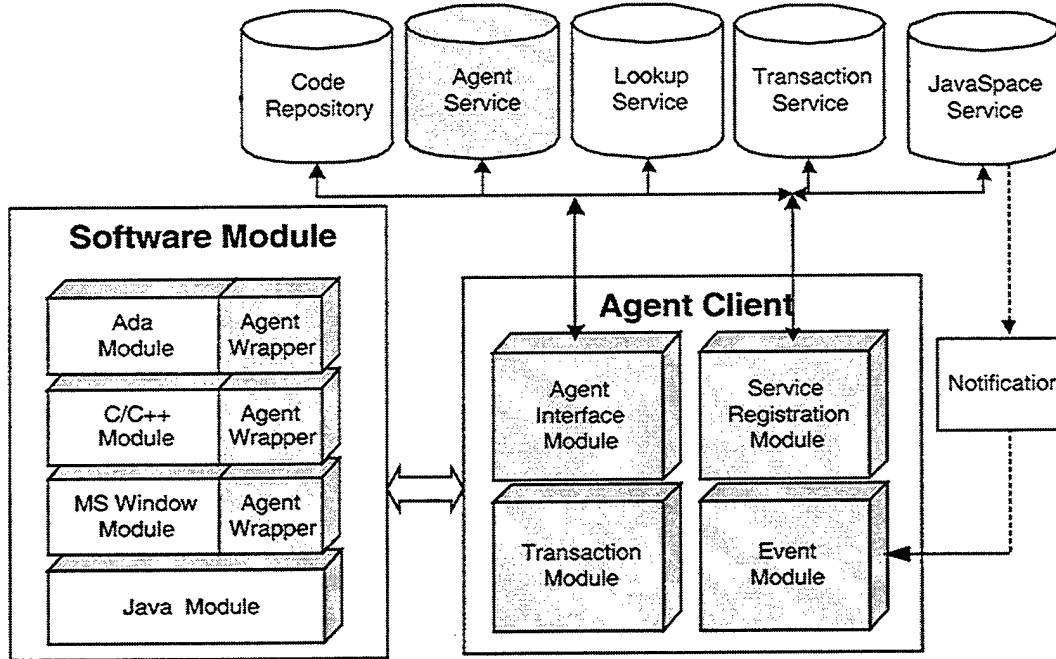


Figure 16 Agent Architecture

Our agent architecture, as depicted in Figure 17, consists of six main modules. They are Agent Service module, Service Registration module, Agent Interface module, Event Module, Transaction Module and Agent Wrapper module. Agent Service module implements a service to manage and control the access of agents into the network environment. Service Registration module offers the means for processes to locate and register with the agent service. Agent Interface module provides processes a simple interface to access the agent operations. The Event module handles notification event issued by the repository service when new entry arrives at the repository. Transaction Module supports the entry operations with transaction services. Lastly, Agent Wrappers

expose the interfaces of the agent to processes written in other languages such as Ada, C, C++ and Visual Basic/C++

B. AGENT SERVICE MODULE

Agent service module implements authentication and control mechanism for processes running in the network environment. It also publishes an *agentService* interface to inform clients what methods are available in the service where they invoke. The *agentService* interface class consists of two methods, *joinService* and *endService*. *JoinService* method allows client to linkup with agent service and *endService* terminates connection with service. The *JoinService* method takes 2 parameters, a *name* and a *password*. Both parameters are used to verify the client identification and must match one of the records held inside its service database.

```
public interface agentService {  
    public boolean joinService(String name, String Password);  
    public boolean endService();  
}
```

C. SERVICE REGISTRATION MODULE

Service Registration Module offers the means for the process to locate and register with the service. It composes of two classes: *ServiceAccessor* and *ServiceFinder*. *ServiceAccessor* class provides the method for processes to discover and register with services in the JINI community. *ServiceFinder* class, on the other hand, helps the *ServiceAccessor* to determine whether a request exists in a list of services that it has discovered.

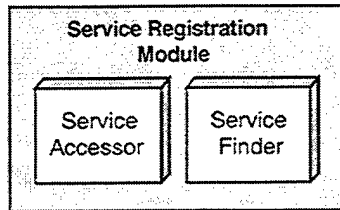


Figure 17 Service Registration Module

1. Service Accessor

ServiceAccessor class provides the method for processes to discover and register with services in the JINI community. Service discovery is done using the *GetLocator* method and service registration is performed using the *getService* method.

```

public class ServiceAccessor implements java.io.Serializable{
    public static Locator getLocator( long lookupTimeout)
    public static AgentService getAgent()
    public static AgentService getAgent(String name)
    public static JavaSpace getSpace()
    public static JavaSpace getSpace(String name)
    public static TransactionManager getTransaction ()
    public static TransactionManager getTransaction (String name)
}
  
```

GetLocator method takes one parameter - timeout, which defines a maximum time (in millisecond) that it can spend searching for a lookup service (a facility where services in the JINI community publish their services). If it fails to locate one within a given time, an exception is returned to the calling process.

GetService methods (e.g. *getAgent*, *getSpace* and *getTransaction*) return the handle of services that a process has registered with, otherwise they return a null value. These methods require processes to provide names that can identify the services if they are being setup using different names.

2. Service Finder

ServiceFinder class provides a method, *find*, for ServiceAccessor to determine whether a requested service exists in the list of services that it has discovered. The *find* method takes three parameters –*locator*, *name* and *timeout*. *Locator* is a valid lookup service handle where the request service is likely to have registered. *Name* is a string that identifies the requested service. *Timeout* is the time in millisecond that limits the search duration.

```
public class ServiceFinder extends Finder {  
    public ServiceFinder() {}  
    public Object find (Locator locator, String name, long timeout) {  
    }  
}
```

D. EVENT MODULE

The event module composes of three classes: *SpaceEventRegistration*, *SpaceEventListener* and *SpaceActionHandler*. *SpaceEventRegistration* class registers a notification event with the repository service. *SpaceEventListener* class monitors notification events generated from the repository service. *SpaceActionHandler* class analyzes the notification events and takes appropriate actions

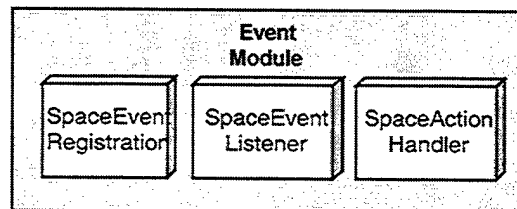


Figure 18 Event Module

1. Space Event Registration

The *SpaceEventRegistration* class is an interface responsible for registering notification event with the repository service to monitor new entries written into the

space by other cooperating agents. Every entry handler implements this interface to provide a consistent way of starting and stopping notification events. This class consists of two types of abstract methods: *startEvent* and *stopEvent*.

```
public interface SpaceEventRegistration
{
    public abstract boolean startEvent();
    public abstract boolean startEvent1(SpaceActionHandler spaceAction);
    public abstract boolean startEvent2(SpaceActionHandler spaceAction, long lease );
    public abstract boolean startEvent3(SpaceActionHandler spaceAction, long lease ,
        Transaction txn);
    public abstract boolean stopEvent();
}
```

StartEvent method takes three parameters (if not specified, the default values will be used): *SpaceActionHandler*, *Lease* and *Transaction*. *SpaceActionHandler* is a valid object that is assigned to handle the notification event. *Lease* determines the expiring time in millisecond for a notification. *Transaction* is a valid handle, requested from Jini/Transaction service if a transaction is invoked. *StopEvent* method removes the event registration from the Repository service. It returns an exception if it fails to remove the registration or if the registration is already expired.

2. Space Event Listener

The Space Event Listener class listens to events issued by the repository service. It implements a RemoteEventListener interface, which includes a notify method that the repository service used for raising remote events when new entries arrive at its repository. The constructor has three parameters: *entryType*, *entryID*, and *SpaceActionHandler* object. Both *entryType* and *entryID* are required to identify the entry being monitored. *SpaceActionHandle* is a valid object that implements the SpaceActionHandler interface for handling of notification events.

```

public class SpaceEventListener implements RemoteEventListener {
    private int eventType;
    private String eventID;
    private SpaceActionHandler action;

    public SpaceEventListener(int entryType, String entryID, SpaceActionHandler action)
        throws RemoteException
    public void notify(RemoteEvent ev)
}

```

3. Space Action handler

SpaceActionHandler is a simple interface class with an abstract *ActionPerformed* method and *fire* method. The agent implements this interface to handle notification event. *ActionPerformed* is a callback method that encapsulates actions for handling of the notification events; any class implementing this interface must overwrite the *actionPerformed* method. The *ActionPerformed method* is triggered every time SpaceEventListener invokes the *fire* method, together with an *entryType* and *entryID* to identify the entry that cause the notification.

```

public interface SpaceActionHandler extends ActionListener {
    public void actionPerformed(ActionEvent e);
    public void fire(int entryType, String entryID);
}

```

E. INTERFACE MODULE

The interface module composes of three sub-modules: Application Interface, Entry Handlers and Entry templates. Application Interface provides a simple interface for the process to access the agent's functions. Entry Handlers allow the process to carry out operations pertaining to the declared entries. Entry templates are used for retrieving entries from the repository service.

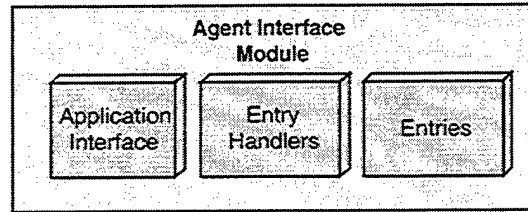


Figure 19 Agent Interface Module

1. Application Interface

The application Interface provides a simple way for processes to access the agent's operations. This interface can be sub-divided into 3 groups of methods: agent configuration, entry declaration, transaction handling.

a. Agent configuration

Agent configuration methods allow processes to change the setup properties as well as to initialize and to terminate connections with the services. Many of these properties determine how connections with the services are setup, thus changing these properties has to be done before invoking the *initAgent* method. In total, there are 6 properties that are modifiable: *AgentSecurityPolicy*, *AgentServerCodebase*, *AgentSpaceName*, *AgentLookupGroup*, *AgentSecurityPolicy* and *AgentLookupURL*. Modifying and reading these properties can be done using their respective *setAgentProperty* and *getAgentProperty* methods. *AgentSecurityPolicy* property defines a policy file that controls all security permissions granted to the JVM. *AgentServerCodebase* property specifies the location of the code server where agents can download the required codes. *AgentSpaceName* property denotes the name of the repository service. *AgentLookupGroup* property indicates the group type to join in, either

public or private. And *AgentLookupURL* property defines the location of the repository service that the agent should start looking for this service.

InitAgent method allows a process to establish connections with services; this has to be done explicitly after the process has started. *TerminateAgent* method gives the system a chance to release allocated resources and relinquish service connections before a process ends. Processes written in other languages besides Java Programming languages have to explicitly call this method before termination to prevent any possible memory leaks.

```
public class Agent implements java.io.Serializable, SpaceActionHandler, TSConstants
{
    .
    .
    // System setting
    public void setAgentSecurityPolicy(String str)
    public void setAgentSpaceName(String str)
    public void setAgentServerCodebase(String str)
    public void setAgentLookupGroup(String str)
    public void setAgentLookupURL(String str)

    public String getAgentSecurityPolicy()
    public String getAgentSpaceName()
    public String getAgentServerCodebase()
    public String getAgentLookupGroup()
    public String getAgentLookupURL()

    public boolean InitAgent(long timeout)
    public void TerminateAgent()
    .
}
```

b. Entry declaration

Entry declaration methods allow a process to declare entries it wants to share with other processes running in the network. There are three methods provided, namely *createID*, *removeID* and *getTSObjec*. *CreateID* method creates a new entry handler to manage the declared entry. It takes 2 input parameters, *entryID* and *entryType*.

Entry ID must be a unique string and *entryType* must be one of the following enumerate types: TS_BOOLEAN, TS_INTEGER, TS_FLOAT, TS_DOUBLE, TS_LONG, TS_STRING, TS_HASH, TS_QUEUE, TS_LIST or TS_STACK. Each enumerate type symbolize the kind of entry the process wants to create. *RemoveID* method purges the entry handler that had been created. Both *CreateID* and *RemoveID* methods return a *true* value if they succeeded in completing the actions, otherwise they return a *false* value. *GetTSObject* method returns the handle of an entry handler only if the handler exists. Otherwise, it returns a null value indicating an error has occurred.

```
public class Agent implements java.io.Serializable, SpaceActionHandler, TSConstants
{
    private Map tsStringMap;
    private Map tsBooleanMap;
    private Map tsIntegerMap;
    private Map tsFloatMap;
    private Map tsLongMap;
    private Map tsDoubleMap;
    private Map tsQueueMap;
    private Map tsStackMap;
    private Map tsLinkListMap;
    private Map tsHashMap;
    .
    .
    public boolean createID(int entryType, String entryID)
    public boolean removeID(int entryType, String entryID)
    public Object getTSObject(int entryType, String entryID)
}
```

c. *Transaction handling*

Transaction handling methods allow a process to start, stop, abort and query transaction. *StartTransaction* method starts a new transaction, registers a transaction request with the running transaction service. *CloseTransaction* completes the transaction, commits all entry operations that have been carried after the start transaction and relinquishes the transaction handle. *GetTransaction* method returns the current

transaction handle. *AbortTransaction* terminates the transaction; all the entry operations that have been invoked are cancelled.

To perform a transaction, the process first has to invoke the *startTransaction* method, where the agent would then get the transaction handle using *getTransaction* methods, and then to pass it together with every entry operation that is executed under this transaction. Finally, the process should invoke a *closeTransaction* method to commit the transaction. If any operation fails to complete or the transaction's duration fall beyond the given lease time, all issues that the operations have defined will be recalled.

```
public class Agent implements java.io.Serializable, SpaceActionHandler, TSConstants
{
    .
    .
    public boolean abortTransaction()
    public boolean closeTransaction
    public boolean startTransaction()
    public Transaction getTransaction()
    public boolean isTransactionStarted()
    .
}
```

2. Entry Handlers

Entry Handlers allow processes to carry out operations pertaining to their entries. In total, there are 10 types of entry handlers; each corresponds to an entry type. For instance, an integer handler (*TSInteger*) corresponds to an integer entry (*EntryInteger*), offering methods needed to operate an integer entry in the repository service.

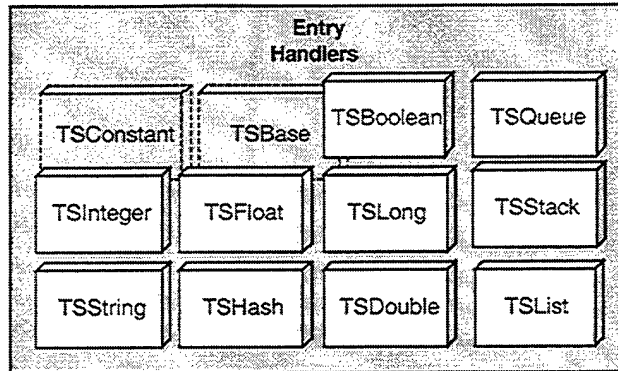


Figure 20 Entry Handlers

Every Entry Handler inherits from a base class, *TSBase*. This class contains a list of entry attributes that determine how its entry behaves or how it should handle entry within the repository service. Upon initializing, these attributes are preloaded with default values. Processes can easily modify and query them by invoking their respective *setAttribute* and *getAttribute* methods. For example, to change or to check read-timeout attribute, a process can invoke *setReadTimeout* and *getReadTimeout* respectively.

```

public class TSBase implements java.io.Serializable{
    public TSBase() {
        writeLeaseTime = Lease.FOREVER;
        updateLeaseTime = Lease.FOREVER;
        notifyLeaseTime = 10000;
        readTimeOut = Long.MAX_VALUE;
        takeTimeOut = Long.MAX_VALUE;
    }
    public void setJavaSpace(JavaSpace space)
    public void setSpaceActionHandler( SpaceActionHandler spaceAction)
    public void setTransactionHandler( Transaction trans)
    public void setWriteLeaseTime(long leaseTime)
    public void setUpdateLeaseTime(long leaseTime)
    public void setReadTimeOut(long timeOut)
    public void setTakeTimeOut(long timeOut)

    public Transaction getTransactionHandler()
    public long getWriteLeaseTime(){
    public long getUpdateLeaseTime(){
    public long getReadTimeOut()
    public long getTakeTimeOut()
}
  
```

Every Entry Handler contains a common set of methods, but with some variation in input parameters depending on its entry type. These methods are *readIfExists*, *takeIfExists*, *write*, *update*, *startEvent* and *stopEvent*. *ReadIfExists* method returns the entry value if it manages to find the request entry from the repository service. Similarly, *takeIfExists* also returns the entry value but would remove the request entry from the repository service. Both methods take up to two input parameters: *timeOut* and *transaction*. The value of *Timeout* determines how long the handler should wait for an entry if it is available but still being held back by the repository service because of a transaction operation in progress. *Transaction* is a handle provided by transaction service to carry out a transaction. Both *write* and *update* methods create a new entry that contains an entry value and put it into the repository service. Unlike *write* method, which just adds a new entry to the repository service; *update* method removes any existing entries, having the same entry name before placing a new one. Lease and transaction inputs are optional. Lease determines how long an entry remains valid in the repository service, after this time it would be automatically remove from repository. *StartEvent* and *stopEvent* methods allow a process to register and terminate entry notification with the repository service respectively.

```
public class TSEntryType extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
{
    public boolean write1(dataType value)
    public boolean write2(dataType value, long lease)
    public boolean write3(dataType value, long lease, Transaction txn)

    public boolean update1(dataType value)
    public boolean update2(dataType value, long lease)
    public boolean update3(dataType value, long lease, Transaction txn)

    public dataType readIfExists()
    public dataType readIfExists1(long timeOut)
    public dataType readIfExists2(long timeOut, Transaction txn)
```

```

    public dataType takelfExists()
    public dataType takelfExists1( long timeOut)
    public dataType takelfExists2( long timeOut, Transaction txn)

    public boolean startEvent()
    public boolean startEvent1(SpaceActionHandler spaceAction)
    public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
    public boolean startEvent3(SpaceActionHandler spaceAction, long lease ,
                                Transaction txn)
    public boolean stopEvent()

}

```

3. Entries

Entries are serialized Java object instances that are stored within the repository service, where processes used them as a means to share information. In total, there are 10 types of common entries created ranging from primitive type entry like integer entry to more complex type entry like list and queue entry. Each entry class implements the Entry interface found in the net.jinni.core.entry.Entry package.

Below is an example of an entry class - *EntryHash*. *EntryHash* allows a process to define its own data structure. Its constructor takes one parameter or two parameters: *entryID* and *map* Object (Optional). The *EntryID* identifies the entry placed in the repository service and the *map* object is a container that holds the data fields defined by the process.

```

public class EntryHash implements Entry {
    public EntryHash(String entryID)
    public EntryHash(String entryID, HashMap map)
    public HashMap getHashMap()
    public void setHashMap(HashMap map)
}

```

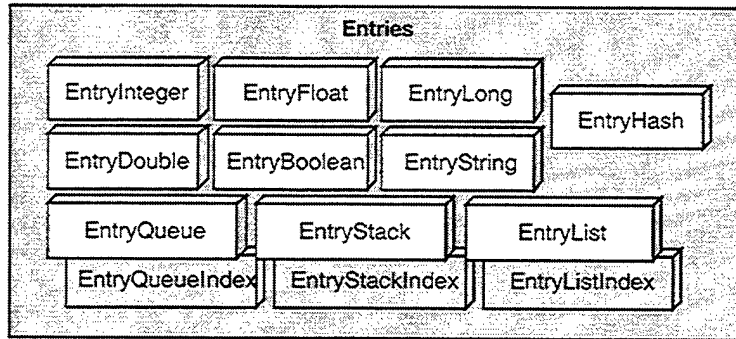


Figure 21 Entries

F. AGENT WRAPPER MODULES

The Agent wrapper module consists of two separate components: ActiveX wrapper and C Library wrapper. ActiveX wrapper embeds the agent as an object such that processes written in Visual Basic, Visual C++ or Microsoft Office applications running in the Window environment can call it. C Library wrapper allows the agent to be bound together with processes written using machine dependents languages like C, C++ or Ada.

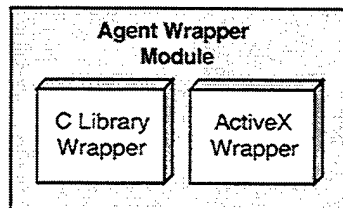


Figure 22 Agent Wrapper Module

The ActiveX Wrapper is implemented by using a packager, ActiveX Packager for Java Bean, that comes along with JVM plug-in provided by the Sun MicroSystem. This packager automatically generates the wrapper for any Java bean by going through the procedure of pre-compiling. Two files are eventually generated after the process, an OCX (OLE Control Extension) and a TLB (Type LiBrary). To make the OCX available to the Window environment, developers should explicitly register them in the window registry.

Together with the Java Bean Bridge and JVM (Java Runtime Environment), any method calls on this OCX component will marshal over the bridge and gets executed in the JRE memory space; the return for the function is unmarshaled by the bridge and given back to the OCX component.

The C Library Wrapper is built by using JNI (Java Native Interface) APIs. The procedure is more complicated and tedious than that of ActiveX Wrapper. We have to map every Java types to C, create corresponding interfaces in C language for every method defined, and manage the memory resources to prevent any memory leak.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS

This session presents the results of an experiment conducted to measure the response time of JINI/JavaSpace service for inter-process communications. It also provides some guidelines on constructing a distributed system by using the proposed agent framework. An Elevator Control System (ECS) is developed as an example.

A. SERVICE RESPONSE TIME

Chart 1 and Chart 2, as depicted in Figure 23 and Figure 24 respectively, show the response times of the JINI/JavaSpace Service that we have tested over a local area network on two Pentium III/500MHz machines. One machine is loaded with Jini Services (including Agent Service, Lookup Service, Transaction Service and JavaSpace Service) and the other is loaded with a client program making requests with Jini Services. The local area network traffic is pretty light – it takes an average latency time of 0.6 milisecond for a 512 bytes data package to travel across the network from one machine to the other machine when using sockets.

Both experiment are performed over the duration of 2 minutes with a total of 1,000 test cases. Each case consists of 5 roundtrip requests, 5 read and 5 write operations. Roundtrip response time, time between the issue of a complete write operation follow by a successful read operation, was measured by taking the average of 5 roundtrip requests. The result of each case was taken and plotted on the charts below.

The average roundtrip respond times in Chart 1 (1,000 tuples in the repository) and Chart 2 (10,000 tuples in repository) are 12.4 milisecond and 12.03 milisecond respectively. Chart 2, despite having more tuples, has a lower average roundtrip respond

time. We believe that it is due to fluctuation in the network traffic rather than changes in system performance.

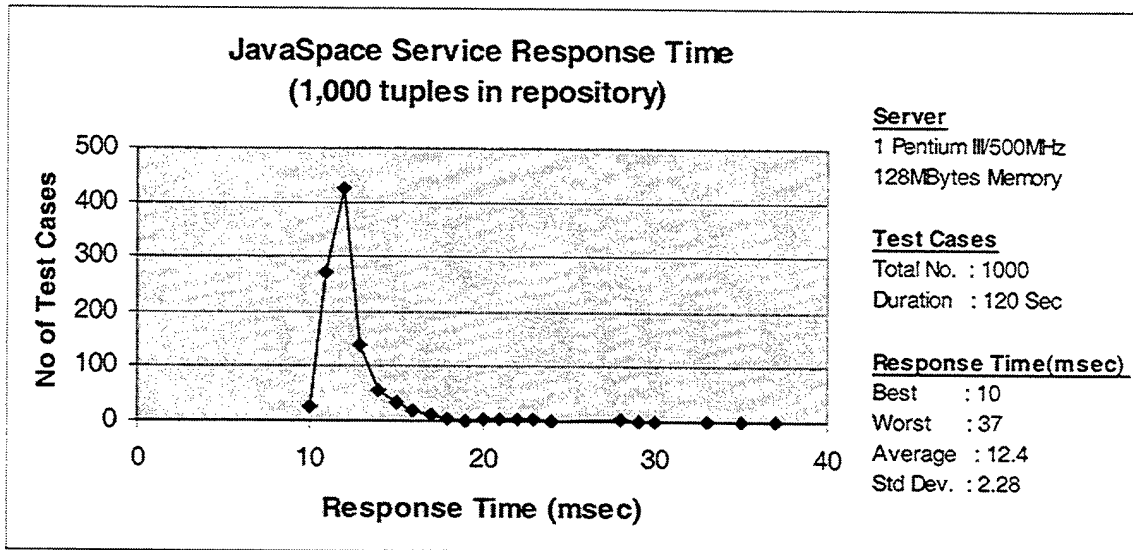


Figure 23 Chart 1 JINI/JavaSpace Response Time

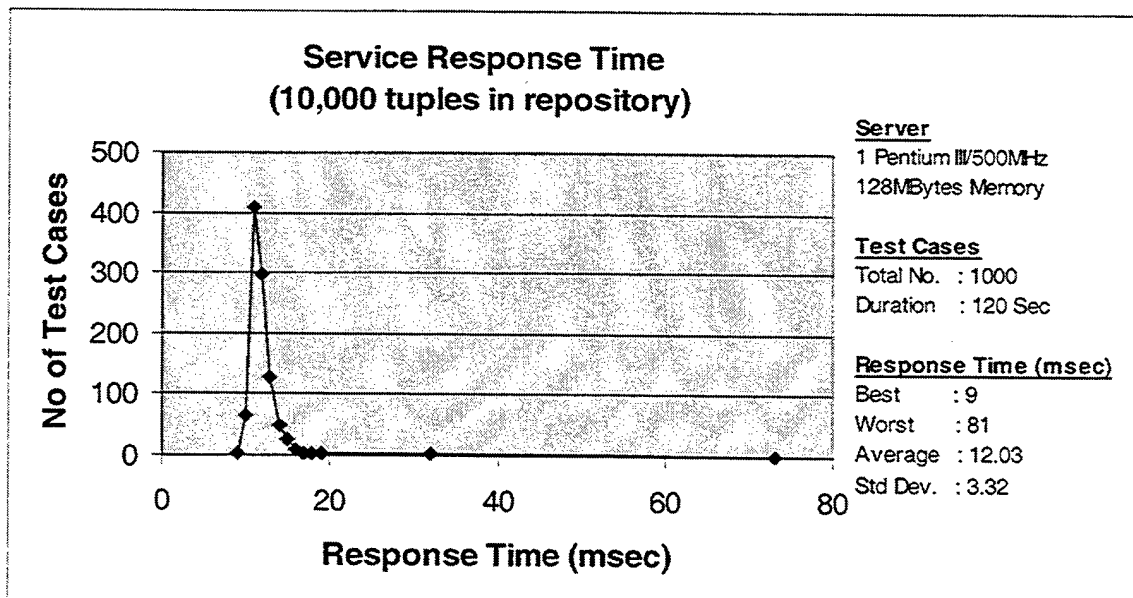


Figure 24 Chart 2 JINI/JavaSpace Response Time

B. DEVELOPMENT GUIDELINES

The following presents some guidelines on how to develop distributed applications using our agent framework; developers are advised to try out the test programs to get familiarized with the features that our framework provides.

1. Identify the number of independent processes required for an application.
They can either reside on the same machine or on different machines but must share a common network.
2. Determine the number of entries and their types need to be defined in the repository service whereby different processes can use them for exchanging information.
3. Use the appropriate wrapper if processes are written in other languages besides Java Language. ActiveX wrapper for Delphi, Visual Basic and Visual V++ processes, and C Library wrapper for C, C++ and Ada processes.
4. Within every process,
 - a. Instantiate an agent
 - b. Update the system setting; Agent Service, Security Policy file, Server Codebase, Lookup Group and Lookup URL before establishing connections with the services.
 - c. Invoke initAgent method to establish connections with the services. It takes up to a maximum of 20 seconds to establish a connection.

- d. Declare all entries, defined earlier, with the instantiated agent. Every entry declared has to be identified by a unique name otherwise it will overwrite any earlier declaration.
5. Use the appropriate entry operation to access entries stored in the repository service. It is important to minimize the number of operations as far as possible since the network latency is about 10 milliseconds as compared to a few nanoseconds for accessing a local shared variable within a process.

C. TESTING JINI/SERVICES

We have created 3 test programs, which are written in different languages, to test the configurations of services and client processes. Figure 25a, 25b and 25c showed the screen captures of these test programs that are implemented in Java language, Visual Basic language and C language respectively. Each test program is a GUI component enhanced by an agent. These test programs consist of many useful functions described in the framework. Besides using them for testing the configurations, they can also be used to demonstrate how our framework works.

We also included a simple script in Appendix B to simplify the complicated process of setting up the JINI/Services. Designers can load this script with JINI service starter toolkits provided by the Sun Microsystems to start the services. A detailed description on how to setup JINI Services can also be found in [KEI99].

After all the JINI services have been started, run any of the test programs on a separate machine anywhere in the network. Updates the agent setting, if the setup parameters are different from the default values, and then press the “initAgents” button to

establish connection with the agent service. If successful it will return a “service connected” message. Otherwise, it returns an error message indicating the problem encountered during initialization.

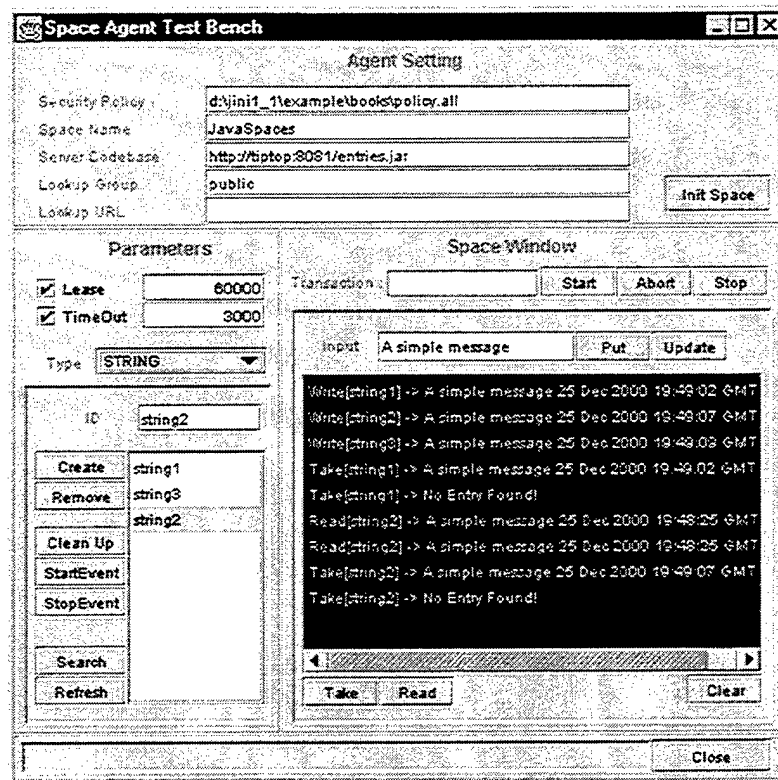


Figure 25a Agent Test Bench (Java Language version)

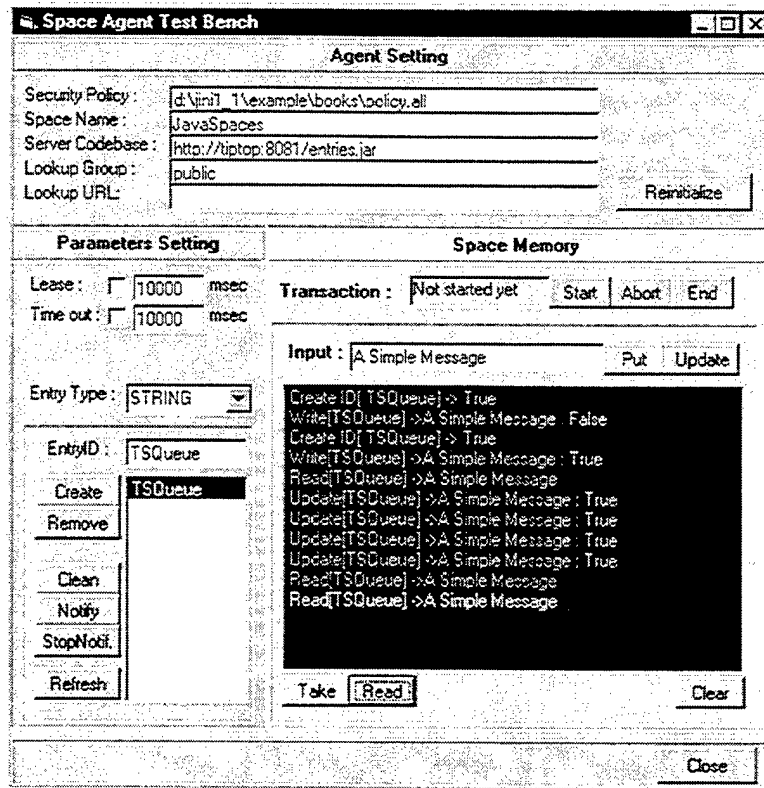


Figure 25b Agent Test Bench (Visual Basic Language version)

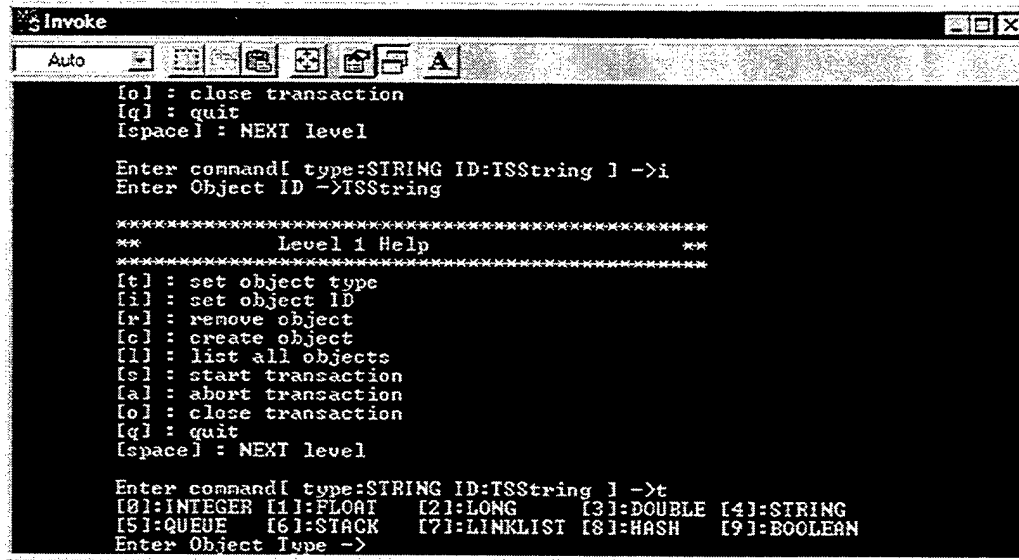


Figure 25c Agent Test Bench (C Language version)

D. AN EXAMPLE: AN ELEVATOR CONTROL SYSTEM

We have designed a simple elevator control system (ECS) to demonstrate how distributed applications containing processes written in different programming languages, use our agent framework. Table 1 gives a brief description of the ECS requirements. Figure 26 is a state chart of the ECS, explaining the various operating states inside the system. It also shows the required commands to transit between states, the input guards for each command, if any, and the time duration to operate each state.

We have divided the ECS into six independent processes: one *scheduler*, two *hardware emulators*, two *control panels*, one *floor panel*. The *Scheduler* determines an optimal solution to schedule the two elevators as well as monitors, and dispatches the commands to the hardware emulators. The *Hardware emulator* simulates the timing and sequences that an actual controller would need to control its hardware. The *Control panel* provides the interface for passengers to select their destinations, controls the closing and opening of the elevator door, and views the status of the elevator. *Floor panel* allows passenger to request for the elevator at each floor.

Each of the six processes is designed to run on a different machine and to use the agents to interact with one another. Four processes (two hardware emulators and two control panels) are written in Java programming language. One process (floor control) is written in Visual Basic language with ActiveX agent wrapper. The last process (scheduler) is written in ANSI C language with C agent wrapper. We will only illustrate in this thesis how we design the process interface and the shared data, the details on how to build the logical portions of each process in relation to the ECS functionality will not be discussed. Screen captures of the final product is shown in Figures 28A-28D.

ECS Requirements

The elevator control system controls 2 elevators in a building consisting of 10 floors. It schedules elevators to respond to requests from passengers at various floors and controls the motion of the elevators between floors.

Inside each elevator, there is a set of elevator buttons and lamps: 10 floor buttons for the passengers to select their destination, an "open" button to keep the door open, a "close" button to close the elevator door, and 10 floor lamps to indicate either the current floor if the elevator is stationary or the arriving floor if the elevator is in motion. Each elevator also has a motor that is controlled by commands to open and close the door.

At the second to the ninth floor, there is a pair of "up" and "down" call buttons for passengers to request for an elevator. A corresponding pair of lamps will indicate the directions that have been requested. There are only an "up" button and an "up" lamp in the first floor and only a "down" button and a "down" lamp in the tenth floor.

At each floor for each elevator, there is a pair of direction lamps to indicate whether an arriving elevator is heading in the up or down direction. For the top and the bottom floors, there is only one direction lamp per elevator. There is also a floor arrival sensor at each floor in each elevator shaft to detect the arrival of an elevator.

The elevator buttons, floor buttons, and floor arrival sensors are active asynchronous devices; that is, input from these devices will automatically turn on their corresponding registers. The control software is responsible to poll these registers. The registers will automatically return to the off state once polled by the control software. The other I/O devices are all passive. The elevator and floor lamps, as well as the direction lamps are switched on and off by the software.

Timing requirements:

- Elevator buttons are pressed with a maximum frequency of 5 times per second, which represents a minimum inter-arrival time of 200 msec.*
- Floor buttons are pressed with a maximum frequency of 2.5 times per second, which represents a minimum inter-arrival time of 400 msec.*

Each elevator takes at least 2 seconds to open or close its door, 9 seconds to accelerate, travel and then decelerate to the next floor, and about 1 second to travel one floor once the elevator attains its constant speed. The corresponding floor arrival sensor will be turned on when the elevator is halfway between the two floors.

Table 1 ECS Requirements

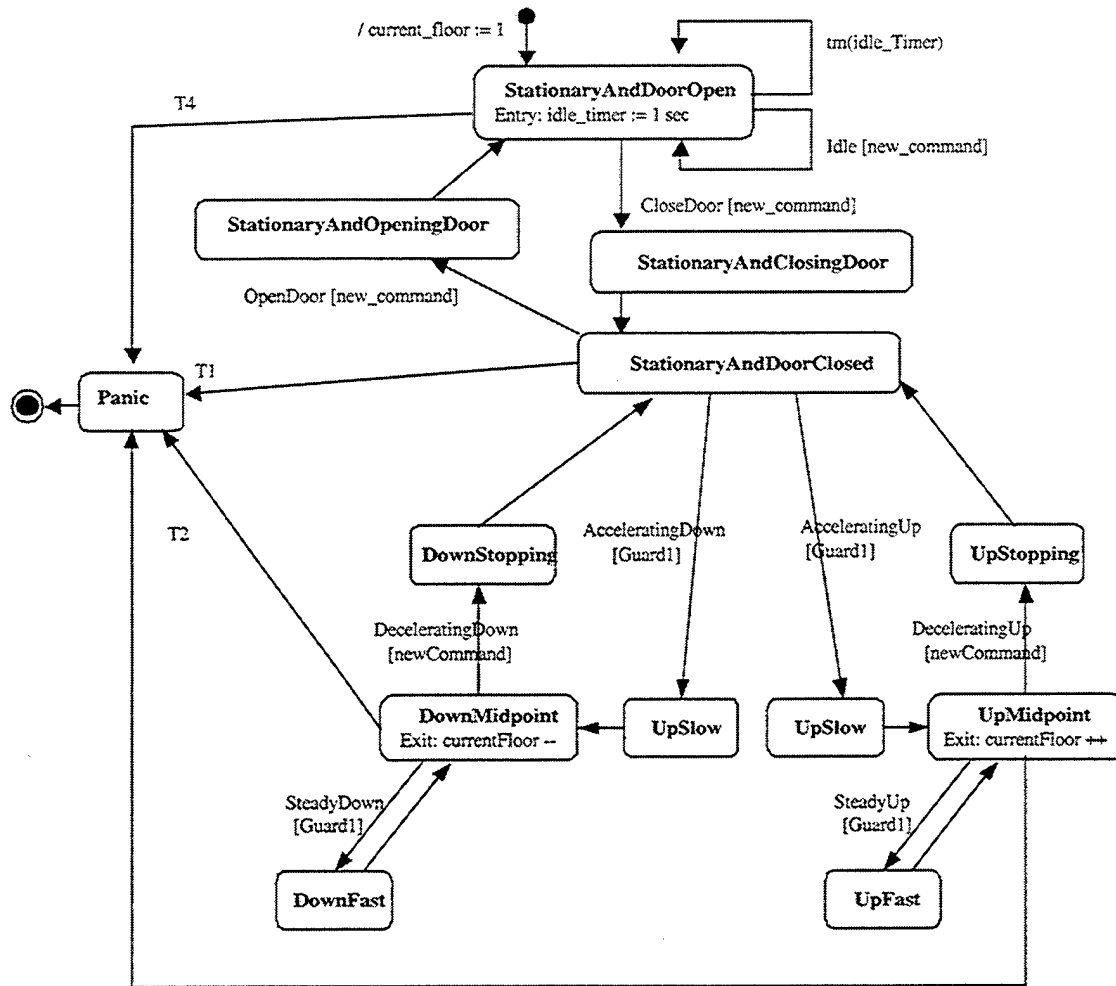


Figure 26 ECS State Chart

We have created 5 globally shared entries, as depicted in table 2, for processes to share information. These shared entries are: *AStatusEntry*, *BStatusEntry*, *AScheduleEntry*, *BScheduleEntry* and *requestEntry*. *AStatusEntry* entry and *BStatusEntry* entry stored the status of Elevator A and Elevator B respectively. For examples, push button states, elevator heading direction, door states and etc. *XstatusEntry* entry allows *hardware emulator* to update the *control panel* the status of the elevator when it transits from one state to another. Similarly, for *AScheduleEntry* entry and *BScheduleEntry* entry, they store the data (e.g. request floor) for the next schedule planned by the scheduler for Elevator A and Elevator B respectively. *Hardware emulator* read this entry to determine the next

floor to service. *RequestEntry* entries contain the floor requests submitted to the *scheduler* either from the *Control Panel* or the *Floor Panel*. Once the *scheduler* reads them, it will remove them from the repository.

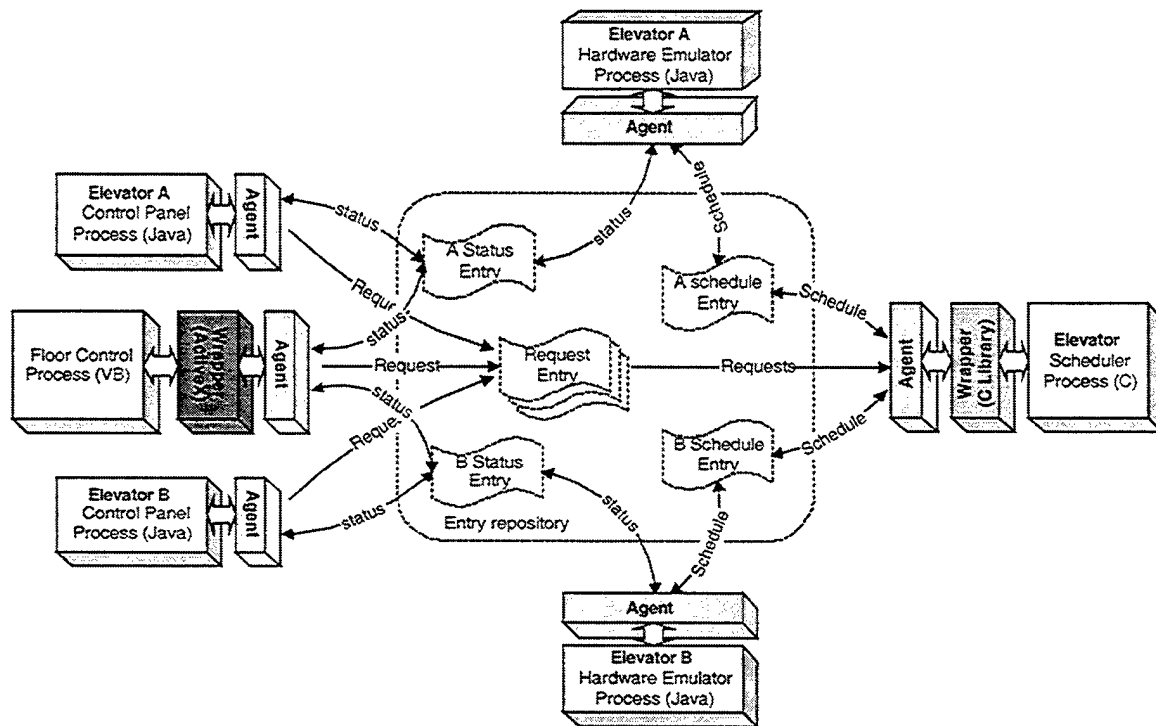


Figure 27 ECS processes

No	EntryID	EntryType	Content	Remark
1 2	AstatusEntry BStatusEntry	HashEntry	String: elevatorID Boolean: button1 Boolean: button2 Boolean: button3 Boolean: button4 Boolean: button5 Boolean: button6 Boolean: button7 Boolean: button8 Boolean: button9 Boolean: button10 Integer: direction Integer: Level	A,B T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None T: Active F:None 0:None 1:Up 2:Down 1 st - 10 th floor
3	RequestEntry	HashEntry	String: elevatorID Integer: requestLevel Integer: direction	None, A,B 1 st - 10 th floor 0:None 1:Up 2:Down
4 5	AScheduleEntry BScheduleEntry	HashEntry	String: elevatorID Integer: destination Integer: direction Integer: current Boolean: status	A,B 1 st - 10 th floor 0:None 1:Up 2:Down 1 st - 10 th floor 0:Done 1:In-progress

Table 2 ECS Shared Entries

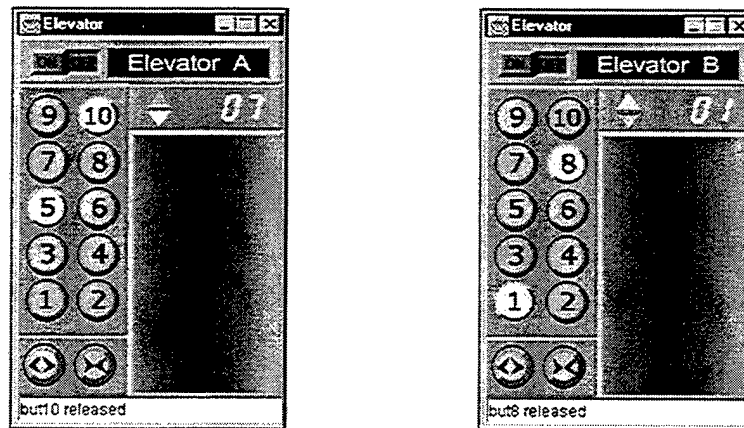


Figure 28A Elevator A & B Control Panel

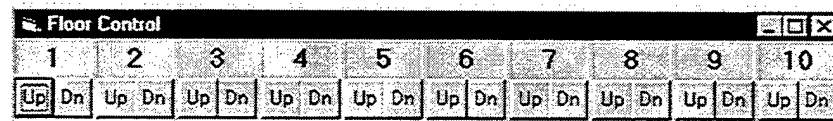


Figure 28B Floor Control Panel

1. Declare and initialize entry

The following code snippets show how we declare the entries and initialize their values in different programming languages

XstatusEntry (In Java language)

```
//Declare an status Entry
agent.createID(agent.TS_HASH , statusEntryID );

//Initialize value
TSHash ts = (TSHash) agent.getTSObject( agent.TS_HASH, statusEntryID);
if(ts != null){
    ts.setBoolean("button1",false);
    ts.setBoolean("button2",false);
    ts.setBoolean("button3",false);
    ts.setBoolean("button4",false);
    ts.setBoolean("button5",false);
    ts.setBoolean("button6",false);
    ts.setBoolean("button7",false);
    ts.setBoolean("button8",false);
    ts.setBoolean("button9",false);
    ts.setBoolean("button10",false);
    ts.setBoolean("up",true);
    ts.setBoolean("down",false);
    ts.setInteger("level",1);           // level : 1 to 10 floor
    ts.setFloat("door",(float)1.0);    // Door open percentage : 0 to 100%
}
```

requestEntry (In visual basic language)

```
'Declare an status Entry
agent.createID(TS_HASH, requestEntryID)

'Initialize value
agent.getTSObject(TS_HASH, requestEntryID).setString("elevatorID","A")
agent.getTSObject(TS_HASH, requestEntryID).setInteger("requestLevel",1)
agent.getTSObject(TS_HASH, requestEntryID).setInteger("direction",0)
```

Xschedule (In C language)

```
//Declare an status Entry
createID(TS_HASH, "scheduleEntryID");

//Initialize value
TSHASHSetString("scheduleEntryID", "elevatorID", "A");
TSHASHSetInteger("scheduleEntryID", "destination", 1);
TSHASHSetInteger("scheduleEntryID", "direction", 0);
```

```
TSHASHSetInteger("scheduleEntryID", "current",1);
```

2. Read and Write Entry

The following code snippets show how we read entries and write entries to and from the repository respectively.

XstatusEntry (In Java language)

```
// Read status entry contents
TSHash ts = (TSHash) agent.getTSObject(TS_HASH, statusEntryID);
if(ts.readIfExists()){
    frame.but1.setState(ts.getBoolean("button1"));
    frame.but2.setState(ts.getBoolean("button2"));
    frame.but3.setState(ts.getBoolean("button3"));
    frame.but4.setState(ts.getBoolean("button4"));
    frame.but5.setState(ts.getBoolean("button5"));
    frame.but6.setState(ts.getBoolean("button6"));
    frame.but7.setState(ts.getBoolean("button7"));
    frame.but8.setState(ts.getBoolean("button8"));
    frame.but9.setState(ts.getBoolean("button9"));
    frame.but10.setState(ts.getBoolean("button10"));
    frame.up.setState(ts.getBoolean("up"));
    frame.down.setState(ts.getBoolean("down"));
    frame.level.setValue(ts.getInteger("level"));
    frame.door.setValue(ts.getFloat("door"));
}

// Update status entry contents
TSHash ts = (TSHash) agent.getTSObject(TS_HASH, statusEntryID);
ts.setBoolean("button1",false);
ts.setBoolean("button2",false);
ts.setBoolean("button3",false);
ts.setBoolean("button4",false);
ts.setBoolean("button5",false);
ts.setBoolean("button6",false);
ts.setBoolean("button7",false);
ts.setBoolean("button8",false);
ts.setBoolean("button9",false);
ts.setBoolean("button10",false);
ts.setBoolean("up",true);
ts.setBoolean("down",false);
ts.setInteger("level",1);           // level : 1 to 10 floor
ts.setFloat("door", (float)1.0);    // Door open percentage : 0 to 100%
ts.update();
```

requestEntry (In visual basic language)

```
' Read schedule entry contents
if agent.getTSObject(TS_HASH, requestEntryID).readIfExists() then
    elevatorID = agent.getTSObject(TS_HASH, requestEntryID).getString("elevatorID")
    level = agent.getTSObject(TS_HASH, requestEntryID).getInteger("requestLevel")
    dir = agent.getTSObject(TS_HASH, requestEntryID).getInteger("direction")

endif

'Write a new request entry
agent.getTSObject(TS_HASH, requestEntryID).setString("elevatorID","A")
agent.getTSObject(TS_HASH, requestEntryID).setInteger("requestLevel",1)
agent.getTSObject(TS_HASH, requestEntryID).setInteger("direction",0)
agent.getTSObject(TS_HASH, requestEntryID).write()
```

Xschedule (In C language)

```
// Read schedule entry contents
if(TSHASHReadIfExist("scheduleEntryID"))
{
    TSHASHGetString("scheduleEntryID", "elevatorID", elevatorID);
    destination = TSHASHSetInteger("scheduleEntryID", "destination");
    direction = TSHASHSetInteger("scheduleEntryID", "direction");
    current = TSHASHSetInteger("scheduleEntryID", "current");
}

// Update schedule entry contents
TSHASHSetString("scheduleEntryID", "elevatorID", "A");
TSHASHSetInteger("scheduleEntryID", "destination", 1);
TSHASHSetInteger("scheduleEntryID", "direction", 0);
TSHASHSetInteger("scheduleEntryID", "current", 1);
TSHASHWrite("scheduleEntryID");
```

VI. DISCUSSION

A. JAVA PROGRAMMING LANGUAGE

We have chosen Java programming language instead of other languages to implement the framework. Using Java programming language, it allows our codes to remain portable between various computing platforms without worrying about the underlying architecture, data representation and operating system of the actual machine that they are running. Java's rich sets of APIs and object-oriented nature simplify our design. We have reused many packages that come along with JDK (Java Development Toolkit) in our implementation and our designs are simple and easy to understand.

Java programming language also offers another distinct advantage. It allows code besides data to travel across the network and run on a client machine avoiding the tedious process of system configuration. This is important especially for distributed applications because processes tend to execute on different computing platforms; maintaining and managing a uniform operating environment can be very difficult.

We foresee Java programming language to replace many of the existing native programming languages in the future. The evolution of the Internet has changed how people view software. With more devices like hand phone, PDA (personal device assistance), game console, connecting to the network in each day, writing software using machine dependent languages to meet new requirements and expect it to run and support a wide variety of platform and hardware might no longer be possible in the future. Nevertheless, we still have to co-exist with the existing languages in the mean time. Thus

we have built wrappers, ActiveX wrapper and native C library wrapper, to allow other languages to access our agent as well.

B. JINI TECHNOLOGY

Jini Connection technology provides a simple mechanism for objects (clients or services) to discover, join and detach with one another. This makes it very attractive to us in terms of designing and managing processes running in a distributed environment. Our agent service can easily locate and replace services (such as JINI/transaction service) that its processes are using but have become unavailable because of system failure or network congestion, and processes can still find the agent service even if we change its network location.

Another reason that has affected our decision is that many services such as JavaSpace service, transaction service, leasing service and etc are already available. It would have taken us a lot of efforts if we were to build similar functions from scratch. JavaSpace service has given us a Linda TupleSpace Model type of communication mechanism that we are looking for our loosely coupled processes. Although there are similar implementations like IBM Tspace Cloudscape's Java database, which are also written in Java programming language, we preferred JavaSpace because of its scalability. We could create new services to supplement any features that it does not have.

C. FUTURE WORKS

Our current implementation is only the first step towards fulfilling our agent framework. We have focused more on designing and building an architecture that is

scalable and robust. Although we have addressed many of the difficult issues caused by the distributed environment, solved the problems of inter-processes communications and managed a dynamic network, there are still many other areas in our framework where improvements can be made on. Some of them are listed as below.

Enhance Agent Service – The features provided by our agent service are currently quite limited. It only provides simple authentication and coordination mechanism for controlling process using its service. Other features that can be added are

- Provide an GUI to manage and monitor processes running in the network
- Provide connection to backend server like email, database, rule-base engine and etc

Point-to-Point Communication Service – The Linda TupleSpace type of communication is a great way of sharing information in a distributed environment if there is no strict constraint on response latency. From the experiments that we have carried out every entry operation, e.g. read an entry from repository, carried a latency penalty of eight milliseconds compare to less than a millisecond for point-to-point communication using sockets. Point-to-Point communication is also more suitable for sending large amounts of data between two processes.

Security Issues – Security is one of the areas we did not spend much time on. One of the reasons is that we are waiting for the JINI Community to finish their design on how they are going to incorporate the security mechanism into their infrastructure. Security will be a very important issue if processes are connected to the Internet.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION

This thesis presented a simple agent-based framework to address the concerns for building distributed applications. Agents act as the interfaces for processes to interact and to cooperate in a distributed heterogeneous environment. It shields developers from the underlying dynamic and complex network environment by encapsulating the implementation details in the agents and providing a simple set of Application Program Interfaces (API), where processes writing in variety of programming languages that can be easily invoked. The thesis discussed in detail the characteristics of the software agents including agent service, agent operations, agent attributes, and agent language wrappers. Software agents provide a wide range of primitive data types, as well as the feasibility of user-defined formats. Concerns in a distributed environment, such as partial failure, synchronization and coordination, have been taken into consideration in the proposed agent framework through various agent operations. The agent attributes, by communication leasing time, time-out and transaction, explore the possibility of providing time constraints for distributed system communications over the network. It enriches the research of distributed system prototyping. The language wrapper concept makes the proposed framework feasible in most of language platforms, which achieves the interoperability among heterogeneous software components. Test-bed applications and the example demonstrate the important features and show the feasibility of the proposed technology and methodology.

The framework is built on JINI infrastructure to simplify the tasks of building and maintaining reliable distributed systems. It uses a Linda TupleSpace model, a shared network-accessible repository, for different processes to exchange information.

Processes are loosely coupled. They discover and linkup with one another by using services residing on JINI infrastructure. Based on the JINI structure, the agent service provides an interface of developing other features in building a distributed system such as security management.

This framework is to be used in the Distributed Computer Aided Prototyping System (DCAPS) [LUQI92] to provide the inter-process communication layer. The agent framework provides the glue library in the wrapper/glue architecture for distributed system prototyping. It simplifies the tasks of designing, binding and analyzing multiple processes of real-time, distributed systems. Introducing time constraints for network communications into the distributed system design is an initial effort in the research of distributed system prototyping.

LIST OF REFERENCES

- [CLOUD00] Cloudscape, *Cloudscape Java database*, <http://www.cloudscape.com>
- [DCOM96] Microsoft Corporation, *DCOM Technical Overview*, Nov.1996.
http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomtec.htm
- [GEL85] D.Gelernter, *Generative Communication in Linda*, ACM Trans. Programming Languages and Systems, 7(1), Jan.1985, pp. 80-112
- [JOY99] Bill Joy, *The Jini Specifications*, Addison Wesley, Inc., 1999
- [JVM96] Lindholm, T. and Yellin, F., *The Java Virtual Machine Specification*, Addison-Wesley, ISBN 0-201-63451-1
- [JS99] E. Freeman, S. Hupfer and K. Arnold, *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, 1999
- [LUQI92] Luqi, "Computer-aided prototyping for comand and control system using CAPS", IEEE Software, 9(1), Jan. 1992, pp 56-67
- [KEI99] Edward Keith, *Core Jini*, Prentice, Gall PTR, 1999
- [MEESON97] Reginald N. Meeson, *Analysis of Secure Wrapping Technologies* (Alexandria, VA: Institute for Defense Analysis).
- [ORB91] The Object Management Group, *Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.12.1(1991)
- [ROBERT98] Robert Holton , *Real Time Systems*,
<http://www.comp.brad.ac.uk/home/computing/Modules/CM0506D/Courseware/week11/node1.html>
- [RMI00] Sun Microsystems, *Remote Method Invocation. Java 2 SDK Documentation*, Dec 2000. <http://java.sun.com/products/jdk/1.2/docs>
- [TSPACES00] IBM, *Tspaces*. <http://www.almaden.ibm.com/cs/TSpaces>

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. AGENT API

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE CLASS AGENT

java.lang.Object

|

+--tuplesource.core.Agent

public class **Agent**

extends java.lang.Object

implements java.io.Serializable, tuplesource.entries.SpaceActionHandler, [TSConstants](#)

The Agent class implements the methods to configure agent properties, establish connection with Jini Services, request for transaction and create new entry handlers

See Also:

[Serialized Form](#)

Constructor Summary

[Agent\(\)](#)

[Agent](#)(java.lang.String propertiesFilename)

Method Summary

boolean [abortTransaction\(\)](#)

Abort the current transaction; all the commands issued with this transaction after the start transaction will be rolled back.

void [actionPerformed](#)(java.awt.event.ActionEvent e)

void [addActionListener](#)(java.awt.event.ActionListener l)

boolean [cleanTSClass](#)(int type, java.lang.String id)

Creates a new TSString ID

boolean	<u>closeTransaction()</u> closes the current transaction; all the commands issued with transaction after the start transaction will become active.
boolean	<u>createID(int type, java.lang.String id)</u> Create a new entry in agent
void	<u>fireAction(int entryType, java.lang.String entryID)</u>
tuplespace.entries.SpaceActionHandler	<u>getActionHandler()</u> Return the spaceAction handle
java.lang.String	<u>getAgentLookupGroup()</u> Return Agent Lookup Group
java.lang.String	<u>getAgentLookupURL()</u> Return Agent Lookup URL
java.lang.String	<u>getAgentSecurityPolicy()</u> Return Agent Security Policy path and filename
java.lang.String	<u>getAgentServerCodebase()</u> Return Agent Server Codebase
java.lang.String	<u>getAgentSpaceName()</u> Return Agent Space Name
tuplespace.core.Transaction	<u>getTransaction()</u> Return the transaction handle; transaction manager must be initialized and started
java.lang.String	<u>getTSClassIDs(int type)</u> Return all TSString IDs created
java.lang.Object	<u>getTSObject(int type, java.lang.String id)</u> Remove an existing TSString ID
int	<u>getTSType(java.lang.String type)</u>
boolean	<u>InitAgent(long timeout)</u> Look for the Jini Services and test whether the interface is functioning
boolean	<u>isTransactionStarted()</u> Return the current state of transaction handle
void	<u>print(java.lang.String str)</u>

void	<u>readAgentProperties</u> (java.lang.String filename)
void	<u>removeActionListener</u> (java.awt.event.ActionListener l)
boolean	<u>removeID</u> (int type, java.lang.String id) Remove an existing TSString ID
void	<u>SearchTSClassIDs</u> ()
void	<u>setAgentLookupGroup</u> (java.lang.String str) Set Agent Lookup Group
void	<u>setAgentLookupURL</u> (java.lang.String str) Set Agent Lookup URL
void	<u>setAgentSecurityPolicy</u> (java.lang.String str) Set Agent Security Policy path and filename
void	<u>setAgentServerCodebase</u> (java.lang.String str) Set Agent Server Codebase
void	<u>setAgentSpaceName</u> (java.lang.String str) Set Agent Space Name
boolean	<u>startTransaction</u> () Start transaction manager; transaction handle will remain valid for a maximum of 5 minutes (Default setting)
void	<u>TerminateAgent</u> ()
boolean	<u>updateTransactionHandle</u> (int type, java.lang.String id) Update the transaction handle in TSObject

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#)[NEXT CLASS](#)[FRAMES](#)[NO FRAMES](#)[SUMMARY: INNER | FIELD | CONSTR | METHOD](#)[DETAIL: FIELD | CONSTR | METHOD](#)

TUPLESOURCE.CORE

CLASS SERVICEFINDER

tuplesource.core.ServiceFinder

public class **ServiceFinder**

A LookupFinder implements the methods needed locate a service in a Jini(tm) Lookup service.

Constructor Summary

[ServiceFinder\(\)](#)

Create a new LookupFinder object

Method Summary

java.lang.Object [ind](#)(com.sun.jini.mahout.Locator locator, java.lang.String name)

Using the Jini lookup service returned by locator find the service registered with a net.jini.lookup.entry.Name attribute who's value is name.

TUPLESOURCE.CORE

CLASS SERVICEACCESSOR

java.lang.Object

|

+--tuplesource.core.ServiceAccessor

```

public class ServiceAccessor
    extends java.lang.Object
    implements java.io.Serializable
    
```

The ServiceAccessor class implements the methods for registering the Services

See Also:

[Serialized Form](#)

Constructor Summary

[ServiceAccessor\(\)](#)

Method Summary

static com.sun.jini.mahout.Locator	getLocator (long lookupTimeout)
static net.jini.core.transaction.server. TransactionManager	getManager ()
static net.jini.core.transaction.server. TransactionManager	getManager (java.lang.String name)
static net.jini.space.JavaSpace	getSpace ()
static net.jini.space.JavaSpace	getSpace (java.lang.String name)

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)[SUMMARY: INNER | FIELD | CONSTR | METHOD](#)[DETAIL: FIELD | CONSTR | METHOD](#)

TUPLESPEC.CORE

INTERFACE SPACEEVENTREGISTRATION

All Known Implementing Classes:

[TSBoolean](#), [TSClass](#), [TSDouble](#), [TSFloat](#), [TSHash](#), [TSInteger](#), [TSLong](#), [TSQueue](#), [TSString](#)

public abstract interface **SpaceEventRegistration**

The SpaceEventRegistration class implements the methods for for registering with Space Service to monitor new entries written into the space.

Method Summary

boolean	startEvent()
boolean	startEvent1 (tuplespace.entries.SpaceActionHandler spaceAction)
boolean	startEvent2 (tuplespace.entries.SpaceActionHandler spaceAction, long lease)
boolean	startEvent3 (tuplespace.entries.SpaceActionHandler spaceAction, long lease, tuplespace.core.Transaction txn)
boolean	stopEvent()

[Package](#) [Class](#) [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLES.PACE.CORE

INTERFACE SPACELISTENER

public abstract interface **SpaceListener**
extends java.awt.event.ActionListener

Method Summary

void	<u>actionPerformed</u> (java.awt.event.ActionEvent e)
void	<u>fireAction</u> ()

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE

INTERFACE TSCONSTANTS

All Known Implementing Classes:

[Agent](#), [TSClass](#), [TSDouble](#), [TSFloat](#), [TSHash](#), [TSInteger](#), [TSLong](#), [TSQueue](#), [TSBoolean](#), [TSString](#)

public abstract interface **TSConstants**

Field Summary	
static int	TS_BOOLEAN
static int	TS_DOUBLE
static int	TS_FLOAT
static int	TS_HASH
static int	TS_INTEGER
static int	TS_LINKLIST
static int	TS_LONG
static int	TS_QUEUE
static int	TS_STACK
static int	TS_STRING

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#)[NEXT CLASS](#)[FRAMES](#)[NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE

CLASS TSOURCE

java.lang.Object

|

+--tupleSource.core.TSource

Direct Known Subclasses:

[TSBoolean](#), [TSClass](#), [TSDouble](#), [TSFloat](#), [TSHash](#), [TSInteger](#), [TSLong](#),
[TSQueue](#), [TSString](#)

```
public class TSource
extends java.lang.Object
implements java.io.Serializable
```

The TSource class implements the methods for setting the attribute of entry. Every entry handler will inherit this class

See Also:

[Serialized Form](#)

Constructor Summary

[TSource\(\)](#)

Method Summary

long	getReadTimeOut() get read time out
boolean	getResult()
long	getTakeTimeOut() get take time out
tupleSource.core. Transaction	getTransactionHandler()

long	<u>getUpdateLeaseTime()</u> get update lease time
long	<u>getWriteLeaseTime()</u> get write lease time
void	<u>initTSBase</u> (net.jini.space.JavaSpace space, tuplespace.entries.SpaceActionHandler spaceAction, java.lang.String entryID)
protected void	<u>print</u> (java.lang.String str) For debugging purposes
void	<u>setJavaSpace</u> (net.jini.space.JavaSpace space) set space
void	<u>setReadTimeOut</u> (long timeOut) set read time out
void	<u>setResult</u> (boolean value)
void	<u>setSpaceActionHandler</u> (tuplespace.entries.SpaceActionHandler spaceAction) set action listener
void	<u>setTakeTimeOut</u> (long timeOut) set take time out
void	<u>setTransactionHandler</u> (tuplespace.core.Transaction trans) set action listener
void	<u>setUpdateLeaseTime</u> (long leaseTime) set update lease time
void	<u>setWriteLeaseTime</u> (long leaseTime) set write lease time

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE

CLASS TSBOOLEAN

java.lang.Object

|

+--[tuplesource.core.TSBase](#)

|

+--tuplesource.core.TSBoolean

public class **TSBoolean**extends [TSBase](#)implements java.io.Serializable, [SpaceEventRegistration](#), [TSConstants](#)

The TSBoolean class implements the methods for reading, writing, updating, notifying and retrieving EntryBoolean entry from space. Every EntryBoolean entry in the space is identified by an unique ID (entryID). A subclass that implements the SpaceActionHandler interface has to load during initialization if remote event notification is used.

See Also:

EntryBoolean, [Serialized Form](#)

Fields inherited from class tuplesource.core.TSBase

[entryID](#), [eventRegistration](#), [notifvLeaseTime](#), [readTimeOut](#), [result](#), [space](#), [spaceAction](#), [takeTimeOut](#), [transaction](#), [updateLeaseTime](#), [writeLeaseTime](#)

Constructor Summary

[TSBoolean](#)(net.jini.space.JavaSpace space,
tuplesource.entries.SpaceActionHandler spaceAction, java.lang.String entryID)
Constructor for TSBoolean

Method Summary

void [cleanSpace\(\)](#)

Remove all existing entries from space

void	<u>cleanSpace1</u> (tuplespace.core.Transaction txn) Remove all existing entries from space
boolean	<u>initTSBoolean</u> (net.jini.space.JavaSpace space, tuplespace.entries.SpaceActionHandler spaceAction, java.lang.String entryID) initialize TSBoolean
boolean	<u>readIfExists</u> () Reads an entry value
boolean	<u>readIfExists1</u> (long timeOut) Reads an entry value
boolean	<u>readIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Reads an entry value
boolean	<u>startEvent</u> () Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent1</u> (tuplespace.entries.SpaceActionHandler spaceAction) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent2</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent3</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease, tuplespace.core.Transaction txn) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>stopEvent</u> () Stop notification
boolean	<u>takeIfExists</u> () Takes an entry value; entry will be removed from space
boolean	<u>takeIfExists1</u> (long timeOut) Takes an entry value; entry will be removed from space
boolean	<u>takeIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Takes an entry value; entry will be removed from space
boolean	<u>transRead</u> (long timeOut) Reads an entry value with transaction

boolean	<u>transTake</u> (long timeOut) Takes an entry value with transaction; entry will be removed from space
boolean	<u>transUpdate</u> (boolean value, long lease) Updates an existing entry value with transaction
boolean	<u>transWrite</u> (boolean value, long lease) Writes entry value with transaction
boolean	<u>update1</u> (boolean value) Updates an existing entry value
boolean	<u>update2</u> (boolean value, long lease) Updates an existing entry value
boolean	<u>update3</u> (boolean value, long lease, tuplespace.core.Transaction txn) Updates an existing entry value
boolean	<u>write1</u> (boolean value) Writes value to space
boolean	<u>write2</u> (boolean value, long lease) Writes value to space
boolean	<u>write3</u> (boolean value, long lease, tuplespace.core.Transaction txn) Writes value to space

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE

CLASS TSDOUBLE

java.lang.Object

|

+--[tuplesource.core.TSBase](#)

|

+--tuplesource.core.TSDouble

public class **TSDouble**extends [TSBase](#)implements java.io.Serializable, [SpaceEventRegistration](#), [TSConstants](#)

The TSDouble class implements the methods for reading, writing, updating, notifying and retrieving EntryDouble entry from space. Every EntryDouble entry in the space is identified by a unique ID (entryID). A subclass that implements the SpaceActionHandler interface has to load during initialization if remote event notification is used.

See Also:[EntryDouble](#), [Serialized Form](#)

Fields inherited from class tuplesource.core.TSBase

[entryID](#), [eventRegistration](#), [notifyLeaseTime](#), [readTimeOut](#), [result](#), [space](#), [spaceAction](#), [takeTimeOut](#), [transaction](#), [updateLeaseTime](#), [writeLeaseTime](#)

Constructor Summary

[TSDouble](#)([net.jini.space.JavaSpace](#) space,
[tuplesource.entries.SpaceActionHandler](#) spaceAction, java.lang.String entryID)
Constructor for TSDouble

Method Summary

void	cleanSpace() Remove all existing entries from space
void	cleanSpace1 (tuplesource.core.Transaction txn)

	Remove all existing entries from space
boolean	<u>initTSDouble</u> (net.jini.space.JavaSpace space, tuplespace.entries.SpaceActionHandler spaceAction, java.lang.String entryID) initialize TSDouble
double	<u>readIfExists</u> () Reads an entry value
double	<u>readIfExists1</u> (long timeOut) Reads an entry value
double	<u>readIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Reads an entry value
boolean	<u>startEvent</u> () Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent1</u> (tuplespace.entries.SpaceActionHandler spaceAction) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent2</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent3</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease, tuplespace.core.Transaction txn) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>stopEvent</u> () Stop notification
double	<u>takeIfExists</u> () Takes an entry value; entry will be removed from space
double	<u>takeIfExists1</u> (long timeOut) Takes an entry value; entry will be removed from space
double	<u>takeIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Takes an entry value; entry will be removed from space
double	<u>transRead</u> (long timeOut) Reads an entry value with transaction
double	<u>transTake</u> (long timeOut) Takes an entry value with transaction: entry will be removed from

	space
boolean	<u>transUpdate</u> (double value, long lease) Updates an existing entry value with transaction
boolean	<u>transWrite</u> (double value, long lease) Writes entry value with transaction
boolean	<u>update1</u> (double value) Updates an existing entry value
boolean	<u>update2</u> (double value, long lease) Updates an existing entry value
boolean	<u>update3</u> (double value, long lease, tuplespace.core.Transaction txn) Updates an existing entry value
boolean	<u>write1</u> (double value) Writes value to space
boolean	<u>write2</u> (double value, long lease) Writes value to space
boolean	<u>write3</u> (double value, long lease, tuplespace.core.Transaction txn) Writes value to space

[Package](#)[Class](#)[Tree](#)[Index](#)[Help](#)[PREV CLASS](#)[NEXT CLASS](#)[FRAMES](#)[NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

TUPLESOURCE.CORE

CLASS TSHASH

java.lang.Object

|

+--[tuplespace.core.TSBase](#)

|

+--[tuplespace.core.TSHash](#)public class **TSHash**extends [TSBase](#)implements java.io.Serializable, [SpaceEventRegistration](#), [TSConstants](#)

The TSHash class implements the methods for reading, writing, updating, notifying and retrieving EntryHash entry from space. Every EntryHash entry in the space is identified by an unique ID (entryID). A subclass that implements the SpaceActionHandler interface has to load during initialization if remote event notification is used.

See Also:EntryHash, [Serialized Form](#)

Fields inherited from class tuplespace.core.[TSBase](#)

[entryID](#), [eventRegistration](#), [notifvLeaseTime](#), [readTimeOut](#), [result](#), [space](#), [spaceAction](#), [takeTimeOut](#), [transaction](#), [updateLeaseTime](#), [writeLeaseTime](#)

Constructor Summary

[TSHash](#)([net.jini.space.JavaSpace](#) space, [tuplespace.entries.SpaceActionHandler](#) spaceAction, [java.lang.String](#) entryID)
Constructor for TSHash

Method Summary

void [cleanSpace\(\)](#)

Remove all existing entries from space

void [cleanSpace1](#)([tuplespace.core.Transaction](#) txn)

	Remove all existing entries from space
boolean	<u>clearContainer</u> ()
boolean	<u>getBoolean</u> (java.lang.String id)
java.util.HashMap	<u>getContainer</u> ()
void	<u>getContainer</u> (java.util.HashMap map)
double	<u>getDouble</u> (java.lang.String id)
float	<u>getFloat</u> (java.lang.String id)
int	<u>getInteger</u> (java.lang.String id)
long	<u>getLong</u> (java.lang.String id)
java.lang.String	<u>getString</u> (java.lang.String id)
boolean	<u>initTSHash</u> (net.jini.space.JavaSpace space, tuplespace.entries.SpaceActionHandler spaceAction, java.lang.String entryID) initialize TSHash
boolean	<u>readIfExists</u> () Read entry value
boolean	<u>readIfExists1</u> (long timeOut) Read entry value
boolean	<u>readIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Read entry value
boolean	<u>remove</u> (java.lang.String id)
boolean	<u>setBoolean</u> (java.lang.String id, boolean value)

boolean	<u>setDouble</u> (java.lang.String id, double value)
boolean	<u>setFloat</u> (java.lang.String id, float value)
boolean	<u>setInteger</u> (java.lang.String id, int value)
boolean	<u>setLong</u> (java.lang.String id, long value)
boolean	<u>setString</u> (java.lang.String id, java.lang.String content)
boolean	<u>startEvent</u> () Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent1</u> (tuplespace.entries.SpaceActionHandler spaceAction) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent2</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>startEvent3</u> (tuplespace.entries.SpaceActionHandler spaceAction, long lease, tuplespace.core.Transaction txn) Start notification; remote event will be raised if any entry that matches the entry ID is added into the space.
boolean	<u>stopEvent</u> () Stop notification
boolean	<u>takeIfExists</u> () Take entry value; entry will be removed from space
boolean	<u>takeIfExists1</u> (long timeOut) Take entry value; entry will be removed from space
boolean	<u>takeIfExists2</u> (long timeOut, tuplespace.core.Transaction txn) Take entry value; entry will be removed from space
boolean	<u>transRead</u> (long timeOut) Reads an entry value with transaction

boolean	<u>transTake</u> (long timeOut) Takes an entry value with transaction; entry will be removed from space
boolean	<u>transUpdate</u> (long lease) Updates an existing entry value with transaction
boolean	<u>transWrite</u> (long lease) Writes entry value with transaction
boolean	<u>update</u> () Updates an existing entry value
boolean	<u>update1</u> (long lease) Updates an existing entry value
boolean	<u>update2</u> (long lease, tuplespace.core.Transaction txn) Updates an existing entry value
boolean	<u>write</u> () Writes value to space
boolean	<u>write1</u> (long lease) Writes value to space
boolean	<u>write2</u> (long lease, tuplespace.core.Transaction txn) Writes value to space

APPENDIX B. JINI/SERVICES SETUP SCRIPT

StartService	
File	
LookupBrowser	TxnManager
Template	Run
JavaSpace	RMID
AgentService	WebServer
LookupService	
Java Command :	java
Java Options :	-cp
Setup JVM Options :	-Djava.library.path=h:\jini1_1\lib\jini-core.jar;h:\jini1_1\lib\jini-ext.jar;h:\jini1_1\lib\jini-sun-util.jar;h:\jini1_1\lib\jini-rt.jar
Security Policy File :	-Djava.security.policy=h:\jini1_1\policy\policy.all
Codebase :	-Djava.rmi.server.codebase=http://MELPOMENE:8081/
Executable :	tuplespace.services.AgentService

```
#StartService
#Mon Sep 18 17:00:03 EDT 2000
#

service.list=RMID
    WebServer
    LookupService
    LookupBrowser
    TxnManager
    JavaSpace
    AgentService

#
# RMID
#
RMID.label1=RMID Command :
RMID.label2=Options :
RMID.option1=rmid
RMID.option2=-J-Dsun.rmi.activation.execPolicy\=none

#
# WebServer
#
WebServer.label1=Java Command :
WebServer.label2=Java Options :
WebServer.label3=Executable Jar File :
WebServer.label4=Port :
WebServer.label5=Document Area :
WebServer.label6=Log Downloads :

WebServer.option1=java
WebServer.option2=-jar
WebServer.option3=d:\jini1_1\lib\tools.jar
WebServer.option4=-port 8081
WebServer.option5=-dir d:\jini1_1\lib
WebServer.option6=
```

```

#
# Reggie - Lookup Service
#
LookupService.label1=Java Command :
LookupService.label2=Java Options :
LookupService.label3=Setup JVM Options :
LookupService.label4=Executable Jar File :
LookupService.label5=Codebase :
LookupService.label6=Security Policy File :
LookupService.label7=Log Directory :
LookupService.label8=Groups :
LookupService.label9=Server JVM :
LookupService.label10=Server JVM Arguments :

LookupService.option1=java
LookupService.option2=-jar
LookupService.option3=-Djava.security.policy=d:\\jini1_1\\policy\\policy.all
LookupService.option4=d:\\jini1_1\\lib\\reggie.jar
LookupService.option5=http://tiptop:8081/reggie-dl.jar
LookupService.option6=d:\\jini1_1\\policy\\policy.all
LookupService.option7=\\tmp\\reggie_log
LookupService.option8=public
LookupService.option9=
LookupService.option10=

#
# Lookup Browser
#
LookupBrowser.label1=Java Command :
LookupBrowser.label2=Java Options :
LookupBrowser.label3=Jar File :
LookupBrowser.label4=Security Policy File :
LookupBrowser.label5=Codebase :
LookupBrowser.label6=Lookup Browser :
LookupBrowser.label7=Admin Mode :
LookupBrowser.label8=Groups :

LookupBrowser.option1=java
LookupBrowser.option2=-cp
LookupBrowser.option3=d:\\jini1_1\\lib\\jini-examples.jar
LookupBrowser.option4=-Djava.security.policy=d:\\jini1_1\\example\\browser\\policy
LookupBrowser.option5=-Djava.rmi.server.codebase=http://tiptop:8081/jini-examples-dl.jar
LookupBrowser.option6=com.sun.jini.example.browser.Browser
LookupBrowser.option7=
LookupBrowser.option8=

#
# Mahalo - TxnManager
#
TxnManager.label1=Java Command :
TxnManager.label2=Java Options :
TxnManager.label3=Setup JVM Options :
TxnManager.label4=Executable Jar File :
TxnManager.label5=Codebase :
TxnManager.label6=Security Policy File :
TxnManager.label7=Log Directory :
TxnManager.label8=Groups and Locators :
TxnManager.label9=Server JVM :
TxnManager.label10=Server JVM Arguments :

TxnManager.option1=java
TxnManager.option2=-jar
TxnManager.option3=
TxnManager.option4=d:\\jini1_1\\lib\\mahalo.jar
TxnManager.option5=http://tiptop:8081/mahalo-dl.jar
TxnManager.option6=d:\\jini1_1\\policy\\policy.all
TxnManager.option7=d:\\tmp\\mahalo_log
TxnManager.option8=public
TxnManager.option9=

```

```

TxnManager.option10=

#
# JavaSpace
#
JavaSpace.option1=java
JavaSpace.option2=-jar
JavaSpace.option3=-Djava.security.policy=d:\\jini1_1\\policy\\policy.all
JavaSpace.option4=-Djava.rmi.server.codebase=http://tiptop:8081/outrigger-dl.jar
JavaSpace.option5=-Dcom.sun.jini.outrigger.spaceName=JavaSpaces
JavaSpace.option6=d:\\jini1_1\\lib\\transient-outrigger.jar
JavaSpace.option7=public
JavaSpace.option8=

JavaSpace.label1=Java Command :
JavaSpace.label2=Java Options :
JavaSpace.label3=Java Security File :
JavaSpace.label4=Codebase :
JavaSpace.label5=JavaSpace Name :
JavaSpace.label6=Executable Jar File :
JavaSpace.label7=Groups :
JavaSpace.label8=Locators :

#
# AgentService
#
AgentService.label1=Java Command :
AgentService.label2=Java Options :
AgentService.label3=Setup JVM Options :
AgentService.label4=Security Policy File :
AgentService.label5=Codebase :
AgentService.label6=Executable :

AgentService.option1=java
AgentService.option2=-cp
AgentService.option3=D:\\Jini1_1\\lib\\jini-core.jar;D:\\Jini1_1\\lib\\jini-
ext.jar;D:\\Jini1_1\\lib\\sun-util.jar;d:\\
AgentService.option4=-Djava.security.policy=d:\\jini1_1\\policy\\policy.all
#AgentService.option4=-Djava.rmi.server.codebase=http://tiptop:8081/
AgentService.option5=-Djava.rmi.server.codebase=http://tiptop:8081/
AgentService.option6=tuplespace.services.AgentService

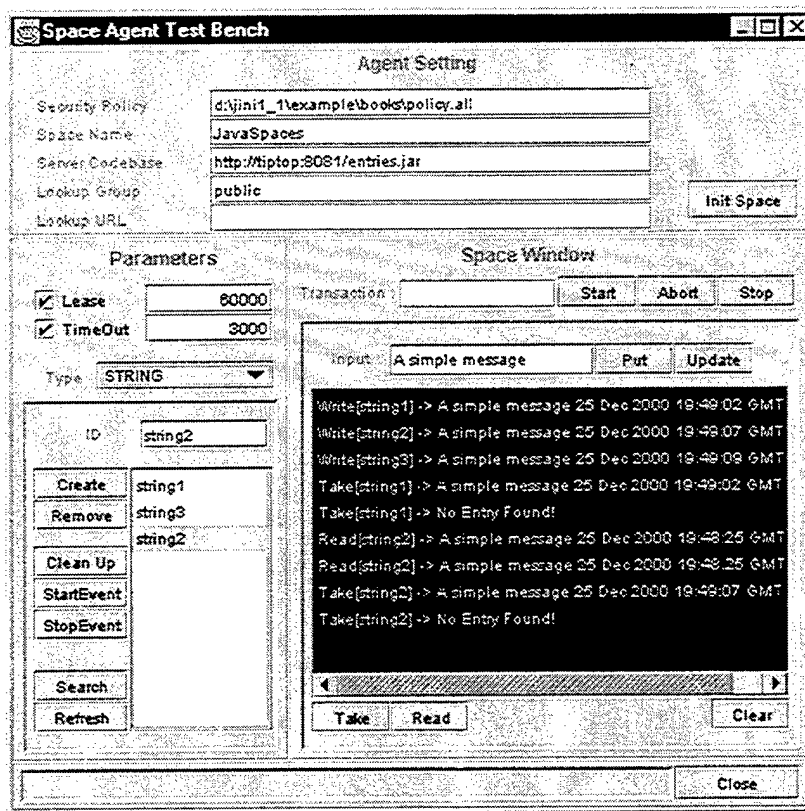
#
# END
#

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. AGENT TEST BENCH LISTING

A. JAVA VERSION



1. JavaTestBench.java

```
package tuplespace.core;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.util.Vector;
import com.thwt.layout.*;

/**
 * The <code> JavaTestBench</code> class implements the GUI for testing
 * the agent interface.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */
public class JavaTestBench extends JApplet {
    boolean isStandalone = false;
    String[] tupleTypes = { "STRING", "BOOLEAN", "INTEGER", "FLOAT", "LONG", "DOUBLE", "HASH",
"QUEUE", "STACK" };
    PerformActions perform;
    Vector vectorID = new Vector();
    Vector vectorOutput = new Vector();

    JPanel jPanel1 = new JPanel();
    SmartLayout smartLayout1 = new SmartLayout();
    JButton cmdClose = new JButton();
    SmartLayout smartLayout2 = new SmartLayout();
    JLabel txtStatus = new JLabel();
```

```

JPanel jPanel2 = new JPanel();
JLabel jLabelC1 = new JLabel();
JLabel jLabelC2 = new JLabel();
JScrollPane listOutputScroll = new JScrollPane();
JButton cmdInitSpace = new JButton();
JButton cmdStopEvent = new JButton();
JTextField txtID = new JTextField();
JList listID = new JList();
SmartLayout smartLayoutLeft = new SmartLayout();
JTextField txtLookupURL = new JTextField();
JTextField txtTimeout = new JTextField();
JTextField txtLease = new JTextField();
JButton cmdRead = new JButton();
JPanel jPanelTop = new JPanel();
JCheckBox checkTimeout = new JCheckBox();
JTextField txtLookupGroup = new JTextField();
JButton cmdTake = new JButton();
JButton cmdTxnAbort = new JButton();
JTextField txtSpaceName = new JTextField();
JButton cmdClean = new JButton();
JButton cmdClear = new JButton();
JCheckBox checkLease = new JCheckBox();
JTextField txtInput = new JTextField();
SmartLayout smartLayoutMain = new SmartLayout();
SmartLayout smartLayoutRight1 = new SmartLayout();
JButton cmdTxnClose = new JButton();
JPanel jPanelLeft = new JPanel();
SmartLayout smartLayoutLeft1 = new SmartLayout();
JButton cmdRefresh = new JButton();
SmartLayout smartLayoutRight = new SmartLayout();
JPanel jPanelRight1 = new JPanel();
JButton cmdStartEvent = new JButton();
JComboBox cbType = new JComboBox(tupleTypes);
JLabel jLabelA0 = new JLabel();
JLabel jLabelA1 = new JLabel();
JScrollPane listIDScroll = new JScrollPane();
JLabel jLabelA2 = new JLabel();
JLabel jLabelA3 = new JLabel();
JLabel jLabelA4 = new JLabel();
JLabel jLabelA5 = new JLabel();
SmartLayout smartLayoutTop = new SmartLayout();
JTextField txtCodebase = new JTextField();
JLabel jLabelB0 = new JLabel();
JLabel jLabelB1 = new JLabel();
JLabel jLabelB3 = new JLabel();
JList listOutput = new JList();
JTextField txtTransaction = new JTextField();
JButton cmdPut = new JButton();
JPanel jPanelMain = new JPanel();
JPanel jPanelLeft1 = new JPanel();
JTextField txtSecurityPolicy = new JTextField();
JButton cmdRemove = new JButton();
JPanel jPanelRight = new JPanel();
JButton cmdCreate = new JButton();
JButton cmdTxnStart = new JButton();
JLabel jLabelC0 = new JLabel();
JButton cmdSearch = new JButton();
JButton cmdUpdate = new JButton();

//Construct the applet
public JavaTestBench () {
}

//Initialize the JavaTestBench
public void init() {
    try {
        jbInit();
        listID.setListData(vectorID);
        listOutput.setListData(vectorOutput);
    }
}

```

```

        perform = new PerformActions(this);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

//Component initialization
private void jbInit() throws Exception {
    this.setSize(new Dimension(518, 488));
    jPanel1.setLayout(smartLayout1);
    cmdClose.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdClose_actionPerformed(e);
        }
    });
    cmdClose.setText("Close");
    cmdClose.setFont(new java.awt.Font("Dialog", 0, 10));
    txtStatus.setFont(new java.awt.Font("Dialog", 0, 11));
    txtStatus.setBorder(BorderFactory.createLoweredBevelBorder());
    jPanel2.setBorder(BorderFactory.createEtchedBorder());
    jPanel2.setLayout(smartLayout2);
    jLabelC1.setFont(new java.awt.Font("SansSerif", 0, 10));
    jLabelC1.setText("Transaction :");
    jLabelC2.setText("Input");
    jLabelC2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabelC2.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdInitSpace.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdInitSpace_actionPerformed(e);
        }
    });
    cmdInitSpace.setText("Init Space");
    cmdInitSpace.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdInitSpace.setMargin(new Insets(2, 2, 2, 2));
    cmdStopEvent.setMargin(new Insets(2, 2, 2, 2));
    cmdStopEvent.setEnabled(false);
    cmdStopEvent.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdStopEvent.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdStopEvent_actionPerformed(e);
        }
    });
    cmdStopEvent.setText("StopEvent");
    txtID.setFont(new java.awt.Font("Dialog", 0, 10));
    txtID.setText("EntryID");
    listID.addMouseListener(new java.awt.event.MouseAdapter() {

        public void mouseClicked(MouseEvent e) {
            listID_mouseClicked(e);
        }
    });
    listID.setBorder(BorderFactory.createEtchedBorder());
    listID.setFont(new java.awt.Font("Dialog", 0, 10));
    txtLookupURL.setText("");
    txtLookupURL.setFont(new java.awt.Font("Dialog", 0, 10));
    txtTimeout.setText("3000");
    txtTimeout.setHorizontalAlignment(SwingConstants.RIGHT);
    txtTimeout.setFont(new java.awt.Font("Dialog", 0, 10));
    txtLease.setFont(new java.awt.Font("Dialog", 0, 10));
    txtLease.setText("60000");
    txtLease.setHorizontalAlignment(SwingConstants.RIGHT);
    cmdRead.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdRead_actionPerformed(e);
        }
    });

```

```

    }
    });
    cmdRead.setText("Read");
    cmdRead.setEnabled(false);
    cmdRead.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdRead.setMargin(new Insets(2, 2, 2, 2));
    jPanelTop.setBorder(BorderFactory.createEtchedBorder());
    jPanelTop.setLayout(smartLayoutTop);
    checkTimeout.setFont(new java.awt.Font("SansSerif", 0, 10));
    checkTimeout.setSelected(true);
    checkTimeout.setText("TimeOut");
    txtLookupGroup.setFont(new java.awt.Font("Dialog", 0, 10));
    txtLookupGroup.setText("");
    cmdTake.setText("Take");
    cmdTake.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdTake_actionPerformed(e);
        }

    });
    cmdTake.setEnabled(false);
    cmdTake.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdTake.setMargin(new Insets(2, 2, 2, 2));
    cmdTxnAbort.setText("Abort");
    cmdTxnAbort.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdTxnAbort_actionPerformed(e);
        }

    });
    cmdTxnAbort.setEnabled(false);
    cmdTxnAbort.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdTxnAbort.setMargin(new Insets(2, 2, 2, 2));
    txtSpaceName.setFont(new java.awt.Font("Dialog", 0, 10));
    txtSpaceName.setText("");
    cmdClean.setText("Clean Up");
    cmdClean.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdClean_actionPerformed(e);
        }

    });
    cmdClean.setEnabled(false);
    cmdClean.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdClean.setMargin(new Insets(2, 2, 2, 2));
    cmdClear.setMargin(new Insets(2, 2, 2, 2));
    cmdClear.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdClear.setText("Clear");
    cmdClear.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdClear_actionPerformed(e);
        }

    });
    checkLease.setSelected(true);
    checkLease.setText("Lease");
    checkLease.setFont(new java.awt.Font("SansSerif", 0, 10));
    txtInput.setFont(new java.awt.Font("Dialog", 0, 10));
    txtInput.setText("A simple message");
    cmdTxnClose.setText("End");
    cmdTxnClose.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(ActionEvent e) {
            cmdTxnClose_actionPerformed(e);
        }

    });
    cmdTxnClose.setEnabled(false);
    cmdTxnClose.setFont(new java.awt.Font("SansSerif", 0, 10));
    cmdTxnClose.setActionCommand("Close");

```

```

cmdTxnClose.setMargin(new Insets(2, 2, 2, 2));
jPanelLeft.setBorder(BorderFactory.createEtchedBorder());
jPanelLeft.setLayout(smartLayoutLeft);
cmdRefresh.setText("Refresh");
cmdRefresh.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdRefresh_actionPerformed(e);
    }

});
cmdRefresh.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdRefresh.setMargin(new Insets(2, 2, 2, 2));
jPanelRight1.setBorder(BorderFactory.createLoweredBevelBorder());
jPanelRight1.setLayout(smartLayoutRight1);
cmdStartEvent.setText("StartEvent");
cmdStartEvent.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdStartEvent_actionPerformed(e);
    }

});
cmdStartEvent.setEnabled(false);
cmdStartEvent.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdStartEvent.setMargin(new Insets(2, 2, 2, 2));
cbType.setFont(new java.awt.Font("Dialog", 1, 10));
cbType.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cbType_actionPerformed(e);
    }

});
jLabelA0.setText("Agent Setting");
jLabelA0.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelA1.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelA1.setText("Security Policy");
jLabelA2.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelA2.setText("Space Name");
jLabelA3.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelA3.setText("Server Codebase");
jLabelA4.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelA4.setText("Lookup Group");
jLabelA5.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelA5.setText("Lookup URL");
txtCodebase.setFont(new java.awt.Font("Dialog", 0, 10));
txtCodebase.setText("");
jLabelB0.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelB0.setHorizontalAlignment(SwingConstants.CENTER);
jLabelB0.setText("Parameters ");
jLabelB1.setText("Type");
jLabelB1.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelB3.setText("ID");
jLabelB3.setFont(new java.awt.Font("SansSerif", 0, 10));
jLabelB3.setHorizontalAlignment(SwingConstants.CENTER);
listOutput.setBackground(Color.black);
listOutput.setFont(new java.awt.Font("Dialog", 0, 10));
listOutput.setForeground(Color.green);
listOutput.setBorder(BorderFactory.createEtchedBorder());
txtTransaction.setText("");
txtTransaction.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdPut.setText("Put");
cmdPut.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdPut_actionPerformed(e);
    }

});
cmdPut.setEnabled(false);
cmdPut.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdPut.setMargin(new Insets(2, 2, 2, 2));

```

```

jPanelMain.setLayout(smartLayoutMain);
jPanelLeft1.setBorder(BorderFactory.createLoweredBevelBorder());
jPanelLeft1.setLayout(smartLayoutLeft1);
txtSecurityPolicy.setFont(new java.awt.Font("Dialog", 0, 10));
txtSecurityPolicy.setText("");
cmdRemove.setText("Remove");
cmdRemove.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdRemove_actionPerformed(e);
    }

});
cmdRemove.setEnabled(false);
cmdRemove.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdRemove.setMargin(new Insets(2, 2, 2, 2));
jPanelRight.setBorder(BorderFactory.createEtchedBorder());
jPanelRight.setLayout(smartLayoutRight);
cmdCreate.setText("Create");
cmdCreate.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdCreate_actionPerformed(e);
    }

});
cmdCreate.setEnabled(false);
cmdCreate.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdCreate.setMargin(new Insets(2, 2, 2, 2));
cmdTxnStart.setMargin(new Insets(2, 2, 2, 2));
cmdTxnStart.setBackground(new java.awt.Color(204, 204, 204));
cmdTxnStart.setEnabled(false);
cmdTxnStart.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdTxnStart.setText("Start");
cmdTxnStart.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdTxnStart_actionPerformed(e);
    }

});
jLabelC0.setText("Space Window");
jLabelC0.setHorizontalAlignment(SwingConstants.CENTER);
jLabelC0.setFont(new java.awt.Font("Dialog", 1, 12));
cmdSearch.setMargin(new Insets(2, 2, 2, 2));
cmdSearch.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdSearch.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdSearch_actionPerformed(e);
    }

});
cmdSearch.setText("Search");
cmdUpdate.setMargin(new Insets(2, 2, 2, 2));
cmdUpdate.setEnabled(false);
cmdUpdate.setFont(new java.awt.Font("SansSerif", 0, 10));
cmdUpdate.addActionListener(new java.awt.event.ActionListener() {

    public void actionPerformed(ActionEvent e) {
        cmdUpdate_actionPerformed(e);
    }

});
cmdUpdate.setText("Update");
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanelMain, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 505),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 453),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 0),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 0)));
jPanelMain.add(jPanelTop, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 120),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 0),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 0),

```

```

        new com.thwt.layout.EdgeAnchor(jPanelRight, Anchor.Right, Anchor.Same,
Anchor.Right, 0));
jPanelTop.add(jLabelA2, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(jLabelA3, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA2, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(jLabelA4, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA3, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(jLabelA5, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA4, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(txtSecurityPolicy, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 280),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA1, Anchor.Top, Anchor.Same, Anchor.Top,
0)));
jPanelTop.add(txtSpaceName, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA2, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA2, Anchor.Top, Anchor.Same, Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(txtCodebase, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA3, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA3, Anchor.Top, Anchor.Same, Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(txtLookupGroup, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA4, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA4, Anchor.Top, Anchor.Same, Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));

```

```

jPanelTop.add(txtLookupURL, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(jLabelA5, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelA5, Anchor.Top, Anchor.Same, Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(txtSecurityPolicy, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelTop.add(cmdInitSpace, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 70),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 24),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 426),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 85)));
jPanelTop.add(jLabelA1, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 110),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 14),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 26)));
jPanelTop.add(jLabelA0, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 110),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 16),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 217),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 1)));
jPanelMain.add(jPanelLeft, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 175),
    new com.thwt.layout.ContainerAnchor(Anchor.Bottom, 0),
    new com.thwt.layout.EdgeAnchor(jPanelTop, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jPanelTop, Anchor.Left, Anchor.Same, Anchor.Left,
0)));
jPanelLeft.add(checkTimeout, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Bottom, Anchor.Below,
Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelLeft.add(txtLease, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 80),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Right, Anchor.Right,
Anchor.Left, 0),
    new com.thwt.layout.EdgeAnchor(checkLease, Anchor.Top, Anchor.Same, Anchor.Top,
0)));
jPanelLeft.add(txtTimeout, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 80),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.EdgeAnchor(checkTimeout, Anchor.Right, Anchor.Right,
Anchor.Left, 0),
    new com.thwt.layout.EdgeAnchor(checkTimeout, Anchor.Top, Anchor.Same, Anchor.Top,
0)));
jPanelLeft.add(checkLease, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 70),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 13),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 28)));
jPanelLeft.add(jLabelB0, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 110),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 16),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 41),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 3)));
jPanelLeft.add(cbType, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 112),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 52),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 76)));
jPanelLeft.add(jPanelLeft1, new com.thwt.layout.LayoutConstraint(

```

```

        new com.thwt.layout.ContainerAnchor(Anchor.Left, 5),
        new com.thwt.layout.ContainerAnchor(Anchor.Right, 5),
        new com.thwt.layout.ContainerAnchor(Anchor.Top, 100),
        new com.thwt.layout.ContainerAnchor(Anchor.Bottom, 5));
jPanelLeft1.add(cmdRemove, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelLeft1.add(cmdClean, new com.thwt.layout.LayoutConstraint(
new com.thwt.layout.EdgeAnchor(cmdRemove, Anchor.Bottom, Anchor.Below, Anchor.Top, 10),
new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Left, Anchor.Same, Anchor.Left, 0),
new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Width, Anchor.Same, Anchor.Width, 0),
new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Height, Anchor.Same, Anchor.Height,
0)));
jPanelLeft1.add(cmdStartEvent, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelLeft1.add(cmdRefresh, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.ContainerAnchor(Anchor.Bottom, 10),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Left, Anchor.Same, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(cmdCreate, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelLeft1.add(cmdCreate, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 60),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 5),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 41));
jPanelLeft1.add(jLabelB3, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 60),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 16),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 13),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 11));
jPanelLeft1.add(txtID, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 80),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.EdgeAnchor(jLabelB3, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(jLabelB3, Anchor.Top, Anchor.Same, Anchor.Top,
0)));
jPanelLeft1.add(listIDScroll, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 90),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 66),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 41),
    new com.thwt.layout.ContainerAnchor(Anchor.Bottom, 10));
jPanelLeft1.add(cmdStopEvent, new com.thwt.layout.LayoutConstraint(
new com.thwt.layout.EdgeAnchor(cmdStartEvent, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Left, Anchor.Same, Anchor.Left, 0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Width, Anchor.Same, Anchor.Width, 0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Height, Anchor.Same, Anchor.Height, 0)));
jPanelLeft1.add(cmdSearch, new com.thwt.layout.LayoutConstraint(
new com.thwt.layout.EdgeAnchor(cmdRefresh, Anchor.Top, Anchor.Above, Anchor.Bottom, 0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Left, Anchor.Same, Anchor.Left, 0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Width, Anchor.Same, Anchor.Width, 0),
new com.thwt.layout.EdgeAnchor(cmdClean, Anchor.Height, Anchor.Same, Anchor.Height, 0)));

```

```

jPanelLeft.add(jLabelB1, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 70),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 16),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 19),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 77)));
jPanelMain.add(jPanelRight, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 330),
    new com.thwt.layout.EdgeAnchor(jPanelLeft, Anchor.Right, Anchor.Right,
Anchor.Left, 0),
    new com.thwt.layout.EdgeAnchor(jPanelTop, Anchor.Bottom, Anchor.Below, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(jPanelLeft, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelRight.add(cmdTxnAbort, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(cmdTxnStart, Anchor.Right, Anchor.Right,
Anchor.Left, 0),
    new com.thwt.layout.EdgeAnchor(cmdTxnStart, Anchor.Top, Anchor.Same, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(cmdTxnStart, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(cmdTxnStart, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelRight.add(cmdTxnClose, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(cmdTxnAbort, Anchor.Right, Anchor.Right, Anchor.Left, 0),
    new com.thwt.layout.EdgeAnchor(cmdTxnAbort, Anchor.Top, Anchor.Same, Anchor.Top,
0),
    new com.thwt.layout.EdgeAnchor(cmdTxnAbort, Anchor.Width, Anchor.Same,
Anchor.Width, 0),
    new com.thwt.layout.EdgeAnchor(cmdTxnAbort, Anchor.Height, Anchor.Same,
Anchor.Height, 0)));
jPanelRight.add(jLabelC0, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 85),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 17),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 108),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 2)));
jPanelRight.add(txtTransaction, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 100),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 18),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 68),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 26)));
jPanelRight.add(jLabelC1, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 110),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 16),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 4),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 25)));
jPanelRight.add(cmdTxnStart, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 50),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 169),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 24)));
jPanelRight.add(jPanelRight1, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 5),
    new com.thwt.layout.ContainerAnchor(Anchor.Right, 5),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 51),
    new com.thwt.layout.ContainerAnchor(Anchor.Bottom, 5)));
jPanelRight1.add(cmdPut, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 50),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.EdgeAnchor(txtInput, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(txtInput, Anchor.Top, Anchor.Same, Anchor.Top,
0)));
jPanelRight1.add(jLabelC2, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 110),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 18),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 11)));
jPanelRight1.add(txtInput, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 130),

```

```

        new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
        new com.thwt.layout.ContainerAnchor(Anchor.Left, 55),
        new com.thwt.layout.ContainerAnchor(Anchor.Top, 13));
jPanelRight1.add(cmdRead, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 50),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.EdgeAnchor(cmdTake, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(cmdTake, Anchor.Top, Anchor.Same, Anchor.Top, 0));
jPanelRight1.add(listOutputScroll, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 303),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 198),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 5),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 40));
jPanelRight1.add(cmdTake, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 50),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 240),
    new com.thwt.layout.EdgeAnchor(listOutputScroll, Anchor.Left, Anchor.Same,
Anchor.Left, 0));
jPanelRight1.add(cmdClear, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 50),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 20),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 258),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 239));
jPanelRight1.add(cmdUpdate, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.EdgeAnchor(cmdPut, Anchor.Right, Anchor.Right, Anchor.Left,
0),
    new com.thwt.layout.EdgeAnchor(cmdPut, Anchor.Top, Anchor.Same, Anchor.Top, 0),
    new com.thwt.layout.EdgeAnchor(cmdClear, Anchor.Width, Anchor.Same, Anchor.Width,
0),
    new com.thwt.layout.EdgeAnchor(cmdClear, Anchor.Height, Anchor.Same,
Anchor.Height, 0));
jPanel1.add(jPanel2, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 505),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 31),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 0),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 453));
jPanel2.add(cmdClose, new com.thwt.layout.LayoutConstraint(
new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 79),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 22),
    new com.thwt.layout.ContainerAnchor(Anchor.Right, 5),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 2));
jPanel2.add(txtStatus, new com.thwt.layout.LayoutConstraint(
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Width, 413),
    new com.thwt.layout.FixedDimensionAnchor(Anchor.Height, 21),
    new com.thwt.layout.ContainerAnchor(Anchor.Left, 3),
    new com.thwt.layout.ContainerAnchor(Anchor.Top, 2));
listOutputScroll.getViewPort().add(listOutput, null);
listIDScroll.getViewPort().add(listID, null);
}

//Start the applet
public void start() {
}

//Stop the applet
public void stop() {
}

//Destroy the applet
public void destroy() {
}

//Get Applet information
public String getAppletInfo() {
    return "Applet Information";
}

```

```

//Get parameter info
public String[][] getParameterInfo() {
    return null;
}

//Main method
public static void main(String[] args) {
    Applet1 applet = new JavaTestBench();
    applet.isStandalone = true;
    JFrame frame = new JFrame();
    frame.setTitle("Space Agent Test Bench");
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.setSize(515,510);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) / 2, (d.height -
frame.getSize().height) / 2);
    frame.setVisible(true);
}

// static initializer for setting look & feel
static {
    try {
        UIManager.setLookAndFeel( "com.sun.java.swing.plaf.windows.MetalLookAndFeel");
        //UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        //UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception e) {}
}

void cmdTxnStart_actionPerformed(ActionEvent e) {
    cmdPut.setBackground(Color.red);
    cmdUpdate.setBackground(Color.red);
    cmdRead.setBackground(Color.red);
    cmdTake.setBackground(Color.red);
    cmdStartEvent.setBackground(Color.red);
    txtTransaction.setText("Running...");
    txtStatus.setText(" Start Transaction Manager ->" + perform.startTransaction());
}

void cmdTxnAbort_actionPerformed(ActionEvent e) {
    cmdPut.setBackground(new Color(204,204,204));
    cmdUpdate.setBackground(new Color(204,204,204));
    cmdRead.setBackground(new Color(204,204,204));
    cmdTake.setBackground(new Color(204,204,204));
    cmdStartEvent.setBackground(new Color(204,204,204));
    txtTransaction.setText("");
    txtStatus.setText(" Abort Transaction Manager ->" + perform.abortTransaction());
}

void cmdTxnClose_actionPerformed(ActionEvent e) {
    cmdPut.setBackground(new Color(204,204,204));
    cmdUpdate.setBackground(new Color(204,204,204));
    cmdRead.setBackground(new Color(204,204,204));
    cmdTake.setBackground(new Color(204,204,204));
    cmdStartEvent.setBackground(new Color(204,204,204));
    txtTransaction.setText("");
    txtStatus.setText(" Close Transaction Manager ->" + perform.closeTransaction());
}

void cmdInitSpace_actionPerformed(ActionEvent e) {
    perform.initSpace();
}

void cmdCreate_actionPerformed(ActionEvent e) {
    perform.createID((String) cbType.getSelectedItem());
}

void cmdRemove_actionPerformed(ActionEvent e) {

```

```

        perform.removeID((String) cbType.getSelectedItem());
    }

    void cmdClean_actionPerformed(ActionEvent e) {
        perform.cleanSpace((String) cbType.getSelectedItem());
    }

    void cmdStartEvent_actionPerformed(ActionEvent e) {
        perform.startEvent((String) cbType.getSelectedItem());
    }
    void cmdStopEvent_actionPerformed(ActionEvent e) {
        perform.stopEvent((String) cbType.getSelectedItem());
    }

    void cmdRefresh_actionPerformed(ActionEvent e) {
        perform.refreshListIDs((String) cbType.getSelectedItem());
    }

    void cmdTake_actionPerformed(ActionEvent e) {
        perform.takeEntry((String) cbType.getSelectedItem());
    }

    void cmdRead_actionPerformed(ActionEvent e) {
        perform.readEntry((String) cbType.getSelectedItem());
    }

    void cmdClear_actionPerformed(ActionEvent e) {
        perform.clearOutput();
    }

    void cmdClose_actionPerformed(ActionEvent e) {
        perform.close();
        System.exit(0);
    }

    void cmdPut_actionPerformed(ActionEvent e) {
        perform.putEntry((String) cbType.getSelectedItem());
    }

    void listID_mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 1) {
            int index = listID.locationToIndex(e.getPoint());
            if(index != -1)
                perform.IDSelected(index);
        }
    }

    void listID_mousePressed(MouseEvent e) { }
    void listID_mouseReleased(MouseEvent e) { }
    void listID_mouseEntered(MouseEvent e) { }
    void listID_mouseExited(MouseEvent e) { }

    void cbType_actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        perform.changeType( (String) cbType.getSelectedItem());
    }

    void printClassName(Object obj) {
        System.out.println("The class of " + obj +
            " is " + obj.getClass().getName());
    }

    void cmdSearch_actionPerformed(ActionEvent e) {
        perform.SearchTSClassIDs();
    }

    void cmdUpdate_actionPerformed(ActionEvent e) {
        perform.updateEntry((String) cbType.getSelectedItem());
    }

```

)

2. PerformActions.java

```
package tuplespace.core;

import java.util.Date;
import java.util.Vector;
import java.lang.StringBuffer;
import javax.swing.JOptionPane;
import javax.swing.JScrollBar;
import java.util.EventListener;
import java.awt.event.*;
import java.awt.Cursor;

/**
 * The <code>PerformActions</code> class handle the events for
 * the GUI.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */
public class PerformActions implements ActionListener{
    private Applet1 frame;
    private Agent agent;
    public PerformActions(Applet1 frame) {
        this.frame = frame;
        this.agent = new Agent();
        this.agent.addActionListener(this);
        refreshAll();
    }

    public void initSpace(){
        frame.getGlassPane().addMouseListener( new MouseAdapter() {});
        frame.getGlassPane().setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        frame.getGlassPane().setVisible(true);

        agent.setAgentSecurityPolicy(frame.txtSecurityPolicy.getText());
        agent.setAgentSpaceName(frame.txtSpaceName.getText());
        agent.setAgentServerCodebase(frame.txtCodebase.getText());
        agent.setAgentLookupGroup(frame.txtLookupGroup.getText());
        if (frame.txtLookupURL.getText().length() > 7)
            agent.setAgentLookupURL(frame.txtLookupURL.getText());
        //agent.setAgentLookupURL(frame.txtLookupURL.getText());
        if ( agent.InitAgent(10000) ){
            print("Space Services Initialised.");
            frame.cmdCreate.setEnabled(true);
            frame.cmdRefresh.setEnabled(true);
            frame.cmdRemove.setEnabled(true);
        }else{
            print("Fail! to initialise services, timeout 20 secs");
        }
        frame.getGlassPane().setVisible(false);
    }

    public void createID(String type){
        String tmpStr = frame.txtID.getText();
        print("Create      "+      type      +      "["      +      tmpStr+      "]"      "      +
agent.createID(TSType2Int(type),tmpStr));
        refreshListIDs(type);
        frame.cmdPut.setEnabled(true);
        frame.cmdUpdate.setEnabled(true);
        frame.cmdRead.setEnabled(true);
        frame.cmdTake.setEnabled(true);
        frame.cmdStartEvent.setEnabled(true);
        frame.cmdStopEvent.setEnabled(true);
        frame.cmdTxnAbort.setEnabled(true);
        frame.cmdTxnClose.setEnabled(true);
        frame.cmdTxnStart.setEnabled(true);
    }
}
```

```

    }

    public boolean startTransaction(){
        return agent.startTransaction();
    }

    public boolean closeTransaction(){
        return agent.closeTransaction();
    }

    public boolean abortTransaction(){
        return agent.abortTransaction();
    }

    public void close(){
        agent.TerminateAgent();
    }

    public void removeID(String type){
        String tmpStr = frame.txtID.getText();
        print("Remove      +      type      +      "["      +      tmpStr+      "]"      "      +
agent.removeID(agent.getTSType(type), tmpStr));
        refreshListIDs(type);
    }

    public void refreshIDs(){
        //print(" String Map Keys -> " + agent.getTSStringMapKeys());
        //refreshListIDs();
    }

    public void cleanSpace(String type){
        String tmpStr = frame.txtID.getText();
        print("Clean      +      type      +      "["      +      tmpStr+      "]"      "      +
agent.cleanTSClass(TSType2Int(type), tmpStr));
    }

    boolean putEntry(String type) {
        if( type.compareTo("BOOLEAN") == 0){
            TSBoolean      ts      =      (TSBoolean)      agent.getTSObject(
agent.TS_BOOLEAN, frame.txtID.getText());
            if(ts != null){
                try{
                    boolean tmp = Boolean.valueOf(frame.txtInput.getText()).booleanValue();
                    print("Write["+ frame.txtID.getText()+"] -> " + tmp +
                        "      : " + ts.write3(tmp, getLeaseTime(), agent.getTransaction()));
                }catch(Exception e){}
            }
        }else if( type.compareTo("INTEGER") == 0){
            TSInteger      ts      =      (TSInteger)      agent.getTSObject(
agent.TS_INTEGER, frame.txtID.getText());
            if(ts != null){
                try{
                    int tmp = Integer.valueOf(frame.txtInput.getText()).intValue();
                    print("Write["+ frame.txtID.getText()+"] -> " + tmp +
                        "      : " + ts.write3(tmp, getLeaseTime(), agent.getTransaction()));
                }catch(Exception e){}
            }
        }
        else if( type.compareTo("FLOAT") == 0){
            TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT, frame.txtID.getText());
            if(ts != null){
                try{
                    float tmp = Float.valueOf(frame.txtInput.getText()).floatValue();
                    print("Write["+ frame.txtID.getText()+"] -> " + tmp +
                        "      : " + ts.write3(tmp, getLeaseTime(), agent.getTransaction()));
                }catch(Exception e){}
            }
        }
    }
}

```

```

else if( type.compareTo("LONG") == 0){
    TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG, frame.txtID.getText());
    if(ts != null){
        try{
            long tmp = Long.valueOf(frame.txtInput.getText()).longValue();
            print("Write["+ frame.txtID.getText()+"] -> " + tmp +
                " : " + ts.write3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("DOUBLE") == 0){
    TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE, frame.txtID.getText());
    if(ts != null){
        try{
            double tmp = Double.valueOf(frame.txtInput.getText()).doubleValue();
            print("Write["+ frame.txtID.getText()+"] -> " + tmp +
                " : " + ts.write3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("STRING") == 0){
    Date time = new Date();
    String tmpStr = frame.txtInput.getText(); //+ " " + time.toGMTString();
    TSString ts = (TSString) agent.getTSObject(agent.TS_STRING,
frame.txtID.getText());
    if(ts != null){
        print("Write["+ frame.txtID.getText()+"] -> " + tmpStr +
            " : " + ts.write3(tmpStr, getLeaseTime(), agent.getTransaction()));
    }
}
else if( type.compareTo("QUEUE") == 0){
    Date time = new Date();
    String tmpStr = frame.txtInput.getText()+ " " + time.toGMTString();
    TSQueue ts = (TSQueue) agent.getTSObject(agent.TS_QUEUE, frame.txtID.getText());
    if(ts != null){
        print("Write["+ frame.txtID.getText()+"] -> " + tmpStr + " " +
ts.writel(tmpStr));
    }
}
else if( type.compareTo("STACK") == 0){
}
else if( type.compareTo("LINKLIST") == 0){
}
else if( type.compareTo("HASH") == 0){
    Date time = new Date();
    TSHash tsHash = (TSHash) agent.getTSObject(agent.TS_HASH,
frame.txtID.getText());
    if(tsHash != null){
        tsHash.setString("a", "String a "+time.toGMTString());
        tsHash.setInteger("b", 1234);
        tsHash.setFloat("c", (float) 1234.1234);
        tsHash.setLong("d", 123456789);
        tsHash.setDouble("e", 123456789.123456789);
        tsHash.setBoolean("f", true);
        print("Write["+ frame.txtID.getText()+"] -> " +
tsHash.writel(getLeaseTime()));
    }
}
return true;
}

void updateEntry(String type) {
    if( type.compareTo("BOOLEAN") == 0){
        TSBoolean ts = (TSBoolean) agent.getTSObject(
agent.TS_BOOLEAN, frame.txtID.getText());
        if(ts != null){

```

```

        try{
            boolean tmp = Boolean.valueOf(frame.txtInput.getText()).booleanValue();
            print("Update["+ frame.txtID.getText()+"] -> " + tmp +
                "      : " + ts.update3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("INTEGER") == 0){
    TSInteger ts = (TSInteger) agent.getTSObject(
agent.TS_INTEGER, frame.txtID.getText());
    if(ts != null){
        try{
            int tmp = Integer.valueOf(frame.txtInput.getText()).intValue();
            print("Update["+ frame.txtID.getText()+"] -> " + tmp +
                "      : " + ts.update3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("FLOAT") == 0){
    TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT, frame.txtID.getText());
    if(ts != null){
        try{
            float tmp = Float.valueOf(frame.txtInput.getText()).floatValue();
            print("Update["+ frame.txtID.getText()+"] -> " + tmp +
                "      : " + ts.update3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("LONG") == 0){
    TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG, frame.txtID.getText());
    if(ts != null){
        try{
            long tmp = Long.valueOf(frame.txtInput.getText()).longValue();
            print("Update["+ frame.txtID.getText()+"] -> " + tmp +
                "      : " + ts.update3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("DOUBLE") == 0){
    TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE, frame.txtID.getText());
    if(ts != null){
        try{
            double tmp = Double.valueOf(frame.txtInput.getText()).doubleValue();
            print("Update["+ frame.txtID.getText()+"] -> " + tmp +
                "      : " + ts.update3(tmp, getLeaseTime(), agent.getTransaction()));
        }catch(Exception e){}
    }
}
else if( type.compareTo("STRING") == 0){
    Date time = new Date();
    String tmpStr = frame.txtInput.getText(); //+ " " + time.toGMTString();
    TSString ts = (TSString) agent.getTSObject(agent.TS_STRING,
frame.txtID.getText());
    if(ts != null){
        print("Update["+ frame.txtID.getText()+"] -> " + tmpStr +
            "      : " + ts.update3(tmpStr, getLeaseTime(), agent.getTransaction()));
    }
}
else if( type.compareTo("QUEUE") == 0){
    /*Date time = new Date();
    String tmpStr = frame.txtInput.getText()+ " " + time.toGMTString();
    TSQueue ts = (TSQueue) agent.getTSObject(agent.TS_QUEUE, frame.txtID.getText());
    if(ts != null){
        print("Update["+ frame.txtID.getText()+"] -> " + tmpStr + " " +
ts.update1(tmpStr));
    } */
}
else if( type.compareTo("STACK") == 0){
}
}

```

```

else if( type.compareTo("LINKLIST") == 0){
}
else if( type.compareTo("HASH") == 0){
    Date time = new Date();
    TSHash      tsHash      =      (TSHash)      agent.getTSObject(agent.TS_HASH,
frame.txtID.getText());
    if(tsHash != null){
        tsHash.setString("a","String a "+time.toGMTString());
        tsHash.setInteger("b",1234);
        tsHash.setFloat("c",(float) 1234.1234);
        tsHash.setLong("d",123456789);
        tsHash.setDouble("e",123456789.123456789);
        tsHash.setBoolean("f",true);
        print("Update["+      frame.txtID.getText()+"      ]      ->      "      +
tsHash.update2(getLeaseTime(),agent.getTransaction()));
    }
}
}

```

```

boolean takeEntry(String type) {
    if( type.compareTo("BOOLEAN") == 0){
        TSBoolean      ts      =      (TSBoolean)      agent.getTSObject(
agent.TS_BOOLEAN,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("INTEGER") == 0){
        TSInteger      ts      =      (TSInteger)      agent.getTSObject(
agent.TS_INTEGER,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("FLOAT") == 0){
        TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("LONG") == 0){
        TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("DOUBLE") == 0){
        TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("STRING") == 0){
        TSString ts = (TSString) agent.getTSObject( agent.TS_STRING,frame.txtID.getText());
        if(ts != null){
            print("Take["+      frame.txtID.getText()+"      ]      ->      "      +
ts.takeIfExists2(getTimeout(),agent.getTransaction()));
        }
    }
    else if( type.compareTo("QUEUE") == 0){

```

```

    TSQueue ts = (TSQueue) agent.getTSObject( agent.TS_QUEUE, frame.txtID.getText());
    if(ts != null){
        print("Take["+ frame.txtID.getText()+"] -> " + ts.take());
    }
}
else if( type.compareTo("STACK") == 0){
}
else if( type.compareTo("LINKLIST") == 0){
}
else if( type.compareTo("HASH") == 0){
    TSHash tsHash = (TSHash) agent.getTSObject(agent.TS_HASH, frame.txtID.getText());
    if(tsHash != null){
        print("Take["+ frame.txtID.getText()+"] -> " + tsHash.takeIfExists());
        print("tsHash[a] String -> " + tsHash.getString("a"));
        print("tsHash[b] Integer -> " + tsHash.getInteger("b"));
        print("tsHash[c] Float -> " + tsHash.getFloat("c"));
        print("tsHash[d] Long -> " + tsHash.getLong("d"));
        print("tsHash[e] Double -> " + tsHash.getDouble("e"));
        print("tsHash[f] Boolean -> " + tsHash.getBoolean("f"));
    }
}
return true;
}

boolean readEntry(String type) {
    if( type.compareTo("BOOLEAN") == 0){
        TSBoolean ts = (TSBoolean) agent.getTSObject(
agent.TS_BOOLEAN, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
    else if( type.compareTo("INTEGER") == 0){
        TSInteger ts = (TSInteger) agent.getTSObject(
agent.TS_INTEGER, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
    else if( type.compareTo("FLOAT") == 0){
        TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
    else if( type.compareTo("LONG") == 0){
        TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
    else if( type.compareTo("DOUBLE") == 0){
        TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
    else if( type.compareTo("STRING") == 0){
        TSString ts = (TSString) agent.getTSObject( agent.TS_STRING, frame.txtID.getText());
        if(ts != null){
            print("Read["+ frame.txtID.getText()+"] -> " +
ts.readIfExists2(getTimeout(), agent.getTransaction());
        }
    }
}

```

```

    }
}
else if( type.compareTo("QUEUE") == 0){
    TSQueue ts = (TSQueue) agent.getTSObject( agent.TS_QUEUE, frame.txtID.getText());
    if(ts != null){
        ts.printQueue();
        long startIndex, endIndex;
        startIndex = ts.getStartIndex();
        endIndex = ts.getEndIndex();
        clearOutput();
        print("\n *** Stack Contents *** \n");
        for(long i = startIndex; i <= endIndex ; i++){
            print("[+i+] " + ts.readl(i) );
        }
    }
}
else if( type.compareTo("STACK") == 0){

}
else if( type.compareTo("LINKLIST") == 0){

}
else if( type.compareTo("HASH") == 0){
    TSHash tsHash = (TSHash) agent.getTSObject(agent.TS_HASH, frame.txtID.getText());
    if(tsHash != null){
        print("Read["+ frame.txtID.getText()+"] -> " + tsHash.readIfExists());
        print("tsHash[a] String -> " + tsHash.getString("a"));
        print("tsHash[b] Integer -> " + tsHash.getInteger("b"));
        print("tsHash[c] Float -> " + tsHash.getFloat("c"));
        print("tsHash[d] Long -> " + tsHash.getLong("d"));
        print("tsHash[e] Double -> " + tsHash.getDouble("e"));
        print("tsHash[f] Boolean -> " + tsHash.getBoolean("f"));
    }
}
return true;
}

void startEvent(String type) {
    if( type.compareTo("BOOLEAN") == 0){
        TSBoolean ts = (TSBoolean) agent.getTSObject(
agent.TS_BOOLEAN, frame.txtID.getText());
        if(ts != null){
            print("Notify["+ frame.txtID.getText()+"] -> " +
ts.startEvent3(agent, getLeaseTime(), agent.getTransaction()));
        }
    }
    else if( type.compareTo("INTEGER") == 0){
        TSInteger ts = (TSInteger) agent.getTSObject(
agent.TS_INTEGER, frame.txtID.getText());
        if(ts != null){
            print("Notify["+ frame.txtID.getText()+"] -> " +
ts.startEvent3(agent, getLeaseTime(), agent.getTransaction()));
        }
    }
    else if( type.compareTo("FLOAT") == 0){
        TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT, frame.txtID.getText());
        if(ts != null){
            print("Notify["+ frame.txtID.getText()+"] -> " +
ts.startEvent3(agent, getLeaseTime(), agent.getTransaction()));
        }
    }
    else if( type.compareTo("LONG") == 0){
        TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG, frame.txtID.getText());
        if(ts != null){
            print("Notify["+ frame.txtID.getText()+"] -> " +
ts.startEvent3(agent, getLeaseTime(), agent.getTransaction()));
        }
    }
    else if( type.compareTo("DOUBLE") == 0){

```

```

        TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE, frame.txtID.getText());
        if(ts != null){
            print("Notify["+          frame.txtID.getText()+" ]"          ->          "          +
            ts.startEvent3(agent, getLeaseTime(), agent.getTransaction());
        }
    }
    else if( type.compareTo("STRING") == 0){
        TSString ts = (TSString) agent.getTSObject( agent.TS_STRING, frame.txtID.getText());
        if(ts != null)
            print("Notify["+          frame.txtID.getText()+" ]"          ->          "          +
            ts.startEvent3(agent, getLeaseTime(), agent.getTransaction());
        else
            print("Notify["+ frame.txtID.getText()+" ] -> Fail");
    }
    else if( type.compareTo("QUEUE") == 0){
        TSQueue ts = (TSQueue) agent.getTSObject( agent.TS_QUEUE, frame.txtID.getText());
        if(ts != null){
            print("Notify["+          frame.txtID.getText()+" ]"          ->          "          +
            ts.startEvent3(agent, getLeaseTime(), agent.getTransaction());
        }
    }
    else if( type.compareTo("STACK") == 0){
        TSStack ts = (TSStack) agent.getTSObject( agent.TS_STACK, frame.txtID.getText());
        //if(ts != null){
            //          print("Notify["+          frame.txtID.getText()+" ]"          ->          "          +
            ts.startEvent3(agent, getLeaseTime(), agent.getTransaction());
        //}
    }
    else if( type.compareTo("LINKLIST") == 0){
    }
    else if( type.compareTo("HASH") == 0){
        TSHash ts = (TSHash) agent.getTSObject( agent.TS_HASH, frame.txtID.getText());
        if(ts != null){
            print("Notify["+          frame.txtID.getText()+" ]"          ->          "          +
            ts.startEvent3(agent, getLeaseTime(), agent.getTransaction());
        }
    }
}

void stopEvent(String type) {
    if( type.compareTo("BOOLEAN") == 0){
        TSBoolean ts = (TSBoolean) agent.getTSObject(
            agent.TS_BOOLEAN, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+" ] -> " + ts.stopEvent());
        }
    }
    if( type.compareTo("INTEGER") == 0){
        TSInteger ts = (TSInteger) agent.getTSObject(
            agent.TS_INTEGER, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+" ] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("FLOAT") == 0){
        TSFloat ts = (TSFloat) agent.getTSObject( agent.TS_FLOAT, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+" ] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("LONG") == 0){
        TSLong ts = (TSLong) agent.getTSObject( agent.TS_LONG, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+" ] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("DOUBLE") == 0){

```

```

        TSDouble ts = (TSDouble) agent.getTSObject( agent.TS_DOUBLE, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+"] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("STRING") == 0){
        TSString ts = (TSString) agent.getTSObject( agent.TS_STRING, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+"] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("QUEUE") == 0){
        TSQueue ts = (TSQueue) agent.getTSObject( agent.TS_QUEUE, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+"] -> " + ts.stopEvent());
        }
    }
    else if( type.compareTo("STACK") == 0){
    }
    else if( type.compareTo("LINKLIST") == 0){
    }
    else if( type.compareTo("HASH") == 0){
        TSHash ts = (TSHash) agent.getTSObject( agent.TS_HASH, frame.txtID.getText());
        if(ts != null){
            print("Stop Notification["+ frame.txtID.getText()+"] -> " + ts.stopEvent());
        }
    }
}

public void changeType(String type){
    refreshListIDs(type);

    if( type.compareTo("BOOLEAN") == 0){
        frame.txtInput.setText("TRUE");
    }
    if( type.compareTo("INTEGER") == 0){
        frame.txtInput.setText("12345");
    }
    else if( type.compareTo("FLOAT") == 0){
        frame.txtInput.setText("12345.123456");
    }
    else if( type.compareTo("LONG") == 0){
        frame.txtInput.setText("1234567890");
    }
    else if( type.compareTo("DOUBLE") == 0){
        frame.txtInput.setText("1234567890.123456789");
    }
    else if( type.compareTo("STRING") == 0){
        frame.txtInput.setText("Hello agent!");
    }
    else if( type.compareTo("QUEUE") == 0){
    }
    else if( type.compareTo("STACK") == 0){
    }
    else if( type.compareTo("LINKLIST") == 0){
    }
    else if( type.compareTo("HASH") == 0){
    }

    frame.cmdPut.setEnabled(false);
    frame.cmdUpdate.setEnabled(false);
    frame.cmdRead.setEnabled(false);
}

```

```

        frame.cmdTake.setEnabled(false);
        frame.cmdStartEvent.setEnabled(false);
        frame.cmdStopEvent.setEnabled(false);
        frame.cmdTxnAbort.setEnabled(false);
        frame.cmdTxnClose.setEnabled(false);
        frame.cmdTxnStart.setEnabled(false);
    }

    void IDSelected(int index){
        frame.txtID.setText((String) frame.vectorID.get(index));
        frame.cmdPut.setEnabled(true);
        frame.cmdUpdate.setEnabled(true);
        frame.cmdRead.setEnabled(true);
        frame.cmdTake.setEnabled(true);
        frame.cmdStartEvent.setEnabled(true);
        frame.cmdStopEvent.setEnabled(true);
        frame.cmdTxnAbort.setEnabled(true);
        frame.cmdTxnClose.setEnabled(true);
        frame.cmdTxnStart.setEnabled(true);
    }

    void refreshAll(){
        frame.txtSecurityPolicy.setText(agent.getAgentSecurityPolicy());
        frame.txtSpaceName.setText(agent.getAgentSpaceName());
        frame.txtCodebase.setText(agent.getAgentServerCodebase());
        frame.txtLookupGroup.setText(agent.getAgentLookupGroup());
        frame.txtLookupURL.setText(agent.getAgentLookupURL());
    }

    void refreshListIDs(String type){
        int startIndex, endIndex;
        frame.vectorID.clear();
        String tmpStr = agent.getTSClassIDs(TSType2Int(type));
        if( tmpStr != null ){
            tmpStr = tmpStr.concat(",");
            startIndex = 0;
            endIndex = tmpStr.indexOf(',');
            while(endIndex > 0 ){
                frame.vectorID.add(tmpStr.substring(startIndex,endIndex));
                startIndex = endIndex + 1;
                endIndex = tmpStr.indexOf(',',startIndex);
            }
            frame.listID.setListData(frame.vectorID);
        }

    void clearOutput(){
        frame.vectorOutput.clear();
        frame.listOutput.setListData(frame.vectorOutput);
    }

    void printStatus(String str){
        frame.txtStatus.setText(str);
        //print(str);
    }

    void print(String str){
        frame.vectorOutput.add(str);
        frame.listOutput.setListData(frame.vectorOutput);
        JScrollBar vbar = frame.listOutputScroll.getVerticalScrollBar();
        vbar.setValue(vbar.getMaximum());
        //System.out.println(str);
    }

    long getLeaseTime(){
        Long tmpLong;
        if(frame.checkLease.isSelected() == true){

```

```

        try{
            //System.out.println("                Lease                time:" +
Long.decode(frame.txtLease.getText()).longValue());
            return Long.decode(frame.txtLease.getText()).longValue() ;
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    //System.out.println(" Lease time:" + Long.MAX_VALUE);
    return Long.MAX_VALUE;
}

long getTimeout(){
    Long tmpLong;
    if(frame.checkLease.isSelected() == true){
        try{
            //System.out.println("                Time                Out:" +
Long.decode(frame.txtTimeout.getText()).longValue());
            return Long.decode(frame.txtTimeout.getText()).longValue() ;
        }catch(Exception e){
        }
    }
    return Long.MAX_VALUE;
}

public void actionPerformed(ActionEvent e){
    print("... Remote Event Type :[" + TSType2String(e.getID()) + "] ID:[" +
e.getActionCommand() + "] ");
}

private String TSType2String(int type){
    switch(type){
        case 5000:
            return "INTEGER";
        case 5001:
            return "FLOAT";
        case 5002:
            return "LONG";
        case 5003:
            return "DOUBLE";
        case 5004:
            return "STRING";
        case 5005:
            return "QUEUE";
        case 5006:
            return "STACK";
        case 5007:
            return "LINKLIST";
        case 5008:
            return "HASH";
        case 5009:
            return "BOOLEAN";
        default:
            return "UNKNOWN";
    }
}

private int TSType2Int(String type){
    if( type.compareTo("INTEGER") == 0)
        return agent.TS_INTEGER;
    else if( type.compareTo("FLOAT") == 0)
        return agent.TS_FLOAT;
    else if( type.compareTo("LONG") == 0)
        return agent.TS_LONG;
    else if( type.compareTo("DOUBLE") == 0)
        return agent.TS_DOUBLE;
    else if( type.compareTo("STRING") == 0)
        return agent.TS_STRING;
    else if( type.compareTo("QUEUE") == 0)

```

```

        return agent.TS_QUEUE;
    else if( type.compareTo("STACK") == 0)
        return agent.TS_STACK;
    else if( type.compareTo("LINKLIST") == 0)
        return agent.TS_LINKLIST;
    else if( type.compareTo("HASH") == 0)
        return agent.TS_HASH;
    else if( type.compareTo("BOOLEAN") == 0)
        return agent.TS_BOOLEAN;
    else
        return 0;
}

public void SearchTSClassIDs(){
    long time;
    int i,j;
    int cc[];

    cc = new int[500];
    agent.createID (agent.TS_STRING,"entryID");

    for( j = 0; j < 500 ; j++){
        cc[j] = 0;
    }

    for( j = 0; j < 100 ; j++){
        time = System.currentTimeMillis();
        for( i = 0; i < 10 ; i++)
        {
            TSString ts = (TSString) agent.getTSObject(agent.TS_STRING, "entryID");
            if(ts.write1("hello"))
            {
                if( ts.takeIfExists().length () > 0)
                {
                    //System.out.println(          "Latency      time      ->      "      +
System.currentTimeMillis());
                }
            }
        }
        int t = (int)((double)(System.currentTimeMillis() - time)/10.0);
        System.out.println( "Latency time -> " + t);

        cc[t]++;
    }
    for( j = 0; j < 500 ; j++){
        if( cc[j] != 0 ){
            System.out.println( j + " " + cc[j]);
        }
    }
    agent.removeID (agent.TS_STRING,"entryID");
}
}

```

B. VISUAL BASIC VERSION

Space Agent Test Bench

Agent Setting

Security Policy: d:\ini1_1\example\books\policy.all
 Space Name: JavaSpaces
 Server Codebase: http://tiptop:8081/entries.jar
 Lookup Group: public
 Lookup URL: Reinitialize

Parameters Setting

Lease: ☐ 10000 msec
 Time out: ☐ 10000 msec

Space Memory

Transaction: Not started yet Start Abort End

Input: A Simple Message Put Update

Create ID[TSQueue] -> True
 Write[TSQueue] -> A Simple Message : False
 Create ID[TSQueue] -> True
 Write[TSQueue] -> A Simple Message : True
 Read[TSQueue] -> A Simple Message
 Update[TSQueue] -> A Simple Message : True
 Update[TSQueue] -> A Simple Message : True
 Update[TSQueue] -> A Simple Message : True
 Update[TSQueue] -> A Simple Message : True
 Read[TSQueue] -> A Simple Message

Take Read Clear

Close

1. VBTestBench.vb

```
Option Explicit
Private tmp As String

Private Sub cbType_Click()
  Select Case cbType.Text
    Case "STRING"
      txtMsg.Text = "A Simple Message"
    Case "INTEGER"
      txtMsg.Text = "1234"
    Case "FLOAT"
      txtMsg.Text = "1234.1234"
    Case "LONG"
      txtMsg.Text = "123456789"
    Case "DOUBLE"
      txtMsg.Text = "123456789.12345678"
    Case "BOOLEAN"
      txtMsg.Text = "true"
  End Select
  cmdPut.Enabled = False
  cmdUpdate.Enabled = False
  cmdRead.Enabled = False
  cmdTake.Enabled = False
  cmdStartNotify.Enabled = False
  cmdStopNotify.Enabled = False
  txnStart.Enabled = False
  txnAbort.Enabled = False
  txnClose.Enabled = False

  Call cmdShowID_Click
End Sub
```

```

End Sub

Private Sub cmdClear_Click()
    ListOutput.Clear
End Sub

Private Sub cmdClose_Click()
    'Agent1.TerminateAgent
    Unload Me
End Sub

Private Sub cmdCreateID_Click()
    dump "Create ID[ " & txtID & "]" -> " & Agent1.createID(Agent1.getTSType(cbType.Text),
txtID)
    Call cmdShowID_Click
    ListID.ListIndex = ListID.ListCount - 1
    cmdPut.Enabled = True
    cmdUpdate.Enabled = True
    cmdRead.Enabled = True
    cmdTake.Enabled = True
    cmdStartNotify.Enabled = True
    cmdStopNotify.Enabled = True
    txnStart.Enabled = True
    txnAbort.Enabled = True
    txnClose.Enabled = True
End Sub

Private Sub cmdReinitialize_Click()
    Screen.MousePointer = vbHourglass
    If Agent1.InitAgent(5000) Then
        dump "Services Initialized!"
        cmdCreateID.Enabled = True
        cmdShowID.Enabled = True
        cmdRemoveID.Enabled = True
    Else
        dump "Fail! to initialize services"
        cmdCreateID.Enabled = False
        cmdShowID.Enabled = False
        cmdRemoveID.Enabled = False
    End If
    Screen.MousePointer = vbDefault
End Sub

Private Sub cmdRemoveID_Click()
    dump "Remove ID[" & txtID & "]" -> " & Agent1.removeID(Agent1.getTSType(cbType.Text),
txtID)
    Call cmdShowID_Click
    ListID.ListIndex = ListID.ListCount - 1
End Sub

Private Sub cmdShowID_Click()
    Dim tmp As String, tmp1 As String
    tmp = Agent1.getTSClassIDs(Agent1.getTSType(cbType.Text))

    Dim startPos As Long
    Dim endPos As Long
    tmp = Trim(tmp) & ","
    startPos = 1
    ListID.Clear
    While startPos < Len(tmp)
        endPos = InStr(startPos, tmp, ",", vbTextCompare)
        If endPos > 0 Then
            tmp1 = LTrim(RTrim(Mid(tmp, startPos, endPos - startPos)))
            ListID.AddItem (tmp1)
        End If
        startPos = endPos + 1
    Wend
End Sub

```

```

Private Sub cmdPut_Click()
    Dim writeStr As String
    Dim ret As Boolean

    On Error Resume Next
    writeStr = txtMsg.Text()
    ret = False

    Select Case cbType.Text
        Case "STRING"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(writeStr, getLeaseTime(), Agent1.getTransaction())
        Case "INTEGER"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(CInt(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "FLOAT"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(CDbl(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "LONG"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(CLng(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "DOUBLE"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(CDbl(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "BOOLEAN"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).write3(CBool(writeStr), getLeaseTime(), Agent1.getTransaction())
    End Select

    dump "Write[" & txtID & "] ->" & writeStr & " : " & ret
End Sub

```

```

Private Sub cmdUpdate_Click()
    Dim writeStr As String
    Dim ret As Boolean

    On Error Resume Next
    writeStr = txtMsg.Text()
    ret = False

    Select Case cbType.Text
        Case "STRING"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(writeStr, getLeaseTime(), Agent1.getTransaction())
        Case "INTEGER"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(CInt(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "FLOAT"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(CDbl(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "LONG"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(CLng(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "DOUBLE"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(CDbl(writeStr), getLeaseTime(), Agent1.getTransaction())
        Case "BOOLEAN"
            ret = Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).update3(CBool(writeStr), getLeaseTime(), Agent1.getTransaction())
    End Select

    dump "Update[" & txtID & "] ->" & writeStr & " : " & ret
End Sub

```

```

Private Sub cmdTake_Click()

```

```

        dump "Take[" & txtID & "]" -> " & Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).takeIfExists2(getTimeOut(), Agent1.getTransaction())
    End Sub

Private Sub cmdRead_Click()
    dump "Read[" & txtID & "]" -> " & Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).readIfExists2(getTimeOut(), Agent1.getTransaction())
End Sub

Private Sub cmdStartNotify_Click()
    ' On Error Resume Next
    dump "Notify ID[" & txtID & "]" -> " &
Agent1.getTSObject(Agent1.getTSType(cbType.Text),
txtID).startEvent3(Agent1.getActionHandler, getLeaseTime(), Agent1.getTransaction())
End Sub

Private Sub cmdStopNotify_Click()
    On Error Resume Next
    dump "Notify ID[" & txtID & "]" -> " &
Agent1.getTSObject(Agent1.getTSType(cbType.Text), txtID).stopEvent()
End Sub

'*****
' actionPerformed
'*****

Private Sub Agent1_actionPerformed(ByVal ActionEvent1 As Object)
    dump "... Remote Event[" & ActionEvent1.getActionCommand() & "]" Type[" &
ActionEvent1.getID() & "]"
End Sub

Private Sub dump(str As String)
    ListOutput.AddItem (str)
    ListOutput.ListIndex = ListOutput.ListCount - 1
End Sub

'*****
' Form Methods
'*****
Private Sub Form_Load()
    txtSecurityPolicy = Agent1.getAgentSecurityPolicy
    txtSpaceName = Agent1.getAgentSpaceName
    txtLookupGroup = Agent1.getAgentLookupGroup
    txtServerCodebase = Agent1.getAgentServerCodebase
    txtLookupURL = Agent1.getAgentLookupURL
    ListID.Clear
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Agent1.TerminateAgent
End Sub

Private Sub ListID_Click()
    txtID = ListID.Text
    cmdPut.Enabled = True
    cmdUpdate.Enabled = True
    cmdRead.Enabled = True
    cmdTake.Enabled = True
    cmdStartNotify.Enabled = True
    cmdStopNotify.Enabled = True
    txnStart.Enabled = True
    txnAbort.Enabled = True
    txnClose.Enabled = True
End Sub

```

```

' *****
' User defined functions
' *****

Private Function getLeaseTime() As Long
    If CheckLease.Value = 1 Then
        getLeaseTime = CLng(TxtLease.Text)
    Else
        getLeaseTime = &HFFFFFFF
    End If
End Function

Private Function getTimeOut() As Long
    If CheckTimeOut.Value = 1 Then
        getTimeOut = CLng(txtTimeOut.Text)
    Else
        getTimeOut = &HFFFFFFF
    End If
End Function

' *****
' Transaction
' *****

Private Sub txnAbort_Click()
    If Agent1.abortTransaction() = False Then
        txntext.Caption = "Error !"
    Else
        txntext.Caption = "Stopped"
    End If
    cmdPut.BackColor = &HC0E0FF
    cmdUpdate.BackColor = &HC0E0FF
    cmdTake.BackColor = &HC0E0FF
    cmdRead.BackColor = &HC0E0FF
    cmdStartNotify.BackColor = &H8000000F
    cmdStopNotify.BackColor = &H8000000F
End Sub

Private Sub txnClose_Click()
    If Agent1.closeTransaction() = False Then
        txntext.Caption = "Error !"
    Else
        txntext.Caption = "Stopped"
    End If
    cmdPut.BackColor = &HC0E0FF
    cmdUpdate.BackColor = &HC0E0FF
    cmdTake.BackColor = &HC0E0FF
    cmdRead.BackColor = &HC0E0FF
    cmdStartNotify.BackColor = &H8000000F
    cmdStopNotify.BackColor = &H8000000F
End Sub

Private Sub txnStart_Click()
    If Agent1.startTransaction() = False Then
        cmdPut.BackColor = &HC0E0FF
        cmdUpdate.BackColor = &HC0E0FF
        cmdTake.BackColor = &HC0E0FF
        cmdRead.BackColor = &HC0E0FF
        cmdStartNotify.BackColor = &H8000000F
        cmdStopNotify.BackColor = &H8000000F
        txntext.Caption = "Stopped"
    Else
        cmdPut.BackColor = vbRed
        cmdUpdate.BackColor = vbRed
        cmdTake.BackColor = vbRed
        cmdRead.BackColor = vbRed
        cmdStartNotify.BackColor = vbRed
    End If
End Sub

```

```
        cmdStopNotify.BackColor = vbRed
        txttext.Caption = "Running ..."
    End If
End Sub
```

C. C VERSION



```
Invoke
Auto

[o] : close transaction
[q] : quit
[space] : NEXT level

Enter command[ type:STRING ID:ISString ] ->i
Enter Object ID ->ISString

*****
**          Level 1 Help          **
*****
[t] : set object type
[i] : set object ID
[r] : remove object
[c] : create object
[l] : list all objects
[s] : start transaction
[a] : abort transaction
[o] : close transaction
[q] : quit
[space] : NEXT level

Enter command[ type:STRING ID:ISString ] ->t
[0]:INTEGER [1]:FLOAT [2]:LONG [3]:DOUBLE [4]:STRING
[5]:QUEUE [6]:STACK [7]:LINKLIST [8]:HASH [9]:BOOLEAN
Enter Object Type ->
```

1. CtestBench.C

```
#include <jni.h>
#include <stddef.h>
#include "agent.h"

char level1Help(void);
char level2Help(void);
void objType2Str(int oType, char* oTypeStr);
int objType;
char objTypeStr[20];
char objID[20];

int tranStartedFlag;
char processTSString();
char processTSInteger();
char processTSFloat();
char processTSLong();
char processTSDouble();
char processTSBoolean();

int main()
{
    char ret;
    char buf[255];

    tranStartedFlag = FALSE;
    initJVM();
    if(! initAgent()) goto destroy;

    getAgentSecurityPolicy(buf);
    printf(" AgentSecurityPolicy -> %s\n",buf);

    getAgentSpaceName(buf);
    printf(" AgentSpaceName -> %s\n",buf);

    getAgentServerCodebase(buf);
```

```

printf(" AgentServerCodebase -> %s\n",buf);

getAgentLookupGroup(buf);
printf(" AgentLookupGroup -> %s\n",buf);

getAgentLookupURL(buf);
printf(" AgentLookupURL -> %s\n",buf);

if(InitializeAgent(10000) == 1){
    createID(TS_STRING,"EntryID");
    objType = 5004;
    objType2Str(objType,objTypeStr);
    sprintf(objID,"EntryID");

    while(ret != 'q')
    {

        ret = level1Help();
        if( ret == ' ')
        {
            ret = level2Help();
        }

    }

}

destroy:
    if ((*env)->ExceptionOccurred(env))
        (*env)->ExceptionDescribe(env);

    (*jvm)->DestroyJavaVM(jvm);
    return 0;
}

char level1Help(void)
{
    while(1)
    {
        char inputChar[20];
        char inputStr[20];
        printf("\n\t*****\n");
        printf("\t**          Level 1 Help          **\n");
        printf("\t*****\n");
        printf("\t[t] : set object type \n");
        printf("\t[i] : set object ID\n");
        printf("\t[r] : remove object\n");
        printf("\t[c] : create object\n");
        printf("\t[l] : list all objects\n");
        //printf("\t[e] : clean object entries\n");
        printf("\t[s] : start transaction\n");
        printf("\t[a] : abort transaction\n");
        printf("\t[o] : close transaction\n");
        printf("\t[q] : quit\n");
        printf("\t[space] : NEXT level \n\n");
        if(tranStartedFlag)
            printf("\tTRANSACTION : Enter command[ type:%s ID:%s ] -
>",objTypeStr, objID);
        else
            printf("\tEnter command[ type:%s ID:%s ] ->",objTypeStr, objID);

        gets(inputChar);
        switch(inputChar[0])
        {
            case 'Q':
            case 'q':
                return 'q';
            case ' ':
                return ' ';
        }
    }
}

```

```

        case 't':
            printf("\t[0]:INTEGER [1]:FLOAT [2]:LONG [3]:DOUBLE
[4]:STRING\n");
            printf("\t[5]:QUEUE [6]:STACK [7]:LINKLIST [8]:HASH
[9]:BOOLEAN\n");
            printf("\tEnter Object Type ->");
            gets(inputStr);
            objType = 5000+atoi(inputStr);
            objType2Str(objType,objTypeStr);
            break;
        case 'i':
            printf("\tEnter Object ID ->");
            gets(objID);
            break;
        case 'r':
            printf("\n##Result## -> Remove %s[%s] %s\n",objID,
objTypeStr, (removeID(objType,objID) == 0 ? "Fail!" : "Ok!"));
            break;
        case 'c':
            printf("\n##Result## -> Create %s[%s] %s\n",objID,
objTypeStr, (createID(objType,objID) == 0 ? "Fail!" : "Ok!"));
            break;
        case 'l':
            break;
        case 'e':
            cleanTSClass(objType,objID);
            break;
        case 's':
            tranStartedFlag = startTransaction();
            printf("\n##Result## -> Start Transaction -> %s",
tranStartedFlag == TRUE ? "Ok!" : "Fail!");
            break;
        case 'a':
            tranStartedFlag = FALSE;
            printf("\n##Result## -> Abort Transaction -> %s",
abortTransaction() == TRUE ? "Ok!" : "Fail!");
            break;
        case 'o':
            tranStartedFlag = FALSE;
            printf("\n##Result## -> Close Transaction -> %s",
closeTransaction() == TRUE ? "Ok!" : "Fail!");
            break;
    }
}

char level2Help(void)
{
    char ret = '!';
    while(ret != ' ')
    {
        printf("\n\t*****\n");
        printf("\t** Level 2 Help **\n");
        printf("\t*****\n");
        printf("\t[w] : write entry\n");
        printf("\t[u] : update entry\n");
        printf("\t[r] : read entry\n");
        printf("\t[t] : take entry\n");
        printf("\t[space] : PREVIOUS level \n");
        if(tranStartedFlag)
            printf("\tTRANSACTION : Enter command[ type:%s ID:%s ] -
>",objTypeStr, objID);
        else
            printf("\tEnter command[ type:%s ID:%s ] ->",objTypeStr, objID);

        switch (objType)
        {

```

```

        case TS_INTEGER :
            ret = processTSInteger();
            break;
        case TS_FLOAT :
            ret = processTSFloat();
            break;
        case TS_LONG :
            ret = processTSLong();
            break;
        case TS_DOUBLE :
            ret = processTSDouble();
            break;
        case TS_STRING :
            ret = processTSString();
            break;
        case TS_QUEUE :
            break;
        case TS_STACK :
            break;
        case TS_LINKLIST :
            break;
        case TS_HASH :
            break;
        case TS_BOOLEAN :
            ret = processTSBoolean();
            break;
        default:
            break;
    }

    }
    return ret;
}

char processTSString()
{
    char inputChar[20];
    char inputStr[20];
    char outputStr[255];
    int ret;
    gets(inputChar);
    switch(inputChar[0])
    {
        case 'w':
            printf("\tEnter String ->");
            gets(inputStr);
            if(tranStartedFlag)
                ret = TSStringTransWrite(objID,inputStr,100000);
            else
                ret = TSStringWrite1(objID,inputStr);
            printf("\n##Result## -> Write %s[%s] %s : %s\n",objID,
objTypeStr,inputStr, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter String ->");
            gets(inputStr);
            if(tranStartedFlag)
                ret = TSStringTransUpdate(objID,inputStr,100000);
            else
                ret = TSStringUpdate1(objID,inputStr);
            printf("\n##Result## -> Update %s[%s] %s : %s\n",objID,
objTypeStr,inputStr, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'r':
            if(tranStartedFlag)
                TSStringTransRead(objID,outputStr,100000);
            else

```

```

        TSStringRead1(objID,outputStr);
        printf("\n##Result##    ->    Read    %s[%s]          %s\n",objID,
objTypeStr,outputStr);
        break;
    case 't':
        if(tranStartedFlag)
            TSStringTransTake(objID,outputStr,100000);
        else
            TSStringTake1(objID,outputStr);
        printf("\n##Result##    ->    Take    %s[%s]          %s\n",objID,
objTypeStr,outputStr);
        break;
    case ' ':
        return ' ';
    }
    return '!';
}

```

```

char processTSInteger()
{
    char inputChar[20];
    char inputStr[20];
    int ret, value;
    gets(inputChar);
    switch(inputChar[0])
    {
        case 'w':
            printf("\tEnter Integer value ->");
            value = atoi(gets(inputStr));
            if(tranStartedFlag)
                ret = TSIntegerTransWrite(objID,value,100000);
            else
                ret = TSIntegerWrite1(objID,value);
            printf("\n##Result##    ->    Write    %s[%s]          %d    :    %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter Integer ->");
            value = atoi(gets(inputStr));
            if(tranStartedFlag)
                ret = TSIntegerTransUpdate(objID,value,100000);
            else
                ret = TSIntegerUpdate1(objID,value);
            printf("\n##Result##    ->    Update    %s[%s]          %d    :    %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'r':
            if(tranStartedFlag)
                value = TSIntegerTransRead(objID,100000);
            else
                value = TSIntegerRead1(objID);
            printf("\n##Result##    ->    Read    %s[%s]          %d\n",objID,
objTypeStr,value);
            break;
        case 't':
            if(tranStartedFlag)
                value = TSIntegerTransTake(objID,100000);
            else
                value = TSIntegerTake1(objID);
            printf("\n##Result##    ->    Take    %s[%s]          %d\n",objID,
objTypeStr,value);
            break;
        case ' ':
            return ' ';
    }
}

```

```

    }
    return '!';
}

char processTSFloat()
{
    char inputChar[20];
    char inputStr[20];
    int ret;
    float value;
    gets(inputChar);
    switch(inputChar[0])
    {
        case 'w':
            printf("\tEnter Float value ->");
            //value = (float) atof(gets(inputStr));
            scanf("%f",value);
            if(tranStartedFlag)
                ret = TSFloatTransWrite(objID,value,100000);
            else
                ret = TSFloatWrite1(objID,value);
            printf("\n##Result## -> Write %s[%s] %f : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter Float ->");
            value = (float) atof(gets(inputStr));
            if(tranStartedFlag)
                ret = TSFloatTransUpdate(objID,value,100000);
            else
                ret = TSFloatUpdate1(objID,value);
            printf("\n##Result## -> Update %s[%s] %f : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'r':
            if(tranStartedFlag)
                value = TSFloatTransRead(objID,100000);
            else
                value = TSFloatRead1(objID);
            printf("\n##Result## -> Read %s[%s] %f\n",objID,
objTypeStr,value);
            break;
        case 't':
            if(tranStartedFlag)
                value = TSFloatTransTake(objID,100000);
            else
                value = TSFloatTake1(objID);
            printf("\n##Result## -> Take %s[%s] %f\n",objID,
objTypeStr,value);
            break;
        case ' ':
            return ' ';
    }
    return '!';
}

char processTSLong()
{
    char inputChar[20];
    char inputStr[20];
    int ret;
    long value;
    gets(inputChar);
    switch(inputChar[0])
    {

```

```

        case 'w':
            printf("\tEnter Long value ->");
            value = atol(gets(inputStr));
            if(tranStartedFlag)
                ret = TSLongTransWrite(objID,value,100000);
            else
                ret = TSLongWrite1(objID,value);
            printf("\n##Result## -> Write %s[%s] %d : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter Long ->");
            value = atol(gets(inputStr));
            if(tranStartedFlag)
                ret = TSLongTransUpdate(objID,value,100000);
            else
                ret = TSLongUpdate1(objID,value);
            printf("\n##Result## -> Update %s[%s] %d : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'r':
            if(tranStartedFlag)
                value = TSLongTransRead(objID,100000);
            else
                value = TSLongRead1(objID);
            printf("\n##Result## -> Read %s[%s] %d\n",objID,
objTypeStr,value);
            break;
        case 't':
            if(tranStartedFlag)
                value = TSLongTransTake(objID,100000);
            else
                value = TSLongTake1(objID);
            printf("\n##Result## -> Take %s[%s] %d\n",objID,
objTypeStr,value);
            break;
        case ' ':
            return ' ';
    }
    return '!';
}

```

```

char processTSDouble()
{
    char inputChar[20];
    char inputStr[20];
    int ret;
    double value;
    gets(inputChar);
    switch(inputChar[0])
    {
        case 'w':
            printf("\tEnter Double value ->");
            value = atof(gets(inputStr));
            if(tranStartedFlag)
                ret = TSDoubleTransWrite(objID,value,100000);
            else
                ret = TSDoubleWrite1(objID,value);
            printf("\n##Result## -> Write %s[%s] %f : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter Double ->");
            value = atof(gets(inputStr));
            if(tranStartedFlag)

```

```

        ret = TSDoubleTransUpdate(objID,value,100000);
    else
        ret = TSDoubleUpdate1(objID,value);
    printf("\n##Result## -> Update %s[%s]    %f    : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
    break;
    case 'r':
        if(tranStartedFlag)
            value = TSDoubleTransRead(objID,100000);
        else
            value = TSDoubleRead1(objID);
        printf("\n##Result## -> Read %s[%s]    %f\n",objID,
objTypeStr,value);
        break;
    case 't':
        if(tranStartedFlag)
            value = TSDoubleTransTake(objID,100000);
        else
            value = TSDoubleTake1(objID);
        printf("\n##Result## -> Take %s[%s]    %f\n",objID,
objTypeStr,value);
        break;
    case ' ':
        return ' ';
    }
    return '!';
}

char processTSBoolean()
{
    char inputChar[20];
    char inputStr[20];
    int ret, value;
    gets(inputChar);
    switch(inputChar[0])
    {
        case 'w':
            printf("\tEnter Boolean ->");
            value = atoi(gets(inputStr));
            if(tranStartedFlag)
                ret = TSBooleanTransWrite(objID,value,100000);
            else
                ret = TSBooleanWrite1(objID,value);
            printf("\n##Result## -> Write %s[%s]    %d    : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'u':
            printf("\tEnter Boolean ->");
            value = atoi(gets(inputStr));
            if(tranStartedFlag)
                ret = TSBooleanTransUpdate(objID,value,100000);
            else
                ret = TSBooleanUpdate1(objID,value);
            printf("\n##Result## -> Update %s[%s]    %d    : %s\n",objID,
objTypeStr,value, ret == TRUE ? "Ok!" : "Fail!");
            break;
        case 'r':
            if(tranStartedFlag)
                value = TSBooleanTransRead(objID,100000);
            else
                value = TSBooleanRead1(objID);
            printf("\n##Result## -> Read %s[%s]    %d\n",objID,
objTypeStr,value);
            break;
        case 't':
            if(tranStartedFlag)

```

```

        value = TSBooleanTransTake(objID,100000);
    else
        value = TSBooleanTake1(objID);
    printf("\n##Result##    ->    Take    %s[%s]    %d\n",objID,
objTypeStr,value);
    break;
    case ' ':
        return ' ';
}
return '!';
}

void objType2Str(int oType, char* oTypeStr)
{
    switch (oType)
    {
        case TS_INTEGER :
            sprintf(oTypeStr, "INTEGER");
            break;
        case TS_FLOAT :
            sprintf(oTypeStr, "FLOAT");
            break;
        case TS_LONG :
            sprintf(oTypeStr, "LONG");
            break;
        case TS_DOUBLE :
            sprintf(oTypeStr, "DOUBLE");
            break;
        case TS_STRING :
            sprintf(oTypeStr, "STRING");
            break;
        case TS_QUEUE :
            sprintf(oTypeStr, "QUEUE");
            break;
        case TS_STACK :
            sprintf(oTypeStr, "STACK");
            break;
        case TS_LINKLIST :
            sprintf(oTypeStr, "LINKLIST");
            break;
        case TS_HASH :
            sprintf(oTypeStr, "HASH");
            break;
        case TS_BOOLEAN :
            sprintf(oTypeStr, "BOOLEAN");
            break;
        default:
            sprintf(oTypeStr, "STRING");
            break;
    }
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. AGENT WRAPPER LISTING

1. AGENT.H

```
#ifndef _AGENT_H_
#define _AGENT_H_

#define PATH_SEPARATOR ';' /* define it to be ':' on Solaris */
/* #define USER_CLASSPATH "d:/Tuplespace/Jni/Invoke/Debug/" /* where Prog.class is */
#define USER_CLASSPATH "d:\\\\" /* where Prog.class is */
#define TRUE 1
#define FALSE 0

#define TS_INTEGER 5000
#define TS_FLOAT 5001
#define TS_LONG 5002
#define TS_DOUBLE 5003
#define TS_STRING 5004
#define TS_QUEUE 5005
#define TS_STACK 5006
#define TS_LINKLIST 5007
#define TS_HASH 5008
#define TS_BOOLEAN 5009

//*****

//*****
// Global Variables
//*****

JavaVM *jvm;
JNIEnv *env;
jclass agentClass; // Agent class;
jobject agentObject; // Agent object;

jmethodID jInitializeAgent, jTerminateAgent;
// Java space parameters methods
jmethodID jSetAgentSecurityPolicy, jGetAgentSecurityPolicy;
jmethodID jSetAgentSpaceName, jGetAgentSpaceName;
jmethodID jSetAgentServerCodebase, jGetAgentServerCodebase;
jmethodID jSetAgentLookupGroup, jGetAgentLookupGroup;
jmethodID jSetAgentLookupURL, jGetAgentLookupURL;

// Tuple methods
jmethodID jCreateID, jRemoveID, jCleanTSClass, jGetTSObject;
jmethodID jStartTransaction, jGetTransaction, jIsTransactionStarted;
jmethodID jAbortTransaction, jCloseTransaction, jUpdateTransactionHandle;

// TSInteger
jmethodID jTSIntegerWrite1, jTSIntegerWrite2, jTSIntegerUpdate1, jTSIntegerUpdate2;
jmethodID jTSIntegerRead1, jTSIntegerRead2, jTSIntegerTake1, jTSIntegerTake2;
jmethodID
jTSIntegerTransWrite, jTSIntegerTransUpdate, jTSIntegerTransRead, jTSIntegerTransTake;

// TSFloat
jmethodID jTSFloatWrite1, jTSFloatWrite2, jTSFloatUpdate1, jTSFloatUpdate2;
jmethodID jTSFloatRead1, jTSFloatRead2, jTSFloatTake1, jTSFloatTake2;
jmethodID jTSFloatTransWrite, jTSFloatTransUpdate, jTSFloatTransRead, jTSFloatTransTake;

// TSLong
jmethodID jTSLongWrite1, jTSLongWrite2, jTSLongUpdate1, jTSLongUpdate2;
jmethodID jTSLongRead1, jTSLongRead2, jTSLongTake1, jTSLongTake2;
jmethodID jTSLongTransWrite, jTSLongTransUpdate, jTSLongTransRead, jTSLongTransTake;

// TSDouble
jmethodID jTSDoubleWrite1, jTSDoubleWrite2, jTSDoubleUpdate1, jTSDoubleUpdate2;
jmethodID jTSDoubleRead1, jTSDoubleRead2, jTSDoubleTake1, jTSDoubleTake2;
jmethodID jTSDoubleTransWrite, jTSDoubleTransUpdate, jTSDoubleTransRead, jTSDoubleTransTake;
```

```

// TSString
jmethodID jTSStringWrite1,jTSStringWrite2,jTSStringUpdate1,jTSStringUpdate2;
jmethodID jTSStringRead1,jTSStringRead2,jTSStringTake1,jTSStringTake2;
jmethodID jTSStringTransWrite,jTSStringTransUpdate,jTSStringTransRead,jTSStringTransTake;

// TSHash
jmethodID jTSHashWrite1,jTSHashWrite2,jTSHashUpdate1,jTSHashUpdate2;
jmethodID jTSHashRead1,jTSHashRead2,jTSHashTake1,jTSHashTake2;
jmethodID jTSHashTransWrite,jTSHashTransUpdate,jTSHashTransRead,jTSHashTransTake;

// TSBoolean
jmethodID jTSBooleanWrite1,jTSBooleanWrite2,jTSBooleanUpdate1,jTSBooleanUpdate2;
jmethodID jTSBooleanRead1,jTSBooleanRead2,jTSBooleanTake1,jTSBooleanTake2;
jmethodID
jTSBooleanTransWrite,jTSBooleanTransUpdate,jTSBooleanTransRead,jTSBooleanTransTake;

//*****
// Agent setting
//*****
void initJVM(void);
int initAgent(void);
int initMethods(void);
void logError(char* str);

// Java space parameters methods
void setAgentSecurityPolicy(char* buf);
void getAgentSecurityPolicy(char* buf);
void setAgentSpaceName(char* buf);
void getAgentSpaceName(char* buf);
void setAgentServerCodebase(char* buf);
void getAgentServerCodebase(char* buf);
void setAgentLookupGroup(char* buf);
void getAgentLookupGroup(char* buf);
void setAgentLookupURL(char* buf);
void getAgentLookupURL(char* buf);

int createID(int classType, char* objectID);
int removeID(int classType, char* objectID);
int cleanTSClass(int classType, char* objectID);
jobject getTSObject(int classType, char* objectID);
int startTransaction();
jobject getTransaction();
int isTransactionStarted();
int abortTransaction();
int closeTransaction();
int updateTransactionHandle(int classType, char* objectID);

// TSInteger
int TSIntegerWrite1( char* objectID,int value);
int TSIntegerWrite2( char* objectID,int value, long lease);
int TSIntegerUpdate1( char* objectID,int value);
int TSIntegerUpdate2( char* objectID,int value, long lease);
int TSIntegerRead1( char* objectID);
int TSIntegerRead2( char* objectID, long timeout);
int TSIntegerTake1( char* objectID);
int TSIntegerTake2( char* objectID, long timeout);
int TSIntegerTransWrite( char* objectID,int value, long lease);
int TSIntegerTransUpdate( char* objectID,int value, long lease);
int TSIntegerTransRead( char* objectID, long timeout);
int TSIntegerTransTake( char* objectID, long timeout);

// TSFloat
int TSFloatWrite1( char* objectID,float value);
int TSFloatWrite2( char* objectID,float value, long lease);
int TSFloatUpdate1( char* objectID,float value);
int TSFloatUpdate2( char* objectID,float value, long lease);

```

```

float TSFloatRead1( char* objectID);
float TSFloatRead2( char* objectID, long timeout);
float TSFloatTake1( char* objectID);
float TSFloatTake2( char* objectID, long timeout);
int TSFloatTransWrite( char* objectID,float value, long lease);
int TSFloatTransUpdate( char* objectID,float value, long lease);
float TSFloatTransRead( char* objectID, long timeout);
float TSFloatTransTake( char* objectID, long timeout);

// TSLong
int TSLongWrite1( char* objectID,long value);
int TSLongWrite2( char* objectID,long value, long lease);
int TSLongUpdate1( char* objectID,long value);
int TSLongUpdate2( char* objectID,long value, long lease);
long TSLongRead1( char* objectID);
long TSLongRead2( char* objectID, long timeout);
long TSLongTake1( char* objectID);
long TSLongTake2( char* objectID, long timeout);
int TSLongTransWrite( char* objectID,long value, long lease);
int TSLongTransUpdate( char* objectID,long value, long lease);
long TSLongTransRead( char* objectID, long timeout);
long TSLongTransTake( char* objectID, long timeout);

// TSDouble
int TSDoubleWrite1( char* objectID,double value);
int TSDoubleWrite2( char* objectID,double value, long lease);
int TSDoubleUpdate1( char* objectID,double value);
int TSDoubleUpdate2( char* objectID,double value, long lease);
double TSDoubleRead1( char* objectID);
double TSDoubleRead2( char* objectID, long timeout);
double TSDoubleTake1( char* objectID);
double TSDoubleTake2( char* objectID, long timeout);
int TSDoubleTransWrite( char* objectID,double value, long lease);
int TSDoubleTransUpdate( char* objectID,double value, long lease);
double TSDoubleTransRead( char* objectID, long timeout);
double TSDoubleTransTake( char* objectID, long timeout);

// TSBoolean
int TSBooleanWrite1( char* objectID,int value);
int TSBooleanWrite2( char* objectID,int value, long lease);
int TSBooleanUpdate1( char* objectID,int value);
int TSBooleanUpdate2( char* objectID,int value, long lease);
int TSBooleanRead1( char* objectID);
int TSBooleanRead2( char* objectID, long timeout);
int TSBooleanTake1( char* objectID);
int TSBooleanTake2( char* objectID, long timeout);
int TSBooleanTransWrite( char* objectID,int value, long lease);
int TSBooleanTransUpdate( char* objectID,int value, long lease);
int TSBooleanTransRead( char* objectID, long timeout);
int TSBooleanTransTake( char* objectID, long timeout);

// TSHash
int TSHashWrite1( char* objectID);
int TSHashWrite2( char* objectID, long lease);
int TSHashUpdate1( char* objectID);
int TSHashUpdate2( char* objectID, long lease);
int TSHashRead1( char* objectID);
int TSHashRead2( char* objectID, long timeout);
int TSHashTake1( char* objectID);
int TSHashTake2( char* objectID, long timeout);
int TSHashTransWrite( char* objectID, long lease);
int TSHashTransUpdate( char* objectID, long lease);
int TSHashTransRead( char* objectID, long timeout);
int TSHashTransTake( char* objectID, long timeout);

//TSString
int TSStringWrite1( char* objectID,char* inputStr);
int TSStringWrite2( char* objectID,char* inputStr, long lease);

```

```
int TSStringUpdate1( char* objectID,char* inputStr);
int TSStringUpdate2( char* objectID,char* inputStr, long lease);
void TSStringRead1( char* objectID,char* outputBuf);
void TSStringRead2( char* objectID,char* outputBuf, long timeout);
void TSStringTake1( char* objectID,char* outputBuf);
void TSStringTake2( char* objectID,char* outputBuf, long timeout);
int TSStringTransWrite( char* objectID,char* inputStr, long lease);
int TSStringTransUpdate( char* objectID,char* inputStr, long lease);
void TSStringTransRead( char* objectID,char* outputBuf, long timeout);
void TSStringTransTake( char* objectID,char* outputBuf, long timeout);

#endif /* _AGENT_H_ */
```

2. AGENT.C

```
#include <jni.h>
#include <stddef.h>
#include "agent.h"

/*****
// Initialize Java Virtual Machine
// *****/
void initJVM(void)
{
    JavaVMInitArgs vm_args;
    JavaVMOption options[1];
    jint jvmStatus;

    //-----
    // Initialize JVM
    options[0].optionString = "-Djava.class.path=" USER_CLASSPATH;
    vm_args.version = 0x00010002;
    vm_args.options = options;
    vm_args.nOptions = 1;
    vm_args.ignoreUnrecognized = JNI_TRUE;
    jvmStatus = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
    if (jvmStatus < 0)
    {
        fprintf(stderr, "Can't create Java VM\n");
        exit(0);
    }
    //-----
}

/*****
// Initialize Agent
// *****/
int initAgent(void)
{
    //-----
    // Initialize Java Class
    jmethodID constructor;
    agentClass = (*env)->FindClass(env, "tuplespace.core.Agent");
    if (agentClass == 0)
        return FALSE;

    constructor = (*env)->GetMethodID(env, agentClass, "<init>", "()V");
    agentObject = (*env)->NewObject(env, agentClass, constructor);
    if (agentObject == 0)
        return FALSE;

    logError("init Agent Ok!\n");

    if(! initMethods())
        return FALSE;

    return TRUE;
    //-----
}

int initMethods(void)
{
    jclass tsClass;

    //-----
    // Get method handles of Java Program

    jInitializeAgent = (*env)->GetMethodID(env, agentClass, "InitAgent", "(J)Z");
    jTerminateAgent = (*env)->GetMethodID(env, agentClass, "TerminateAgent", "()V");
    if (!jInitializeAgent || !jTerminateAgent) return FALSE;
}
```

```

        jSetAgentSecurityPolicy = (*env)->GetMethodID(env, agentClass,
"setAgentSecurityPolicy", "(Ljava/lang/String;)V");
        jGetAgentSecurityPolicy = (*env)->GetMethodID(env, agentClass,
"getAgentSecurityPolicy", "()Ljava/lang/String;");
        if (!jGetAgentSecurityPolicy || !jSetAgentSecurityPolicy) return FALSE;

        jSetAgentSpaceName = (*env)->GetMethodID(env, agentClass, "setAgentSpaceName",
"(Ljava/lang/String;)V");
        jGetAgentSpaceName = (*env)->GetMethodID(env, agentClass, "getAgentSpaceName",
"()Ljava/lang/String;");
        if (!jGetAgentSpaceName || !jSetAgentSpaceName) return FALSE;

        jSetAgentServerCodebase = (*env)->GetMethodID(env, agentClass,
"setAgentServerCodebase", "(Ljava/lang/String;)V");
        jGetAgentServerCodebase = (*env)->GetMethodID(env, agentClass,
"getAgentServerCodebase", "()Ljava/lang/String;");
        if (!jGetAgentServerCodebase || !jSetAgentServerCodebase) return FALSE;

        jSetAgentLookupGroup = (*env)->GetMethodID(env, agentClass, "setAgentLookupGroup",
"(Ljava/lang/String;)V");
        jGetAgentLookupGroup = (*env)->GetMethodID(env, agentClass, "getAgentLookupGroup",
"()Ljava/lang/String;");
        if (!jGetAgentLookupGroup || !jSetAgentLookupGroup) return FALSE;

        jSetAgentLookupURL = (*env)->GetMethodID(env, agentClass, "setAgentLookupURL",
"(Ljava/lang/String;)V");
        jGetAgentLookupURL = (*env)->GetMethodID(env, agentClass, "getAgentLookupURL",
"()Ljava/lang/String;");
        if (!jGetAgentLookupURL || !jSetAgentLookupURL) return FALSE;

        jStartTransaction = (*env)->GetMethodID(env, agentClass, "startTransaction",
"()Z");
        jGetTransaction = (*env)->GetMethodID(env, agentClass, "getTransaction",
"()Lnet/jini/core/transaction/Transaction;");
        jIsTransactionStarted = (*env)->GetMethodID(env, agentClass,
"isTransactionStarted", "()Z");
        jAbortTransaction = (*env)->GetMethodID(env, agentClass, "abortTransaction",
"()Z");
        jCloseTransaction = (*env)->GetMethodID(env, agentClass, "closeTransaction",
"()Z");
        jUpdateTransactionHandle = (*env)->GetMethodID(env, agentClass,
"updateTransactionHandle", "(ILjava/lang/String;)Z");
        if (!jStartTransaction || !jGetAgentLookupURL || !jIsTransactionStarted ||
!jUpdateTransactionHandle
|| !jAbortTransaction || !jCloseTransaction) return FALSE;

        jCreateID = (*env)->GetMethodID(env, agentClass, "createID",
"(ILjava/lang/String;)Z");
        jRemoveID = (*env)->GetMethodID(env, agentClass, "removeID",
"(ILjava/lang/String;)Z");
        jCleanTSClass = (*env)->GetMethodID(env, agentClass, "cleanTSClass",
"(ILjava/lang/String;)Z");
        jGetTSObject = (*env)->GetMethodID(env, agentClass, "getTSObject",
"(ILjava/lang/String;)Ljava/lang/Object;");
        if (!jCreateID || !jRemoveID || !jCleanTSClass || !jGetTSObject) return FALSE;

        // TSInteger
        tsClass = (*env)->FindClass(env, "tuplespace.core.TSInteger");
        jTSIntegerWrite1 = (*env)->GetMethodID(env, tsClass, "write1", "(I)Z");
        jTSIntegerWrite2 = (*env)->GetMethodID(env, tsClass, "write2", "(IJ)Z");
        jTSIntegerUpdate1 = (*env)->GetMethodID(env, tsClass, "update1", "(I)Z");
        jTSIntegerUpdate2 = (*env)->GetMethodID(env, tsClass, "update2", "(IJ)Z");
        jTSIntegerRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()I");
        jTSIntegerRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)I");
        jTSIntegerTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()I");

```

```

jTSIntegerTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)I");
jTSIntegerTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(IJ)Z");
jTSIntegerTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(IJ)Z");
jTSIntegerTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)I");
jTSIntegerTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)I");

    if (!jTSIntegerWrite1 || !jTSIntegerWrite2 || !jTSIntegerUpdate1 ||
!jTSIntegerUpdate2 ||
        !jTSIntegerRead1 || !jTSIntegerRead2 || !jTSIntegerTake1 ||
!jTSIntegerTake2 ||
        !jTSIntegerTransWrite || !jTSIntegerTransUpdate || !jTSIntegerTransRead ||
!jTSIntegerTransTake)
        return FALSE;

    // TSFloat
    tsClass = (*env)->FindClass(env, "tuplespace.core.TSFloat");
    jTSFloatWrite1 = (*env)->GetMethodID(env, tsClass, "write1", "(F)Z");
    jTSFloatWrite2 = (*env)->GetMethodID(env, tsClass, "write2", "(FJ)Z");
    jTSFloatUpdate1 = (*env)->GetMethodID(env, tsClass, "update1", "(F)Z");
    jTSFloatUpdate2 = (*env)->GetMethodID(env, tsClass, "update2", "(FJ)Z");
    jTSFloatRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()F");
    jTSFloatRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)F");
    jTSFloatTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()F");
    jTSFloatTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)F");
    jTSFloatTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(FJ)Z");
    jTSFloatTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(FJ)Z");
    jTSFloatTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)F");
    jTSFloatTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)F");

    if (!jTSFloatWrite1 || !jTSFloatWrite2 || !jTSFloatUpdate1 || !jTSFloatUpdate2 ||
        !jTSFloatRead1 || !jTSFloatRead2 || !jTSFloatTake1 || !jTSFloatTake2 ||
        !jTSFloatTransWrite || !jTSFloatTransUpdate || !jTSFloatTransRead ||
!jTSFloatTransTake)
        return FALSE;

    // TSLong
    tsClass = (*env)->FindClass(env, "tuplespace.core.TSLong");
    jTSLongWrite1 = (*env)->GetMethodID(env, tsClass, "write1", "(J)Z");
    jTSLongWrite2 = (*env)->GetMethodID(env, tsClass, "write2", "(JJ)Z");
    jTSLongUpdate1 = (*env)->GetMethodID(env, tsClass, "update1", "(J)Z");
    jTSLongUpdate2 = (*env)->GetMethodID(env, tsClass, "update2", "(JJ)Z");
    jTSLongRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()J");
    jTSLongRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)J");
    jTSLongTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()J");
    jTSLongTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)J");
    jTSLongTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(JJ)Z");
    jTSLongTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(JJ)Z");
    jTSLongTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)J");
    jTSLongTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)J");

    if (!jTSLongWrite1 || !jTSLongWrite2 || !jTSLongUpdate1 || !jTSLongUpdate2 ||
        !jTSLongRead1 || !jTSLongRead2 || !jTSLongTake1 || !jTSLongTake2 ||
        !jTSLongTransWrite || !jTSLongTransUpdate || !jTSLongTransRead ||
!jTSLongTransTake)
        return FALSE;

    // TSDouble
    tsClass = (*env)->FindClass(env, "tuplespace.core.TSDouble");
    jTSDoubleWrite1 = (*env)->GetMethodID(env, tsClass, "write1", "(D)Z");
    jTSDoubleWrite2 = (*env)->GetMethodID(env, tsClass, "write2", "(DJ)Z");
    jTSDoubleUpdate1 = (*env)->GetMethodID(env, tsClass, "update1", "(D)Z");
    jTSDoubleUpdate2 = (*env)->GetMethodID(env, tsClass, "update2", "(DJ)Z");
    jTSDoubleRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()D");
    jTSDoubleRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)D");
    jTSDoubleTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()D");
    jTSDoubleTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)D");
    jTSDoubleTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(DJ)Z");
    jTSDoubleTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(DJ)Z");
    jTSDoubleTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)D");

```

```

jTSDoubleTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)D");

if (!jTSDoubleWrite1 || !jTSDoubleWrite2 || !jTSDoubleUpdate1 || !jTSDoubleUpdate2
||
    !jTSDoubleRead1 || !jTSDoubleRead2 || !jTSDoubleTake1 || !jTSDoubleTake2 ||
    !jTSDoubleTransWrite || !jTSDoubleTransUpdate || !jTSDoubleTransRead ||
!jTSDoubleTransTake)
    return FALSE;

// TSBoolean
tsClass = (*env)->FindClass(env, "tuplespace.core.TSBoolean");
jTSBooleanWrite1 = (*env)->GetMethodID(env, tsClass, "write1", "(Z)Z");
jTSBooleanWrite2 = (*env)->GetMethodID(env, tsClass, "write2", "(ZJ)Z");
jTSBooleanUpdate1 = (*env)->GetMethodID(env, tsClass, "update1", "(Z)Z");
jTSBooleanUpdate2 = (*env)->GetMethodID(env, tsClass, "update2", "(ZJ)Z");
jTSBooleanRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()Z");
jTSBooleanRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)Z");
jTSBooleanTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()Z");
jTSBooleanTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)Z");
jTSBooleanTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(ZJ)Z");
jTSBooleanTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(ZJ)Z");
jTSBooleanTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)Z");
jTSBooleanTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)Z");

if (!jTSBooleanWrite1 || !jTSBooleanWrite2 || !jTSBooleanUpdate1 ||
!jTSBooleanUpdate2 ||
    !jTSBooleanRead1 || !jTSBooleanRead2 || !jTSBooleanTake1 ||
!jTSBooleanTake2 ||
    !jTSBooleanTransWrite || !jTSBooleanTransUpdate || !jTSBooleanTransRead ||
!jTSBooleanTransTake)
    return FALSE;

// TSHash
tsClass = (*env)->FindClass(env, "tuplespace.core.TSHash");
jTSHashWrite1 = (*env)->GetMethodID(env, tsClass, "write", "()Z");
jTSHashWrite2 = (*env)->GetMethodID(env, tsClass, "write1", "(J)Z");
jTSHashUpdate1 = (*env)->GetMethodID(env, tsClass, "update", "()Z");
jTSHashUpdate2 = (*env)->GetMethodID(env, tsClass, "update1", "(J)Z");
jTSHashRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists", "()Z");
jTSHashRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1", "(J)Z");
jTSHashTake1 = (*env)->GetMethodID(env, tsClass, "takeIfExists", "()Z");
jTSHashTake2 = (*env)->GetMethodID(env, tsClass, "takeIfExists1", "(J)Z");
jTSHashTransWrite = (*env)->GetMethodID(env, tsClass, "transWrite", "(J)Z");
jTSHashTransUpdate = (*env)->GetMethodID(env, tsClass, "transUpdate", "(J)Z");
jTSHashTransRead = (*env)->GetMethodID(env, tsClass, "transRead", "(J)Z");
jTSHashTransTake = (*env)->GetMethodID(env, tsClass, "transTake", "(J)Z");

if (!jTSHashWrite1 || !jTSHashWrite2 || !jTSHashUpdate1 || !jTSHashUpdate2 ||
    !jTSHashRead1 || !jTSHashRead2 || !jTSHashTake1 || !jTSHashTake2 ||
    !jTSHashTransWrite || !jTSHashTransUpdate || !jTSHashTransRead ||
!jTSHashTransTake)
    return FALSE;

// TSString
tsClass = (*env)->FindClass(env, "tuplespace.core.TSString");
jTSStringWrite1 = (*env)->GetMethodID(env, tsClass, "write1",
"(Ljava/lang/String;)Z");
jTSStringWrite2 = (*env)->GetMethodID(env, tsClass, "write2",
"(Ljava/lang/String;J)Z");
jTSStringUpdate1 = (*env)->GetMethodID(env, tsClass, "update1",
"(Ljava/lang/String;)Z");
jTSStringUpdate2 = (*env)->GetMethodID(env, tsClass, "update2",
"(Ljava/lang/String;J)Z");
jTSStringRead1 = (*env)->GetMethodID(env, tsClass, "readIfExists",
"()Ljava/lang/String;");
jTSStringRead2 = (*env)->GetMethodID(env, tsClass, "readIfExists1",
"(J)Ljava/lang/String;");

```

```

    jTSStringTake1      =      (*env)->GetMethodID(env,      tsClass,      "takeIfExists",
"()Ljava/lang/String;");
    jTSStringTake2      =      (*env)->GetMethodID(env,      tsClass,      "takeIfExists1",
"(J)Ljava/lang/String;");
    jTSStringTransWrite  =      (*env)->GetMethodID(env,      tsClass,      "transWrite",
"(Ljava/lang/String;J)Z");
    jTSStringTransUpdate =      (*env)->GetMethodID(env,      tsClass,      "transUpdate",
"(Ljava/lang/String;J)Z");
    jTSStringTransRead   =      (*env)->GetMethodID(env,      tsClass,      "transRead",
"(J)Ljava/lang/String;");
    jTSStringTransTake   =      (*env)->GetMethodID(env,      tsClass,      "transTake",
"(J)Ljava/lang/String;");

    if (!jTSStringWrite1 || !jTSStringWrite2 || !jTSStringUpdate1 || !jTSStringUpdate2
||
        !jTSStringRead1 || !jTSStringRead2 || !jTSStringTake1 || !jTSStringTake2 ||
        !jTSStringTransWrite || !jTSStringTransUpdate || !jTSStringTransRead ||
!jTSStringTransTake)
        return FALSE;

    logError("init Methods Ok!\n");
    return TRUE;
    //-----
}

void logError(char* string)
{
    printf("%s", string);
}

//*****
// Terminate Agent - Stop agent and release all resources
//*****

void endAgent(void)
{
    if ((*env)->ExceptionOccurred(env))
        (*env)->ExceptionDescribe(env);

    (*jvm)->DestroyJavaVM(jvm);
}

int InitializeAgent(long timeout)
{
    if ( agentObject == 0 || jInitializeAgent ==0)
    {
        logError("Fail! InitializeAgent \n");
        return 0;
    }

    return (*env)->CallBooleanMethod(env, agentObject, jInitializeAgent, timeout);
}

//*****
// Set/Get AgentSecurityPolicy
//*****
void setAgentSecurityPolicy(char* buf)
{
    jstring jstr;
    if ( agentObject == 0 || jSetAgentSecurityPolicy ==0)
    {
        logError("Fail! setAgentSecurityPolicy \n");
        return;
    }
}

```

```

        jstr = (*env)->NewStringUTF(env,buf);
        (*env)->CallVoidMethod(env, agentObject, jSetAgentSecurityPolicy, jstr);
    }

void getAgentSecurityPolicy(char* buf)
{
    jstring jstr;
    const jbyte *str;
    if ( agentObject == 0 || jGetAgentSecurityPolicy ==0)
    {
        logError("Fail! getAgentSecurityPolicy \n");
        return;
    }

    jstr = (*env)->CallObjectMethod(env, agentObject, jGetAgentSecurityPolicy);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(buf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(buf," ");
}

//*****
// Set/Get AgentSpaceName
//*****
void setAgentSpaceName(char* buf)
{
    jstring jstr;
    if ( agentObject == 0 || jSetAgentSpaceName ==0)
    {
        logError("Fail! setAgentSpaceName \n");
        return;
    }

    jstr = (*env)->NewStringUTF(env,buf);
    (*env)->CallVoidMethod(env, agentObject, jSetAgentSpaceName, jstr);
}

void getAgentSpaceName(char* buf)
{
    jstring jstr;
    const jbyte *str;
    if ( agentObject == 0 || jGetAgentSpaceName ==0)
    {
        logError("Fail! getAgentSpaceName \n");
        return;
    }

    jstr = (*env)->CallObjectMethod(env, agentObject, jGetAgentSpaceName);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(buf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(buf," ");
}

//*****
// Set/Get AgentServerCodebase

```

```

//*****
void setAgentServerCodebase(char* buf)
{
    jstring jstr;
    if ( agentObject == 0 || jSetAgentServerCodebase ==0)
    {
        logError("Fail! setAgentServerCodebase \n");
        return;
    }

    jstr = (*env)->NewStringUTF(env,buf);
    (*env)->CallVoidMethod(env, agentObject, jSetAgentServerCodebase, jstr);
}

void getAgentServerCodebase(char* buf)
{
    jstring jstr;
    const jbyte *str;
    if ( agentObject == 0 || jGetAgentServerCodebase ==0)
    {
        logError("Fail! getAgentServerCodebase \n");
        return;
    }

    jstr = (*env)->CallObjectMethod(env, agentObject, jGetAgentServerCodebase);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(buf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(buf," ");
}

//*****
// Set/Get AgentLookupGroup
//*****
void setAgentLookupGroup(char* buf)
{
    jstring jstr;
    if ( agentObject == 0 || jSetAgentLookupGroup ==0)
    {
        logError("Fail! setAgentLookupGroup \n");
        return;
    }

    jstr = (*env)->NewStringUTF(env,buf);
    (*env)->CallVoidMethod(env, agentObject, jSetAgentLookupGroup, jstr);
}

void getAgentLookupGroup(char* buf)
{
    jstring jstr;
    const jbyte *str;
    if ( agentObject == 0 || jGetAgentLookupGroup ==0)
    {
        logError("Fail! getAgentLookupGroup \n");
        return;
    }

    jstr = (*env)->CallObjectMethod(env, agentObject, jGetAgentLookupGroup);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(buf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
}

```

```

    }
    else
        sprintf(buf, " ");
}

//*****
// Set/Get AgentLookupURL
//*****
void setAgentLookupURL(char* buf)
{
    jstring jstr;
    if ( agentObject == 0 || jSetAgentLookupURL ==0)
    {
        logError("Fail! setAgentLookupURL \n");
        return;
    }

    jstr = (*env)->NewStringUTF(env,buf);
    (*env)->CallVoidMethod(env, agentObject, jSetAgentLookupURL, jstr);
}

void getAgentLookupURL(char* buf)
{
    jstring jstr;
    const jbyte *str;
    if ( agentObject == 0 || jGetAgentLookupURL ==0)
    {
        logError("Fail! getAgentLookupURL \n");
        return;
    }

    jstr = (*env)->CallObjectMethod(env, agentObject, jGetAgentLookupURL);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(buf, "%s", str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(buf, " ");
}

//*****
// Create/Remove tuple ID
//*****
int createID(int classType, char* objectID)
{
    jstring jstr;
    if ( agentObject == 0 || jCreateID ==0 || objectID == 0)
    {
        logError("Fail! createID \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,objectID);
    return (*env)->CallBooleanMethod(env, agentObject, jCreateID, classType, jstr);
}

int removeID(int classType, char* objectID)
{
    jstring jstr;
    if ( agentObject == 0 || jRemoveID ==0 || objectID == 0)
    {
        logError("Fail! removeID \n");
        return FALSE;
    }
}

```

```

        jstr = (*env)->NewStringUTF(env,objectID);
        return (*env)->CallBooleanMethod(env, agentObject, jRemoveID, classType, jstr);
    }

int cleanTSClass(int classType,char* objectID)
{
    jstring jstr;
    if ( agentObject == 0 || jCleanTSClass ==0 || objectID == 0 )
    {
        logError("Fail! cleanTSClass \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,objectID);
    return (*env)->CallBooleanMethod(env, agentObject, jCleanTSClass, classType,
jstr);
}

jobject getTSObject(int classType, char* objectID)
{
    jstring jstr;
    if ( agentObject == 0 || jGetTSObject ==0 || objectID == 0 )
    {
        logError("Fail! getTSObject \n");
        return 0;
    }

    jstr = (*env)->NewStringUTF(env,objectID);
    return (*env)->CallObjectMethod(env, agentObject, jGetTSObject, classType, jstr);
}

int startTransaction()
{
    if ( agentObject == 0 || jStartTransaction ==0 )
    {
        logError("Fail! startTransaction \n");
        return 0;
    }
    return (*env)->CallBooleanMethod(env, agentObject, jStartTransaction);
}

jobject getTransaction()
{
    if ( agentObject == 0 || jGetTransaction ==0 )
    {
        logError("Fail! getTransaction \n");
        return 0;
    }
    return (*env)->CallObjectMethod(env, agentObject, jGetTransaction);
}

int isTransactionStarted()
{
    if ( agentObject == 0 || jIsTransactionStarted ==0 )
    {
        logError("Fail! isTransactionStarted \n");
        return 0;
    }
    return (*env)->CallBooleanMethod(env, agentObject, jIsTransactionStarted);
}

int abortTransaction()
{
    if ( agentObject == 0 || jAbortTransaction ==0 )
    {
        logError("Fail! abortTransaction \n");
        return 0;
    }
    return (*env)->CallBooleanMethod(env, agentObject, jAbortTransaction);
}

```

```

}

int closeTransaction()
{
    if ( agentObject == 0 || jCloseTransaction ==0 )
    {
        logError("Fail! closeTransaction \n");
        return 0;
    }
    return (*env)->CallBooleanMethod(env, agentObject, jCloseTransaction);
}

int updateTransactionHandle(int classType, char* objectID)
{
    jstring jstr;
    if ( agentObject == 0 || jUpdateTransactionHandle ==0 || objectID == 0 )
    {
        logError("Fail! createID \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,objectID);
    return (*env)->CallBooleanMethod(env, agentObject, jUpdateTransactionHandle,
classType, jstr);
}

//*****
//*****
// TSInteger Methods
//*****
// *** Write ***
int TSIntegerWrite1(char* objectID, int value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerWrite1 ==0 || jTSObject == 0 )
    {
        logError("Fail! TSIntegerWrite1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerWrite1, value);
}

int TSIntegerWrite2(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerWrite1 ==0 || jTSObject == 0 )
    {
        logError("Fail! TSIntegerWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerWrite2, value, lease);
}

// *** Update ***
int TSIntegerUpdate1(char* objectID, int value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerUpdate1 ==0 || jTSObject == 0 )
    {
        logError("Fail! TSIntegerUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerUpdate1, value);
}

```

```

int TSIntegerUpdate2(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSIntegerUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerUpdate2, value, lease);
}

// *** Read ***
int TSIntegerRead1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSIntegerRead1 \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerRead1);
}

int TSIntegerRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerRead2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSIntegerRead2 \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerRead2, timeout);
}

// *** Take ***
int TSIntegerTake1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerTake1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSIntegerTake1 \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerTake1);
}

int TSIntegerTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSIntegerTake2 \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerTake2, timeout);
}

// *** Transaction ***
int TSIntegerTransWrite(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);

```

```

        if ( jTSIntegerTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_INTEGER, objectID)== 0)
        {
            logError("Fail! TSIntegerTransWrite \n");
            return FALSE;
        }
        return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerTransWrite, value,
lease);
    }

int TSIntegerTransUpdate(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_INTEGER, objectID)== 0)
    {
        logError("Fail! TSIntegerTransUpdate \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSIntegerTransUpdate, value,
lease);
}

int TSIntegerTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerTransRead ==0 || jTSObject == 0 || updateTransactionHandle(
TS_INTEGER, objectID)== 0)
    {
        logError("Fail! TSIntegerTransRead \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerTransRead, timeout);
}

int TSIntegerTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_INTEGER, objectID);
    if ( jTSIntegerTransTake ==0 || jTSObject == 0 || updateTransactionHandle(
TS_INTEGER, objectID)== 0)
    {
        logError("Fail! TSIntegerTransTake \n");
        return FALSE;
    }
    return (*env)->CallIntMethod(env, jTSObject, jTSIntegerTransTake, timeout);
}

//*****

//*****
//*****
// TSFloat Methods
//*****
// *** Write ***
int TSFloatWritel(char* objectID, float value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatWritel ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatWritel \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatWritel, value);
}

```

```

int TSFloatWrite2(char* objectID, float value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatWrite2, value, lease);
}

// *** Update ***
int TSFloatUpdate1(char* objectID, float value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatUpdate1, value);
}

int TSFloatUpdate2(char* objectID, float value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatUpdate2, value, lease);
}

// *** Read ***
float TSFloatRead1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatRead1 \n");
        return FALSE;
    }
    return (*env)->CallFloatMethod(env, jTSObject, jTSFloatRead1);
}

float TSFloatRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatRead2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatRead2 \n");
        return FALSE;
    }
    return (*env)->CallFloatMethod(env, jTSObject, jTSFloatRead2, timeout);
}

// *** Take ***
float TSFloatTake1(char* objectID)
{
    jobject jTSObject;

```

```

        jTSObject = getTSObject((int)TS_FLOAT, objectID);
        if ( jTSFloatTake1 ==0 || jTSObject == 0)
        {
            logError("Fail! TSFloatTake1 \n");
            return FALSE;
        }
        return (*env)->CallFloatMethod(env, jTSObject, jTSFloatTake1);
    }

float TSFloatTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSFloatTake2 \n");
        return FALSE;
    }
    return (*env)->CallFloatMethod(env, jTSObject, jTSFloatTake2, timeout);
}

// *** Transaction ***
int TSFloatTransWrite(char* objectID, float value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_FLOAT, objectID)== 0)
    {
        logError("Fail! TSFloatTransWrite \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatTransWrite, value,
lease);
}

int TSFloatTransUpdate(char* objectID, float value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_FLOAT, objectID)== 0)
    {
        logError("Fail! TSFloatTransUpdate \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSFloatTransUpdate, value,
lease);
}

float TSFloatTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);
    if ( jTSFloatTransRead ==0 || jTSObject == 0 || updateTransactionHandle( TS_FLOAT,
objectID)== 0)
    {
        logError("Fail! TSFloatTransRead \n");
        return FALSE;
    }
    return (*env)->CallFloatMethod(env, jTSObject, jTSFloatTransRead, timeout);
}

float TSFloatTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_FLOAT, objectID);

```

```

        if ( jTSFloatTransTake ==0 || jTSObject == 0 || updateTransactionHandle( TS_FLOAT,
objectID)== 0)
        {
            logError("Fail! TSFloatTransTake \n");
            return FALSE;
        }
        return (*env)->CallFloatMethod(env, jTSObject, jTSFloatTransTake, timeout);
    }

//*****

//*****
// TSLong Methods
//*****
// *** Write ***
int TSLongWrite1(char* objectID, long value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongWrite1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongWrite1, value);
}

int TSLongWrite2(char* objectID, long value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongWrite2, value, lease);
}

// *** Update ***
int TSLongUpdate1(char* objectID, long value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongUpdate1, value);
}

int TSLongUpdate2(char* objectID, long value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongUpdate2, value, lease);
}

// *** Read ***
long TSLongRead1(char* objectID)

```

```

{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongRead1 \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongRead1);
}

long TSLongRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongRead2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongRead2 \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongRead2, timeout);
}

// *** Take ***
long TSLongTake1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTake1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongTake1 \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongTake1);
}

long TSLongTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSLongTake2 \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongTake2, timeout);
}

// *** Transaction ***
int TSLongTransWrite(char* objectID, long value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_LONG, objectID)== 0)
    {
        logError("Fail! TSLongTransWrite \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongTransWrite, value, lease);
}

int TSLongTransUpdate(char* objectID, long value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_LONG, objectID)== 0)
    {

```

```

        logError("Fail! TSLongTransUpdate \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSLongTransUpdate, value,
lease);
}

long TSLongTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTransRead ==0 || jTSObject == 0 || updateTransactionHandle( TS_LONG,
objectID)== 0)
    {
        logError("Fail! TSLongTransRead \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongTransRead, timeout);
}

long TSLongTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_LONG, objectID);
    if ( jTSLongTransTake ==0 || jTSObject == 0 || updateTransactionHandle( TS_LONG,
objectID)== 0)
    {
        logError("Fail! TSLongTransTake \n");
        return FALSE;
    }
    return (long)(*env)->CallLongMethod(env, jTSObject, jTSLongTransTake, timeout);
}

//*****

//*****
//*****
// TSDouble Methods
//*****
// *** Write ***
int TSDoubleWrite1(char* objectID, double value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleWrite1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleWrite1, value);
}

int TSDoubleWrite2(char* objectID, double value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleWrite2, value, lease);
}

// *** Update ***
int TSDoubleUpdate1(char* objectID, double value)
{

```

```

    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleUpdate1, value);
}

int TSDoubleUpdate2(char* objectID, double value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleUpdate2, value, lease);
}

// *** Read ***
double TSDoubleRead1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleRead1 \n");
        return FALSE;
    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleRead1);
}

double TSDoubleRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleRead2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleRead2 \n");
        return FALSE;
    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleRead2, timeout);
}

// *** Take ***
double TSDoubleTake1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTake1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleTake1 \n");
        return FALSE;
    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleTake1);
}

double TSDoubleTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSDoubleTake2 \n");
        return FALSE;
    }

```

```

    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleTake2, timeout);
}

// *** Transaction ***
int TSDoubleTransWrite(char* objectID, double value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSDoubleTransWrite \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleTransWrite, value,
lease);
}

int TSDoubleTransUpdate(char* objectID, double value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSDoubleTransUpdate \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSDoubleTransUpdate, value,
lease);
}

double TSDoubleTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTransRead ==0 || jTSObject == 0 || updateTransactionHandle(
TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSDoubleTransRead \n");
        return FALSE;
    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleTransRead, timeout);
}

double TSDoubleTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSDoubleTransTake ==0 || jTSObject == 0 || updateTransactionHandle(
TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSDoubleTransTake \n");
        return FALSE;
    }
    return (*env)->CallDoubleMethod(env, jTSObject, jTSDoubleTransTake, timeout);
}

//*****

//*****
//*****
// TSGlobal Methods
//*****
// *** Write ***
int TSGlobalWrite1(char* objectID, int value)

```

```

{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanWrite1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanWrite1, value);
}

int TSBooleanWrite2(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanWrite2, value, lease);
}

// *** Update ***
int TSBooleanUpdate1(char* objectID, int value)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanUpdate1, value);
}

int TSBooleanUpdate2(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanUpdate2, value, lease);
}

// *** Read ***
int TSBooleanRead1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanRead1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanRead1);
}

int TSBooleanRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanRead2 ==0 || jTSObject == 0)
    {

```

```

        logError("Fail! TSBooleanRead2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanRead2, timeout);
}

// *** Take ***
int TSBooleanTake1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTake1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanTake1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTake1);
}

int TSBooleanTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSBooleanTake2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTake2, timeout);
}

// *** Transaction ***
int TSBooleanTransWrite(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSBooleanTransWrite \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTransWrite, value,
lease);
}

int TSBooleanTransUpdate(char* objectID, int value, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSBooleanTransUpdate \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTransUpdate, value,
lease);
}

int TSBooleanTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTransRead ==0 || jTSObject == 0 || updateTransactionHandle(
TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSBooleanTransRead \n");

```

```

        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTransRead, timeout);
}

int TSBooleanTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSBooleanTransTake ==0 || jTSObject == 0 || updateTransactionHandle(
TS_DOUBLE, objectID)== 0)
    {
        logError("Fail! TSBooleanTransTake \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSBooleanTransTake, timeout);
}

//*****

//*****
//*****
// TSString Methods
//*****
// *** Write ***
int TSStringWritel(char* objectID, char* inputString)
{
    jobject jTSObject;
    jstring jstr;

    jTSObject = getTSObject((int)TS_STRING, objectID);
    if ( jTSStringWritel ==0 || jTSObject == 0)
    {
        logError("Fail! TSStringWritel \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env, inputString);
    if(jstr != 0)
        return (*env)->CallBooleanMethod(env, jTSObject, jTSStringWritel, jstr);
    else
        return FALSE;
}

int TSStringWrite2(char* objectID, char* inputString, long lease)
{
    jobject jTSObject;
    jstring jstr;

    jTSObject = getTSObject((int)TS_STRING, objectID);
    if ( jTSStringWrite2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSStringWrite2 \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env, inputString);
    return (*env)->CallBooleanMethod(env, jTSObject, jTSStringWrite2, jstr, lease);
}

// *** Update ***

int TSStringUpdatel(char* objectID, char* inputString)
{
    jobject jTSObject;
    jstring jstr;

```

```

    jTSTObject = getTSTObject((int)TS_STRING, objectID);
    if ( jTSTStringUpdate1 ==0 || jTSTObject == 0)
    {
        logError("Fail! TSTStringUpdate1 \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,inputString);
    if(jstr != 0)
        return (*env)->CallBooleanMethod(env, jTSTObject, jTSTStringUpdate1, jstr);
    else
        return FALSE;
}

int TSTStringUpdate2(char* objectID, char* inputString, long lease)
{
    jobject jTSTObject;
    jstring jstr;

    jTSTObject = getTSTObject((int)TS_STRING, objectID);

    if ( jTSTStringUpdate1 ==0 || jTSTObject == 0)
    {
        logError("Fail! TSTStringUpdate2 \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,inputString);
    return (*env)->CallBooleanMethod(env, jTSTObject, jTSTStringUpdate2, jstr, lease);
}

```

```

// *** Read ***
void TSTStringRead1(char* objectID, char* outputBuf)
{
    const jbyte *str;
    jobject jTSTObject;
    jstring jstr;

    jTSTObject = getTSTObject((int)TS_STRING, objectID);
    if ( jTSTStringRead1 ==0 || jTSTObject == 0)
    {
        logError("Fail! TSTStringRead1 \n");
    }

    jstr = (*env)->CallObjectMethod(env, jTSTObject, jTSTStringRead1);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(outputBuf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(outputBuf," ");
}

```

```

void TSTStringRead2(char* objectID, char* outputBuf, long timeout)
{
    const jbyte *str;
    jobject jTSTObject;
    jstring jstr;

    jTSTObject = getTSTObject((int)TS_STRING, objectID);
    if ( jTSTStringRead2 ==0 || jTSTObject == 0)
    {
        logError("Fail! TSTStringRead2 \n");
    }
}

```

```

    }

    jstr = (*env)->CallObjectMethod(env, jTSObject, jTSStringRead2, timeout);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env, jstr, NULL);
        if(str != NULL)
            sprintf(outputBuf, "%s", str);
        (*env)->ReleaseStringUTFChars(env, jstr, str);
    }
    else
        sprintf(outputBuf, " ");
}

// *** Take ***
void TSStringTake1(char* objectID, char* outputBuf)
{
    const jbyte *str;
    jobject jTSObject;
    jstring jstr;

    jTSObject = getTSObject((int)TS_STRING, objectID);
    if ( jTSStringTake1 == 0 || jTSObject == 0 )
    {
        logError("Fail! TSStringTake1 \n");
    }

    jstr = (*env)->CallObjectMethod(env, jTSObject, jTSStringTake1);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env, jstr, NULL);
        if(str != NULL)
            sprintf(outputBuf, "%s", str);
        (*env)->ReleaseStringUTFChars(env, jstr, str);
    }
    else
        sprintf(outputBuf, " ");
}

void TSStringTake2(char* objectID, char* outputBuf, long timeout)
{
    const jbyte *str;
    jobject jTSObject;
    jstring jstr;

    jTSObject = getTSObject((int)TS_STRING, objectID);
    if ( jTSStringTake2 == 0 || jTSObject == 0 )
    {
        logError("Fail! TSStringTake2 \n");
    }

    jstr = (*env)->CallObjectMethod(env, jTSObject, jTSStringTake2, timeout);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env, jstr, NULL);
        if(str != NULL)
            sprintf(outputBuf, "%s", str);
        (*env)->ReleaseStringUTFChars(env, jstr, str);
    }
    else
        sprintf(outputBuf, " ");
}

// *** Transaction ***
int TSStringTransWrite(char* objectID, char* inputString, long lease)
{
    jobject jTSObject;

```

```

    jstring jstr;

    jobject jTObject = getTObject((int)TS_STRING, objectID);
    if ( jTStringWrite1 ==0 || jTObject == 0 || updateTransactionHandle(
(int)TS_STRING, objectID)== 0)
    {
        logError("Fail! TSStringTransWrite \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,inputString);
    return (*env)->CallBooleanMethod(env, jTObject, jTStringTransWrite, jstr,
lease);
}

int TSStringTransUpdate(char* objectID, char* inputString, long lease)
{
    jobject jTObject;
    jstring jstr;

    jobject jTObject = getTObject((int)TS_STRING, objectID);
    if ( jTStringWrite1 ==0 || jTObject == 0 || updateTransactionHandle( TS_STRING,
objectID)== 0)
    {
        logError("Fail! TSStringUpdate2 \n");
        return FALSE;
    }

    jstr = (*env)->NewStringUTF(env,inputString);
    return (*env)->CallBooleanMethod(env, jTObject, jTStringTransUpdate, jstr,
lease);
}

void TSStringTransRead(char* objectID, char* outputBuf, long timeout)
{
    const jbyte *str;
    jobject jTObject;
    jstring jstr;

    jobject jTObject = getTObject((int)TS_STRING, objectID);
    if ( jTStringWrite1 ==0 || jTObject == 0 || updateTransactionHandle( TS_STRING,
objectID)== 0)
    {
        logError("Fail! TSStringTransRead \n");
    }

    jstr = (*env)->CallObjectMethod(env, jTObject, jTStringTransRead, timeout);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env,jstr,NULL);
        if(str != NULL)
            sprintf(outputBuf,"%s",str);
        (*env)->ReleaseStringUTFChars(env,jstr,str);
    }
    else
        sprintf(outputBuf," ");
}

void TSStringTransTake(char* objectID, char* outputBuf, long timeout)
{
    const jbyte *str;
    jobject jTObject;
    jstring jstr;

    jobject jTObject = getTObject((int)TS_STRING, objectID);
    if ( jTStringWrite1 ==0 || jTObject == 0 || updateTransactionHandle( TS_STRING,
objectID)== 0)
    {
        logError("Fail! TSStringTransTake \n");
    }

```

```

    }

    jstr = (*env)->CallObjectMethod(env, jTSObject, jTSStringTransTake, timeout);
    if(jstr != 0)
    {
        str = (*env)->GetStringUTFChars(env, jstr, NULL);
        if(str != NULL)
            sprintf(outputBuf, "%s", str);
        (*env)->ReleaseStringUTFChars(env, jstr, str);
    }
    else
        sprintf(outputBuf, " ");
}

//*****
//*****
// TSHash Methods
//*****
// *** Write ***
int TSHashWrite1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashWrite1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashWrite1);
}

int TSHashWrite2(char* objectID, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashWrite1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashWrite2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashWrite2, lease);
}

// *** Update ***
int TSHashUpdate1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashUpdate1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashUpdate1);
}

int TSHashUpdate2(char* objectID, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashUpdate1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashUpdate2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashUpdate2, lease);
}

```

```

// *** Read ***
int TSHashRead1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashRead1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashRead1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashRead1);
}

int TSHashRead2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashRead2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashRead2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashRead2, timeout);
}

// *** Take ***
int TSHashTake1(char* objectID)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashTake1 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashTake1 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTake1);
}

int TSHashTake2(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashTake2 ==0 || jTSObject == 0)
    {
        logError("Fail! TSHashTake2 \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTake2, timeout);
}

// *** Transaction ***
int TSHashTransWrite(char* objectID, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashTransWrite ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID) == 0)
    {
        logError("Fail! TSHashTransWrite \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTransWrite, lease);
}

int TSHashTransUpdate(char* objectID, long lease)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);

```

```

        if ( jTSHashTransUpdate ==0 || jTSObject == 0 || updateTransactionHandle(
(int)TS_DOUBLE, objectID)== 0)
        {
            logError("Fail! TSHashTransUpdate \n");
            return FALSE;
        }
        return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTransUpdate, lease);
    }

int TSHashTransRead(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashTransRead ==0 || jTSObject == 0 || updateTransactionHandle( TS_DOUBLE,
objectID)== 0)
    {
        logError("Fail! TSHashTransRead \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTransRead, timeout);
}

int TSHashTransTake(char* objectID, long timeout)
{
    jobject jTSObject;
    jTSObject = getTSObject((int)TS_DOUBLE, objectID);
    if ( jTSHashTransTake ==0 || jTSObject == 0 || updateTransactionHandle( TS_DOUBLE,
objectID)== 0)
    {
        logError("Fail! TSHashTransTake \n");
        return FALSE;
    }
    return (*env)->CallBooleanMethod(env, jTSObject, jTSHashTransTake, timeout);
}

```

APPENDIX E. AGENT API LISTING

A. SERVICE PACKAGE

1. AgentServiceInterface.java

```
// This is the interface that the service's proxy
// implements

package tuplespace.services;

public interface AgentServiceInterface {
    public String getMessage();
    public boolean joinService(String id, String password);
    public boolean endService();
}
```

2. AgentService.java

```
package tuplespace.services;

import net.jini.discovery.DiscoveryListener;
import net.jini.discovery.DiscoveryEvent;
import net.jini.discovery.LookupDiscovery;
import net.jini.core.entry.*;
import net.jini.lookup.entry.*;
import net.jini.core.lookup.ServiceItem;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.lookup.ServiceRegistration;
import net.jini.core.lease.Lease;
import net.jini.core.lease.UnknownLeaseException;
import java.util.Hashtable;
import java.io.IOException;
import java.io.Serializable;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.util.Vector;
import java.util.Enumeration;

// This is the proxy object that will be downloaded
// by clients. It's serializable and implements
// our well-known AgentServiceInterface.
class AgentServiceProxy implements Serializable,
    AgentServiceInterface {
    public AgentServiceProxy() {
    }
    public String getMessage() {
        return "Hello, world!";
    }
    public boolean joinService(String id, String password){
        return true;
    }
    public boolean endService(){
        return true;
    }
}

// AgentService is the "wrapper" class that
// handles publishing the service item.
public class AgentService implements Runnable {
    // 10 minute leases
    protected Thread leaseThread = null;
    protected final int LEASE_TIME = 10 * 60 * 1000;
    protected Hashtable registrations = new Hashtable();
    protected ServiceItem item;
    protected LookupDiscovery disco;

    // Inner class to listen for discovery events
```

```

class Listener implements DiscoveryListener {
    // Called when we find a new lookup service.
    public void discovered(DiscoveryEvent ev) {
        System.out.println("discovered a lookup service!");
        ServiceRegistrar[] newregs = ev.getRegistrars();
        for (int i=0 ; i<newregs.length ; i++) {
            if (!registrations.containsKey(newregs[i])) {
                registerWithLookup(newregs[i]);
            }
        }
    }

    // Called ONLY when we explicitly discard a
    // lookup service, not "automatically" when a
    // lookup service goes down. Once discovered,
    // there is NO ongoing communication with a
    // lookup service.
    public void discarded(DiscoveryEvent ev) {
        ServiceRegistrar[] deadregs = ev.getRegistrars();
        for (int i=0 ; i<deadregs.length ; i++) {
            registrations.remove(deadregs[i]);
        }
    }
}

public AgentService() throws IOException {
    item = new ServiceItem(null, createProxy(), getAttributes());

    // Set a security manager
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    // Search for the "public" group, which by
    // convention is named by the empty string
    disco = new LookupDiscovery(new String[] { "" });

    // Install a listener.
    disco.addDiscoveryListener(new Listener());
}

protected AgentServiceInterface createProxy() {
    return new AgentServiceProxy();
}

// This work involves remote calls, and may take a
// while to complete. Thus, since it's called from
// discovered(), it will prevent us from responding
// in a timely fashion to new discovery events. An
// improvement would be to spin off a separate short-
// lived thread to do the work.
protected synchronized void registerWithLookup(ServiceRegistrar registrar) {
    ServiceRegistration registration = null;

    try {
        registration = registrar.register(item, LEASE_TIME);
    } catch (RemoteException ex) {
        System.out.println("Couldn't register: " + ex.getMessage());
        return;
    }

    // If this is our first registration, use the
    // service ID returned to us. Ideally, we should
    // save this ID so that it can be used after
    // restarts of the service
    if (item.serviceID == null) {
        item.serviceID = registration.getServiceID();
        System.out.println("Set serviceID to " + item.serviceID);
    }
}

```

```

        registrations.put(registrar, registration);
        leaseThread.interrupt();
    }

    // run now maintains our leases
    public void run() {
        while (true) {
            try {
                long sleepTime = computeSleepTime();
                Thread.sleep(sleepTime);
                renewLeases();
            } catch (InterruptedException ex) {
            }
        }
    }

    // Figure out how long to sleep.
    protected synchronized long computeSleepTime() {
        long soonestExpiration = Long.MAX_VALUE;
        Enumeration enum = registrations.elements();
        while (enum.hasMoreElements()) {
            Lease l = ((ServiceRegistration) enum.nextElement()).getLease();
            if (l.getExpiration() - (20 * 1000) < soonestExpiration) {
                soonestExpiration = l.getExpiration() - (20 * 1000);
            }
        }

        long now = System.currentTimeMillis();

        if (now >= soonestExpiration) {
            return 0;
        } else {
            return soonestExpiration - now;
        }
    }

    // Do the work of lease renewal.
    protected synchronized void renewLeases() {
        long now = System.currentTimeMillis();
        Vector deadLeases = new Vector();

        Enumeration keys = registrations.keys();
        while (keys.hasMoreElements()) {
            ServiceRegistrar lu = (ServiceRegistrar) keys.nextElement();
            ServiceRegistration r = (ServiceRegistration) registrations.get(lu);
            Lease l = r.getLease();
            if (now <= l.getExpiration() &&
                now >= l.getExpiration() - (20 * 1000)) {
                try {
                    System.out.println("Renewing lease.");
                    l.renew(LEASE_TIME);
                } catch (Exception ex) {
                    System.err.println("Couldn't renew lease: " +
                                         ex.getMessage());
                    deadLeases.addElement(lu);
                }
            }
        }

        // clean up after any leases that died
        for (int i=0, size=deadLeases.size(); i<size; i++) {
            registrations.remove(deadLeases.elementAt(i));
        }
    }

    protected Entry[] getAttributes(){

```

```

Entry[] entries = new Entry[2];
entries[0] = new ServiceInfo("Tiptop",
                             "Agent",
                             "Agent",
                             " ",
                             "Tiptop",
                             "v1.0");
entries[1] = new Name("Agent");
return entries;
}

// Create the service and start the leasing
// thread.
public static void main(String args[]) {
    try {
        AgentService hws = new AgentService();
        hws.leaseThread = new Thread(hws);
        hws.leaseThread.start();
    } catch (IOException ex) {
        System.out.println("Couldn't create service: " +
                           ex.getMessage());
    }
}

```

B. CORE PACKAGE

1. Agent.java

```
package tuplespace.core;

import java.net.URLClassLoader;
import java.net.URL;
import java.net.MalformedURLException ;
import tuplespace.entries.*;
import net.jini.core.entry.UnusableEntryException;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;
import net.jini.space.JavaSpace;
import java.util.*;
import java.awt.event.*;

/**
 * The <code>Agent</code> class implements the methods to
 * configure agent properties, establish connection with Jini Services,
 * request for transaction and create new entry handlers
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */
public class Agent implements java.io.Serializable, SpaceActionHandler, TSConstants
{
    private JavaSpace space;
    private TransactionManager txnMgr;
    private Transaction txn;
    private TSList tsList;
    private TSStack tsStack;
    private Vector pushListeners = new Vector();
    private Map tsStringMap;
    private Map tsBooleanMap;
    private Map tsIntegerMap;
    private Map tsFloatMap;
    private Map tsLongMap;
    private Map tsDoubleMap;
    private Map tsQueueMap;
    private Map tsStackMap;
    private Map tsLinkListMap;
    private Map tsHashMap;

    public Agent()
    {
        System.setProperty("java.security.policy", "d:\\jini1_1\\example\\books\\policy.all");
        System.setProperty("outrigger.spacename", "JavaSpaces");
        //System.setProperty("com.sun.jini.lookup.locator", null); // unicast set
        property to a proper jini URL
        System.setProperty("com.sun.jini.lookup.groups", "public");
        System.setProperty("java.rmi.server.codebase", "http://tiptop:8081/entries.jar");
        System.out.println(" *** Default System properties ***");
        System.out.println("      1. Security Policy : " +
            System.getProperty("java.security.policy"));
        System.out.println("      2. Space Name : " +
            System.getProperty("outrigger.spacename"));
        System.out.println("      3. Server Codebase : " +
            System.getProperty("java.rmi.server.codebase"));
        System.out.println("      4. Lookup URL : " +
            System.getProperty("com.sun.jini.lookup.locator"));
        System.out.println("      5. Lookup groups : " +
            System.getProperty("com.sun.jini.lookup.groups"));
        System.out.println("... Agent Loaded ! but still not started");

        //InitAgent(10000);
    }
}
```

```

    txn = null;          // Disable Transaction
    addActionListener(this);

    //Initialize Hash Maps
    tsStringMap = new HashMap();
    tsBooleanMap = new HashMap();
    tsIntegerMap = new HashMap();
    tsFloatMap = new HashMap();
    tsLongMap = new HashMap();
    tsDoubleMap = new HashMap();
    tsQueueMap = new HashMap();
    tsStackMap = new HashMap();
    tsLinkedListMap = new HashMap();
    tsHashMap = new HashMap();
}

/** look for the Jini/Space and test whether the interface is functioning
 * @return <tt>true</tt> if agent is successfully initialized. */

public boolean InitAgent(long timeout)
{
    System.out.println("Security Policy : " +
    System.getProperty("java.security.policy"));
    System.out.println("Space Name : " +
    System.getProperty("outrigger.spacename"));
    System.out.println("Server Codebase : " +
    System.getProperty("java.rmi.server.codebase"));
    System.out.println("Lookup URL : " +
    System.getProperty("com.sun.jini.lookup.locator"));
    System.out.println("Lookup groups : " +
    System.getProperty("com.sun.jini.lookup.groups"));

    space = null;
    txnMgr = null;
    try{
        if( ServiceAccessor.getLocator( timeout) != null){
            space = ServiceAccessor.getSpace(getAgentSpaceName());
            txnMgr = ServiceAccessor.getManager();
            if(space == null || txnMgr == null)
                return false;
            else
                return true;
        }
    } catch (Exception e){
        e.printStackTrace();
    }
    return false;
}

public void TerminateAgent()
{
    Iterator it;
    for (it=tsStringMap.keySet().iterator(); it.hasNext(); ) {
        removeID( TS_STRING, (String) it.next());
    }
    for (it=tsBooleanMap.keySet().iterator(); it.hasNext(); ) {
        removeID( TS_BOOLEAN, (String) it.next());
    }
    for (it=tsIntegerMap.keySet().iterator(); it.hasNext(); ) {
        removeID( TS_INTEGER, (String) it.next());
    }
    for (it=tsFloatMap.keySet().iterator(); it.hasNext(); ) {
        removeID( TS_FLOAT, (String) it.next());
    }
    for (it=tsLongMap.keySet().iterator(); it.hasNext(); ) {
        removeID( TS_LONG, (String) it.next());
    }
}

```

```

        for (it=tsDoubleMap.keySet().iterator(); it.hasNext(); ) {
            removeID( TS_DOUBLE, (String) it.next());
        }
        for (it=tsQueueMap.keySet().iterator(); it.hasNext(); ) {
            removeID( TS_QUEUE, (String) it.next());
        }
        for (it=tsStackMap.keySet().iterator(); it.hasNext(); ) {
            removeID( TS_STACK, (String) it.next());
        }
        for (it=tsHashMap.keySet().iterator(); it.hasNext(); ) {
            removeID( TS_HASH, (String) it.next());
        }

        //System.runFinalizersOnExit(0);
        System.exit (0);
    }

    /** returns the spaceAction handle
    * @return <tt>null</tt> if space action handle is invalid. */

    public SpaceActionHandler getActionHandler(){
        return this;
    }

    /** start transaction manager; transaction handle will remain valid
    * for a maximum of 5 minutes (Default setting)
    * @return <tt>false</tt> if transaction manager false to start. */
    public boolean startTransaction(){
        if(txnMgr != null){
            Transaction.Created trc = null;
            try {
                trc = TransactionFactory.create(txnMgr, 300000);
                txn = trc.transaction;

                return true;
            } catch (Exception e) {
                System.err.println("Could not create transaction " + e);
                return false;
            }
        }else
            return false;
    }

    /** returns the transaction handle; transaction manager must be
    * initialized and started
    * @return <tt>null</tt> if transaction handle is invalid. */

    public Transaction getTransaction(){
        return txn;
    }

    /** returns the current state of transaction handle
    * @return <tt>true</tt> if transaction handle is valid. */
    public boolean isTransactionStarted(){
        if(txn != null)
            return true;
        else
            return false;
    }

    /** aborts the current transaction; all the commands issued with
    * transaction after the start transaction will be rolled back.
    * @return <tt>true</tt> if transaction is successfully aborted. */
    public boolean abortTransaction(){
        try {
            txn.abort();
            txn = null;
            return true;
        } catch (Exception e) {

```

```

        return false;
    }
}

/** closes the current transaction; all the commands issued with
 * transaction after the start transaction will become active.
 * @return <tt>true</tt> if transaction is successfully closed. */
public boolean closeTransaction(){
    try {
        txn.commit();
        txn = null;
        return true;
    } catch (Exception e) {
        return false;
    }
}

/** update the transaction handle in TSObject
 * @param type the type of entry to create. E.g EntryInteger
 * @param id a ID that identifies this entry
 * @return <tt>true</tt> if entry is successfully created. */
public boolean updateTransactionHandle(int type, String id){
    TSBase obj = (TSBase) getTSObject(type, id);
    if(obj == null)
        return false;

    obj.setTransactionHandler(txn);
    return true;
}

//*****
//*** System Property Methods
//***
//*****

/** Set Agent Security Policy path and filename*/
public void setAgentSecurityPolicy(String str){
    System.setProperty("java.security.policy",str);
}

/** Returns Agent Security Policy path and filename*/
public String getAgentSecurityPolicy(){
    return System.getProperty("java.security.policy");
}

/** Set Agent Space Name */
public void setAgentSpaceName(String str){
    System.setProperty("outrigger.spacename",str);
}

/** Returns Agent Space Name */
public String getAgentSpaceName(){
    return System.getProperty("outrigger.spacename");
}

/** Set Agent Server Codebase */
public void setAgentServerCodebase(String str){
    System.setProperty("java.rmi.server.codebase",str);
}

/** Returns Agent Server Codebase */
public String getAgentServerCodebase(){
    return System.getProperty("java.rmi.server.codebase");
}

/** Set Agent Lookup Group */

```

```

public void setAgentLookupGroup(String str){
    System.setProperty("com.sun.jini.lookup.groups", str);
}

/** Returns Agent Lookup Group */
public String getAgentLookupGroup(){
    return System.getProperty("com.sun.jini.lookup.groups");
}

/** Set Agent Lookup URL */
public void setAgentLookupURL(String str){
    if(str != null || str.length() > 0)
        System.setProperty("com.sun.jini.lookup.locator", str);
}

/** Returns Agent Lookup URL */
public String getAgentLookupURL(){
    return System.getProperty("com.sun.jini.lookup.locator");
}

//*****
// *** IDs
//*****

/** Create a new entry in agent
 * @param type the type of entry to create. E.g EntryInteger
 * @param id an unique ID that identifies this entry
 * @return <tt>true</tt> if entry is successfully created. */
public boolean createID(int type, String id){
    switch(type){
        case TS_BOOLEAN:
            if(!tsBooleanMap.containsKey(id)){
                TSBoolean ts = new TSBoolean(space, this, id);
                tsBooleanMap.put(id, ts);
                return true;
            }
            break;
        case TS_INTEGER:
            if(!tsIntegerMap.containsKey(id)){
                TSInteger ts = new TSInteger(space, this, id);
                tsIntegerMap.put(id, ts);
                return true;
            }
            break;
        case TS_FLOAT:
            if(!tsFloatMap.containsKey(id)){
                TSFloat ts = new TSFloat(space, this, id);
                tsFloatMap.put(id, ts);
                return true;
            }
            break;
        case TS_LONG:
            if(!tsLongMap.containsKey(id)){
                TSLong ts = new TSLong(space, this, id);
                tsLongMap.put(id, ts);
                return true;
            }
            break;
        case TS_DOUBLE:
            if(!tsDoubleMap.containsKey(id)){
                TSDouble ts = new TSDouble(space, this, id);
                tsDoubleMap.put(id, ts);
                return true;
            }
    }
}

```

```

        break;
    case TS_STRING:
        if(!tsStringMap.containsKey(id)){
            TSString ts = new TSString(space, this, id);
            tsStringMap.put(id, ts);
            return true;
        }
        break;
    case TS_QUEUE:
        if(!tsQueueMap.containsKey(id)){
            TSQueue ts = new TSQueue(space, this, id, 100);
            tsQueueMap.put(id, ts);
            return true;
        }
        break;

    case TS_STACK:
        if(!tsStackMap.containsKey(id)){
            TSStack ts = new TSStack(space, id);
            tsStackMap.put(id, ts);
            return true;
        }
        break;
    case TS_LINKLIST:
        break;
    case TS_HASH:
        if(!tsHashMap.containsKey(id)){
            TSHash ts = new TSHash(space, this, id);
            tsHashMap.put(id, ts);
            return true;
        }
        break;
    }
    return false;
}

/** Remove an existing TSString ID */
public boolean removeID(int type, String id){
    switch(type){
        case TS_BOOLEAN:
            if(tsBooleanMap.containsKey(id)){
                TSBoolean ts = (TSBoolean) tsBooleanMap.get(id);
                ts.stopEvent();
                tsBooleanMap.remove(id);
                return true;
            }
            break;
        case TS_INTEGER:
            if(tsIntegerMap.containsKey(id)){
                TSInteger ts = (TSInteger) tsIntegerMap.get(id);
                ts.stopEvent();
                tsIntegerMap.remove(id);
                return true;
            }
            break;
        case TS_FLOAT:
            if(tsFloatMap.containsKey(id)){
                TSFloat ts = (TSFloat) tsFloatMap.get(id);
                ts.stopEvent();
                tsFloatMap.remove(id);
                return true;
            }
            break;
        case TS_LONG:
            if(tsLongMap.containsKey(id)){
                TSLong ts = (TSLong) tsLongMap.get(id);
                ts.stopEvent();
                tsLongMap.remove(id);
                return true;
            }
    }
}

```

```

    }
    break;
case TS_DOUBLE:
    if(tsDoubleMap.containsKey(id)){
        TSDouble ts = (TSDouble) tsDoubleMap.get(id);
        ts.stopEvent();
        tsDoubleMap.remove(id);
        return true;
    }
    break;
case TS_STRING:
    if(tsStringMap.containsKey(id)){
        TSString ts = (TSString) tsStringMap.get(id);
        ts.stopEvent();
        tsStringMap.remove(id);
        return true;
    }
    break;
case TS_QUEUE:
    if(tsQueueMap.containsKey(id)){
        tsQueueMap.remove(id);
        return true;
    }
    break;
case TS_STACK:
    if(tsStackMap.containsKey(id)){
        tsStackMap.remove(id);
        return true;
    }
    break;
case TS_LINKLIST:
    break;
case TS_HASH:
    if(tsHashMap.containsKey(id)){
        TSHash ts = (TSHash) tsHashMap.get(id);
        ts.stopEvent();
        tsHashMap.remove(id);
        return true;
    }
    break;
}
return false;
}

/** Remove an existing TSString ID */
public Object getTSObject(int type, String id){
    switch(type){
        case TS_BOOLEAN:
            if(tsBooleanMap.containsKey(id))
                return tsBooleanMap.get(id);
            break;
        case TS_INTEGER:
            if(tsIntegerMap.containsKey(id))
                return tsIntegerMap.get(id);
            break;
        case TS_FLOAT:
            if(tsFloatMap.containsKey(id)){
                return tsFloatMap.get(id);
            }
            break;
        case TS_LONG:
            if(tsLongMap.containsKey(id)){
                return tsLongMap.get(id);
            }
            break;
        case TS_DOUBLE:
            if(tsDoubleMap.containsKey(id)){
                return tsDoubleMap.get(id);
            }
    }
}

```

```

        break;
    case TS_STRING:
        if(tsStringMap.containsKey(id))
            return tsStringMap.get(id);
        break;
    case TS_QUEUE:
        if(tsQueueMap.containsKey(id))
            return tsQueueMap.get(id);
        break;
    case TS_STACK:
        if(tsStackMap.containsKey(id))
            return tsStackMap.get(id);
        break;
    case TS_LINKLIST:
        break;
    case TS_HASH:
        if(tsHashMap.containsKey(id))
            return tsHashMap.get(id);
        break;
    default:
        break;
}
System.out.println("Cannot find TSClass! Type:" +type+ " ID:" + id);
return null;
}

/** Creates a new TSString ID */
public boolean cleanTSClass(int type, String id){
    switch(type){
        case TS_BOOLEAN:
            if(tsBooleanMap.containsKey(id)){
                TSBoolean ts = (TSBoolean) tsBooleanMap.get(id);
                ts.cleanSpace();
                return true;
            }
            break;
        case TS_INTEGER:
            if(tsIntegerMap.containsKey(id)){
                TSInteger ts = (TSInteger) tsIntegerMap.get(id);
                ts.cleanSpace();
                return true;
            }
            break;
        case TS_FLOAT:
            if(tsFloatMap.containsKey(id)){
                TSFloat ts = (TSFloat) tsFloatMap.get(id);
                ts.cleanSpace();
                return true;
            }
            break;
        case TS_LONG:
            if(tsLongMap.containsKey(id)){
                TSLong ts = (TSLong) tsLongMap.get(id);
                ts.cleanSpace();
                return true;
            }
            break;
        case TS_DOUBLE:
            if(tsDoubleMap.containsKey(id)){
                TSDouble ts = (TSDouble) tsDoubleMap.get(id);
                ts.cleanSpace();
                return true;
            }
            break;
        case TS_STRING:
            if(tsStringMap.containsKey(id)){
                TSString ts = (TSString) tsStringMap.get(id);
                ts.cleanSpace();
            }
    }
}

```

```

        return true;
    }
    break;
case TS_QUEUE:
    if(tsQueueMap.containsKey(id)){
        //TSQueue ts = (TSQueue) tsQueueMap.get(id);
        //ts.cleanSpace();
        return true;
    }
    break;

case TS_STACK:
    if(tsStackMap.containsKey(id)){
        TSStack ts = (TSStack) tsStackMap.get(id);
        //ts.cleanSpace();
        return true;
    }
    break;
case TS_LINKLIST:
    break;
case TS_HASH:
    if(tsHashMap.containsKey(id)){
        TSHash ts = (TSHash) tsHashMap.get(id);
        //ts.cleanSpace();
        return true;
    }
    break;
}
System.out.println("Cannot find TSClass! Type:" +type+ " ID:" + id);
return false;
}

```

```

/** Returns all TSString IDs created */
public String getTSClassIDs(int type){
    Map map = null;
    switch(type){
        case TS_BOOLEAN:
            map = tsBooleanMap;
            break;
        case TS_INTEGER:
            map = tsIntegerMap;
            break;
        case TS_FLOAT:
            map = tsFloatMap;
            break;
        case TS_LONG:
            map = tsLongMap;
            break;
        case TS_DOUBLE:
            map = tsDoubleMap;
            break;
        case TS_STRING:
            map = tsStringMap;
            break;
        case TS_QUEUE:
            map = tsQueueMap;
            break;
        case TS_STACK:
            map = tsStackMap;
            break;
        case TS_LINKLIST:
            map = tsIntegerMap;
            break;
        case TS_HASH:
            map = tsHashMap;
            break;
    }
}

```

```

if(map != null){
    String tmpStr = null;
    int i =0;
    for (Iterator it=map.keySet().iterator() ;it.hasNext(); ) {
        if(tmpStr == null)
            tmpStr = (String) it.next();
        else
            tmpStr = tmpStr + "," + it.next();
        //print(" Key["+ i++ + "] -> " + tmpStr);
    }
    return tmpStr;
}else
    return null;
}

//*****
// *** Miscellaneous Methods
//*****

public void actionPerformed(ActionEvent e){
    print("action ID :[" + e.getID() + "] Command:[" + e.getActionCommand() + "]  ");
}

public synchronized void addActionListener(ActionListener l) {
    pushListeners.addElement(l);
}

public synchronized void removeActionListener(ActionListener l) {
    pushListeners.removeElement(l);
}

public void fireAction(int entryType, String entryID) {
    Vector targets;
    synchronized (this) {
        targets = (Vector) pushListeners.clone();
    }
    ActionEvent actionEvt = new ActionEvent(this, entryType, entryID);
    for (int i = 0; i < targets.size(); i++) {
        ActionListener target = (ActionListener)targets.elementAt(i);
        target.actionPerformed(actionEvt);
        //print("FireAction triggered!");
    }
}

public void print(String str){
    System.out.println(str);
}

public int getTSType(String type){
    if( type.compareTo("BOOLEAN") == 0)
        return TS_BOOLEAN;
    else if( type.compareTo("INTEGER") == 0)
        return TS_INTEGER;
    else if( type.compareTo("FLOAT") == 0)
        return TS_FLOAT;
    else if( type.compareTo("LONG") == 0)
        return TS_LONG;
    else if( type.compareTo("DOUBLE") == 0)
        return TS_DOUBLE;
    else if( type.compareTo("STRING") == 0)
        return TS_STRING;
    else if( type.compareTo("QUEUE") == 0)
        return TS_QUEUE;
    else if( type.compareTo("STACK") == 0)

```

```

        return TS_STACK;
    else if( type.compareTo("LINKLIST") == 0)
        return TS_LINKLIST;
    else if( type.compareTo("HASH") == 0)
        return TS_HASH;
    else
        return 0;
}

public void SearchTSClassIDs()
{
    System.out.println("Start Search");
    startTransaction();
    searchSpace(TS_STRING);
    searchSpace(TS_INTEGER);
    searchSpace(TS_LONG);
    searchSpace(TS_DOUBLE);

    abortTransaction();
}

private void searchSpace(int type)
{
    boolean breakLoop = false;
    do{
        try{
            switch(type){
                case TS_BOOLEAN:
                    EntryBoolean test0 = (EntryBoolean) space.takeIfExists(new
EntryBoolean(),txn,1000);
                    createID(type, test0.entryID);
                    break;
                case TS_INTEGER:
                    EntryInteger test1 = (EntryInteger) space.takeIfExists(new
EntryInteger(),txn,1000);
                    createID(type, test1.entryID);
                    break;
                case TS_FLOAT:
                    EntryFloat test2 = (EntryFloat) space.takeIfExists(new
EntryFloat(),txn,1000);
                    createID(type, test2.entryID);
                    break;
                case TS_LONG:
                    EntryLong test3 = (EntryLong) space.takeIfExists(new
EntryLong(),txn,1000);
                    createID(type, test3.entryID);
                    break;
                case TS_DOUBLE:
                    EntryDouble test4 = (EntryDouble) space.takeIfExists( new EntryDouble()
,txn,1000);
                    createID(type, test4.entryID);
                    break;
                case TS_STRING:
                    EntryString test5 = (EntryString) space.takeIfExists(new
EntryString(),txn,1000);
                    createID(type, test5.entryID);
                    break;
                case TS_QUEUE:
                    EntryQueueItem test6 = (EntryQueueItem) space.takeIfExists(new
EntryQueueItem(),txn,1000);
                    createID(type, test6.entryID);
                    break;
                case TS_STACK:
                    //EntryStack test = (EntryStack) space.takeIfExists((EntryStack)
template,null,1000);
                    //createID(type, test.entryID);
                    break;
                case TS_LINKLIST:

```

```

        //test = space.takeIfExists((EntryLinkList) template,null,1000);
        //createID(type,(EntryLinkList) test.entryID);
        break;
    case TS_HASH:
        EntryHash      test7      =      (EntryHash)space.takeIfExists(new
EntryHash(),txn,1000);
        createID(type, test7.entryID);
        break;
    }
    }catch (Exception e){
        breakLoop = true;
    }
    }while(breakLoop == false);
}
}

```

2. ServiceFinder.java

```
package tuplespace.core;

import net.jini.core.lookup.*;
import net.jini.discovery.*;
import net.jini.core.entry.*;
import net.jini.lookup.entry.Name;
import com.sun.jini.mahout.Locator;
import com.sun.jini.outrigger.Finder;
import com.sun.jini.outrigger.DiscoveryLocator;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.AccessException;
import java.util.Iterator;

/**
 * A <code>ServiceFinder</code> implements the
 * methods needed locate a service in a Jini(tm) Lookup service.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */
public class ServiceFinder extends Finder {

    private ServiceRegistrar lookup;
    private int retry;
    private static final boolean DEBUG = false;
    private static final int MAX_DISCOVERY_RETRY = 10;
    private static final byte PACKET_TTL = 100;

    //Allow us to easily choose whether or not we want
    //to use the supplied means of locating the lookup
    //within the djinn.

    /**
     * Create a new <code>LookupFinder</code> object
     */
    public ServiceFinder() {
    }

    /**
     * Using the Jini lookup service returned by <code>locator</code>
     * find the service registered with a
     * <code>net.jini.lookup.entry.Name</code> attribute who's value
     * is <code>name</code>. If no service is registered under the
     * specified name retry until such a service appears. <p> This
     * method returns null if the lookup service can not be contacted.
     */
    public Object find (Locator locator, String name) {
        Object tmpobj = null;

        try {
            if (!(locator instanceof DiscoveryLocator))
                throw new ClassCastException("LookupFinder: find: " +
                                                "locator must be a LookupLocator");

            //by casting to ServiceRegistrar here, we
            //are implicitly checking if what is eventually
            //returned by find() is really an instance of
            //ServiceRegistrar.

            ServiceRegistrar registrar = (ServiceRegistrar)locator.locate();

            Entry[] attrs = new Entry[1];
            Name n = new Name();
            n.name = name;
            attrs[0] = n;
        }
    }
}
```

```

        ServiceTemplate tmp1 =
        new ServiceTemplate(null, null, attrs);

        if (DEBUG) {
            System.out.println("LookupFinder: find: name = " + name);
            System.out.println("LookupFinder: find: registrar = " +
                               registrar);
            System.out.println("LookupFinder: find: tmp1 = " + tmp1);
        }

        retry = 0;
        do {
            tmpobj = registrar.lookup(tmp1);
            if (tmpobj == null) {
                try {
                    System.out.println("waiting for " + name);
                    Thread.sleep(2000);
                } catch (Exception te) {
                }
            }
            retry++;
        } while (tmpobj == null && retry < 10);
    } catch (Exception e) {
        System.out.println("LookupLocator: find: " + e.getMessage());
        e.printStackTrace();
    }

    System.out.println("found " + name + " = " + tmpobj);
    return tmpobj;
}
}

```

3. ServiceAccessor.java

```
package tuplespace.core;

import tuplespace.services.AgentServiceInterface;
import java.rmi.*;
import java.awt.event.*;
import net.jini.space.JavaSpace;

import net.jini.core.transaction.server.TransactionManager;
import com.sun.jini.mahalo.TxnManagerImpl;

import com.sun.jini.mahout.binder.RefHolder;
import com.sun.jini.mahout.Locator;
import com.sun.jini.outtrigger.Finder;
import com.sun.jini.*;

/**
 * The <code>ServiceAccessor</code> class implements the methods for
 * registering the Services
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */

public class ServiceAccessor implements java.io.Serializable{
    private static Locator locator;

    public static Locator getLocator( long lookupTimeout){
        try {

            if (System.getSecurityManager() == null) {
                System.setSecurityManager( new RMISecurityManager());
                System.out.println(" Running RMISecurity Manager");
            }

            locator = new com.sun.jini.outtrigger.DiscoveryLocator(lookupTimeout);
            return locator;
        } catch (Exception e) {
            locator = null;
            System.err.println(e.getMessage());
        }
        return null;
    }

    public static JavaSpace getSpace() {
        return getSpace("JavaSpaces");
    }

    public static JavaSpace getSpace(String name) {
        try {

            if (System.getSecurityManager() == null) {
                System.setSecurityManager( new RMISecurityManager());
                System.out.println(" Running RMISecurity Manager");
            }

            if (System.getProperty("com.sun.jini.use.registry") == null)
            {
                if(locator != null){
                    Finder finder = new ServiceFinder();
                    return (JavaSpace)finder.find(locator, name);
                }
            }
            else {
                RefHolder rh = (RefHolder)Naming.lookup(name);
                return (JavaSpace)rh.proxy();
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

```

    }
    return null;
}

public static TransactionManager getManager() {
    return getManager(com.sun.jini.mahalo.TxnManagerImpl.DEFAULT_NAME);
}

public static TransactionManager getManager(String name) {
    try {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager( new RMISecurityManager());
            System.out.println(" Running RMISecurity Manager");
        }

        if (System.getProperty("com.sun.jini.use.registry") == null)
        {
            if(locator != null){
                Finder finder = new ServiceFinder();
                return (TransactionManager)finder.find(locator, name);
            }
        } else {
            RefHolder rh = (RefHolder)Naming.lookup(name);
            return (TransactionManager)rh.proxy();
        }
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
    return null;
}
}

```

4. SpaceEventRegistration.java

```
package tuplespace.core;

import tuplespace.entries.*;
import net.jini.core.transaction.*;

/**
 * The <code>SpaceEventRegistration</code> class implements the methods for
 * for registering with Space Service to monitor new entries written into
 * the space.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */

public interface SpaceEventRegistration
{
    public abstract boolean startEvent();
    public abstract boolean startEvent1(SpaceActionHandler spaceAction);
    public abstract boolean startEvent2(SpaceActionHandler spaceAction, long lease );
    public abstract boolean startEvent3(SpaceActionHandler spaceAction, long lease ,
    Transaction txn);
    public abstract boolean stopEvent();
}
```

TSConstants.java

```
package tuplespace.core;

public interface TSConstants {
    public static final int TS_INTEGER = 5000;
    public static final int TS_FLOAT = 5001;
    public static final int TS_LONG = 5002;
    public static final int TS_DOUBLE = 5003;
    public static final int TS_STRING = 5004;
    public static final int TS_QUEUE = 5005;
    public static final int TS_STACK = 5006;
    public static final int TS_LINKLIST = 5007;
    public static final int TS_HASH = 5008;
    public static final int TS_BOOLEAN = 5009;
}
```

5. TSBase.java

```
package tuplespace.core;

import tuplespace.entries.*;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSBase</code> class implements the methods for setting the
 * attribute of entry. Every entry handler will inherit this class
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 */

public class TSBase implements java.io.Serializable{
    protected JavaSpace space;
    protected long writeLeaseTime, updateLeaseTime, notifyLeaseTime;
    protected long readTimeOut, takeTimeOut;
    protected String entryID;
    protected EventRegistration eventRegistration;
    protected SpaceActionHandler spaceAction;
    protected Transaction transaction;
    protected boolean result;

    public TSBase() {
        System.out.println("TSBase Constructor");
        writeLeaseTime = Lease.FOREVER;
        updateLeaseTime = Lease.FOREVER;
        notifyLeaseTime = 10000; // 1 minute
        readTimeOut = Long.MAX_VALUE;
        takeTimeOut = Long.MAX_VALUE;
        transaction = null;
        eventRegistration = null;
        spaceAction = null;
        space = null;
        result = false;
    }

    public void initTSBase(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        this.space = space;
        this.entryID = entryID;
        this.spaceAction = spaceAction;
    }

    /**
     * *****
     * Change Entry Setting
     * *****
     */
    /** set space
     * @param space JavaSpace handle */
    public void setJavaSpace(JavaSpace space){
        this.space = space;
    }

    /** set action listener
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method; called when remote event is raised. */
    public void setSpaceActionHandler( SpaceActionHandler spaceAction){
        this.spaceAction = spaceAction;
    }

    /** set action listener
```

```

* @param spaceAction the class that implements the actionPerformed(ActionEvent e)
*   method; called when remote event is raised. */
public void setTransactionHandler( Transaction trans){
    this.transaction = trans;
}

public Transaction getTransactionHandler(){
    return this.transaction;
}

/** set write lease time
* @param leaseTime the amount of time for entry to remain valid in space;
*   used when writing entry to space */
public void setWriteLeaseTime(long leaseTime){
    this.writeLeaseTime = leaseTime;
}

/** get write lease time
* @return the current time setting for entry to remain valid in space*/
public long getWriteLeaseTime(){
    return this.writeLeaseTime;
}

/** set update lease time
* @param leaseTime the amount of time for entry to remain valid in space;
*   used when updating an existing entry in space */
public void setUpdateLeaseTime(long leaseTime){
    this.updateLeaseTime = leaseTime;
}

/** get update lease time
* @return the current time setting for entry to remain valid in space*/
public long getUpdateLeaseTime(){
    return this.updateLeaseTime;
}

/** set read time out
* @param timeOut the maximum waiting time when reading an entry from space*/
public void setReadTimeOut(long timeOut){
    this.readTimeOut = timeOut;
}

/** get read time out
* @return the current waiting time setting for reading an entry from space*/
public long getReadTimeOut(){
    return this.readTimeOut;
}

/** set take time out
* @param timeOut the maximum waiting time when taking an entry from space*/
public void setTakeTimeOut(long timeOut){
    this.takeTimeOut = timeOut;
}

/** get take time out
* @return the current waiting time setting for taking an entry from space*/
public long getTakeTimeOut(){
    return this.takeTimeOut;
}

/** For debugging purposes */
protected void print(String str){
    System.out.println(str);
}

public void setResult(boolean value){
    result = value;
}

```

```
    public boolean getResult(){
        return result;
    }
```

6. TSConstants

```
package tuplespace.core;
```

```
public interface TSConstants {
    public static final int TS_INTEGER = 5000;
    public static final int TS_FLOAT = 5001;
    public static final int TS_LONG = 5002;
    public static final int TS_DOUBLE = 5003;
    public static final int TS_STRING = 5004;
    public static final int TS_QUEUE = 5005;
    public static final int TS_STACK = 5006;
    public static final int TS_LINKLIST = 5007;
    public static final int TS_HASH = 5008;
    public static final int TS_BOOLEAN = 5009;
}
```

TSBoolean.java

```

package tuplespace.core;
import tuplespace.entries.*;
import tuplespace.entries.SpaceActionHandler;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSBoolean</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryBoolean entry from space. Every
 * EntryBoolean entry in the space is identified by an unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 * @see EntryBoolean
 */

public class TSBoolean extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryBoolean entry, entryTemplate;

    /** Constructor for TSBoolean
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is raised.
     * @param entryID the unique ID that identifies the Boolean Entry
     */
    public TSBoolean(JavaSpace space, SpaceActionHandler spaceAction, String entryID) {
        super();
        initTSBoolean(space, spaceAction, entryID);
    }

    /** initialize TSBoolean
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method; called when remote event is raised.
     * @param entryID the unique ID that identifies the Boolean Entry
     * @return <tt>true</tt> if internal states are sucessfully reinitialized.
     */
    public boolean initTSBoolean(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        try{
            initTSBase(space, spaceAction, entryID);
            entry = new EntryBoolean(entryID);
            entryTemplate = new EntryBoolean(entryID);
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Remove all existing entries from space */
    public void cleanSpace()
    {
        cleanSpace1(null);
    }

    /** Remove all existing entries from space
     * @param txn a valid transaction handle */
    public void cleanSpace1(Transaction txn)
    {
        Object test;

```

```

do{
    try{
        test = space.takeIfExists(entryTemplate,txn,1000);
        //print("remove [" + entryID + "] from space");
    }catch (Exception e){
        e.printStackTrace();
        test = null;
    }
}while(test != null);
}

//*****
//*** Write Entry
***
//*****

/** Writes value to space
 * @param value the entry value (Boolean)
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write1(boolean value){
    return write3( value, this.writeLeaseTime, null);
}

/** Writes value to space
 * @param value the boolean value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write2(boolean value, long lease){
    return write3( value, lease, null);
}

/** Writes value to space
 * @param value the boolean value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write3(boolean value, long lease, Transaction txn){
    try{
entry = new EntryBoolean(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Update Entry
***
//*****

/** Updates an existing entry value
 * @param value the entry value (Boolean) to be updated
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update1(boolean value) {
    return update3( value, this.updateLeaseTime, null);
}

/** Updates an existing entry value
 * @param value the boolean value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update2(boolean value, long lease) {
    return update3( value, lease, null);
}

/** Updates an existing entry value
 * @param value the boolean value that is to be updated

```

```

* @param lease the amount of time entry is placed in space
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>true</tt> if entry is successfully updated. */
public boolean update3(boolean value, long lease, Transaction txn)
{
try{
    cleanSpace1(txn);
    entry = new EntryBoolean(this.entryID, value);
    space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Read Entry
***
//*****

/** Reads an entry value
* @return <tt>false</tt> if entry is not available, otherwise
* the entry value. */
public boolean readIfExists()
{
    return readIfExists2(this.readTimeOut, null);
}

/** Reads an entry value
* @param timeOut the maximum waiting time when reading the entry from space
* @return <tt>false</tt> if entry is not available, otherwise
* the entry value. */
public boolean readIfExists1(long timeOut)
{
    return readIfExists2(timeOut, null);
}

/** Reads an entry value
* @param timeOut the maximum waiting time when reading the entry from space
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>false</tt> if entry is not available, otherwise
* the entry value. */
public boolean readIfExists2(long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryBoolean) space.readIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryBoolean.booleanValue();
        }
        else
            return false;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Take Entry
***
//*****

```

```

/** Takes an entry value; entry will be removed from space
 * @return <tt>false</tt> if entry is not available, otherwise
 * the entry value. */
public boolean takeIfExists()
{
    return takeIfExists1(this.takeTimeout());
}

/** Takes an entry value; entry will be removed from space
 * @return <tt>false</tt> if entry is not available, otherwise
 * the entry value. */
public boolean takeIfExists1( long timeout)
{
    return takeIfExists2(timeout, null);
}

/** Takes an entry value; entry will be removed from space
 * @param timeout the maximum waiting time when reading the entry
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>false</tt> if entry is not available, otherwise
 * the entry value. */
public boolean takeIfExists2( long timeout, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryBoolean) space.takeIfExists(entryTemplate, txn, timeout);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryBoolean.booleanValue();
        }
        else
            return false;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Notify
//***
//*****

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent()
{
    return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent1(SpaceActionHandler spaceAction)
{
    return startEvent3(spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active

```

```

* @return <tt>true</tt> if notification is enabled. */
public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
{
    return startEvent3(spaceAction, lease , null);
}

/** Start notification; remote event will be raised if any entry
* that matches the entry ID is added into the space.
* @param spaceAction the class that implements the actionPerformed(ActionEvent e)
* method; called when remote event is raised.
* @param lease the amount of time for notification to remain active
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>true</tt> if notification is enabled. */
public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
{
    try{
        // Remove earlier event notification
        if(eventRegistration != null)
            eventRegistration.getLease().cancel();

        // Register new event notification
        SpaceEventListener listener = new SpaceEventListener(TS_BOOLEAN, entryID,
space, spaceAction);
        eventRegistration = space.notify(entryTemplate, txn, listener, lease, null);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Stop notification
* @return <tt>true</tt> if notification is disabled. */
public boolean stopEvent(){
    try{
        if( eventRegistration != null)
        {
            eventRegistration.getLease().cancel();
            eventRegistration = null;
        }
        return true;
    }
    catch (Exception e){
        return false;
    }
}

//*****
/** Transaction Methods
***
//*****

/** Writes entry value with transaction
* @param value the boolean value that is to be written to space
* @param lease the amount of time entry is placed in space
* @param this.transaction must be a valid transaction handle
* @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transWrite(boolean value, long lease)
{
    return write3(value, lease, this.getTransactionHandler());
}

/** Updates an existing entry value with transaction
* @param value the boolean value that is to be written to space
* @param lease the amount of time entry is placed in space
* @param this.transaction must be a valid transaction handle
* @return <tt>true</tt> if entry is sucessfully written to space. */

```

```

public boolean transUpdate(boolean value, long lease)
{
    return update3(value, lease, this.getTransactionHandler());
}

/** Reads an entry value with transaction
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public boolean transRead(long timeOut)
{
    return readIfExists2(timeOut, this.getTransactionHandler());
}

/** Takes an entry value with transaction; entry will be removed from space
 * @param timeOut the maximum waiting time when reading the entry
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public boolean transTake( long timeOut)
{
    return takeIfExists2(timeOut, this.getTransactionHandler());
}
}

```

7. TSDouble.java

```
package tuplespace.core;
import tuplespace.entries.*;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSDouble</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryDouble entry from space. Every
 * EntryDouble entry in the space is identified by a unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 * @see EntryDouble
 */
public class TSDouble extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryDouble entry, entryTemplate;

    /** Constructor for TSDouble
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is raised.
     * @param entryID the unique ID that identifies the Double Entry
     */
    public TSDouble(JavaSpace space, SpaceActionHandler spaceAction, String entryID) {
        super();
        initTSDouble(space, spaceAction, entryID);
    }

    /** initialize TSDouble
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is raised.
     * @param entryID the unique ID that identifies the Double Entry
     * @return <tt>true</tt> if internal states are successfully reinitialized.
     */
    public boolean initTSDouble(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        try{
            initTSBase(space, spaceAction, entryID);
            entry = new EntryDouble(entryID);
            entryTemplate = new EntryDouble(entryID);
            eventRegistration = null;
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Remove all existing entries from space */
    public void cleanSpace()
    {
        cleanSpace1(null);
    }

    /** Remove all existing entries from space
     * @param txn a valid transaction handle */
    public void cleanSpace1(Transaction txn)
    {

```

```

Object test;
do{
    try{
        test = space.takeIfExists(entryTemplate,txn,1000);
        //print("remove [" + entryID + "] from space");
    }catch (Exception e){
        e.printStackTrace();
        test = null;
    }
}while(test != null);
}

//*****
//*** Write Entry
***
//*****

/** Writes value to space
 * @param value the entry value (Double)
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write1(double value)
{
    return write3( value, this.writeLeaseTime, null);
}

/** Writes value to space
 * @param value the double value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write2(double value, long lease)
{
    return write3( value, lease, null);
}

/** Writes value to space
 * @param value the double value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write3(double value, long lease, Transaction txn)
{
    try{
        entry = new EntryDouble(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Update Entry
***
//*****

/** Updates an existing entry value
 * @param value the entry value (Double) to be updated
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update1(double value)
{
    return update3( value, this.updateLeaseTime, null);
}

/** Updates an existing entry value
 * @param value the double value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully updated. */

```

```

public boolean update2(double value, long lease)
{
    return update3( value, lease, null);
}

/** Updates an existing entry value
 * @param value the double value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update3(double value, long lease, Transaction txn)
{
    try{
        cleanSpace1(txn);
        entry = new EntryDouble(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
/** Read Entry
    ***
//*****

/** Reads an entry value
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public double readIfExists()
{
    return readIfExists2(this.readTimeOut, null);
}

/** Reads an entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public double readIfExists1(long timeOut)
{
    return readIfExists2(timeOut, null);
}

/** Reads an entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public double readIfExists2(long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryDouble) space.readIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryDouble.doubleValue();
        }
        else
            return -1;
    }
    catch (Exception e){
        e.printStackTrace();
        return -1;
    }
}

```

```

    }

    /*******
    /**** Take Entry
    /*******

    /** Takes an entry value; entry will be removed from space
    * @return <tt>-1</tt> if entry is not available, otherwise
    *         the entry value. */
    public double takeIfExists()
    {
        return takeIfExists1(this.takeTimeOut);
    }

    /** Takes an entry value; entry will be removed from space
    * @return <tt>-1</tt> if entry is not available, otherwise
    *         the entry value. */
    public double takeIfExists1( long timeOut)
    {
        return takeIfExists2(timeOut, null);
    }

    /** Takes an entry value; entry will be removed from space
    * @param timeOut the maximum waiting time when reading the entry
    * @param txn a valid transaction handle, if transaction is involved
    * @return <tt>-1</tt> if entry is not available, otherwise
    *         the entry value. */
    public double takeIfExists2( long timeOut, Transaction txn)
    {
        this.setResult(false);
        try{
            entry = (EntryDouble) space.takeIfExists(entryTemplate, txn, timeOut);
            if( entry != null)
            {
                this.setResult(true);
                return entry.entryDouble.doubleValue();
            }
            else
                return -1;
        }
        catch (Exception e){
            e.printStackTrace();
            return -1;
        }
    }

    /*******
    /**** Notify
    /*******

    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent()
    {
        return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
    }

    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
    *         method; called when remote event is raised.
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent1(SpaceActionHandler spaceAction)
    {
        return startEvent3(spaceAction, this.notifyLeaseTime, null );
    }

```

```

}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
{
    return startEvent3(spaceAction, lease , null);
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
{
    try{
        // Remove earlier event notification
        if(eventRegistration != null)
            eventRegistration.getLease().cancel();

        // Register new event notification
        SpaceEventListener listener = new SpaceEventListener(TS_DOUBLE, entryID,
space,spaceAction);
        eventRegistration = space.notify(entryTemplate, txn, listener, lease, null);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Stop notification
 * @return <tt>true</tt> if notification is disabled. */
public boolean stopEvent(){
    try{
        if( eventRegistration != null)
        {
            eventRegistration.getLease().cancel();
            eventRegistration = null;
        }
        return true;
    }
    catch (Exception e){
        return false;
    }
}

//*****
//*** Transaction Methods
//***
//*****

/** Writes entry value with transaction
 * @param value the boolean value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>true</tt> if entry is successfully written to space. */
public boolean transWrite(double value, long lease)
{
    return write3(value,lease,this.getTransactionHandler());
}

```

```

    }
    /** Updates an existing entry value with transaction
    * @param value the boolean value that is to be written to space
    * @param lease the amount of time entry is placed in space
    * @param this.transaction must be a valid transaction handle
    * @return <tt>true</tt> if entry is sucessfully written to space. */
    public boolean transUpdate(double value, long lease)
    {
        return update3(value, lease, this.getTransactionHandler());
    }
    /** Reads an entry value with transaction
    * @param timeOut the maximum waiting time when reading the entry from space
    * @param this.transaction must be a valid transaction handle
    * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
    *         the entry value. */
    public double transRead(long timeOut)
    {
        return readIfExists2(timeOut, this.getTransactionHandler());
    }

    /** Takes an entry value with transaction; entry will be removed from space
    * @param timeOut the maximum waiting time when reading the entry
    * @param this.transaction must be a valid transaction handle
    * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
    *         the entry value. */
    public double transTake( long timeOut)
    {
        return takeIfExists2(timeOut, this.getTransactionHandler());
    }
}

```

8. TSLong.java

```
package tuplespace.core;

import tuplespace.entries.*;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSLong</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryLong entry from space. Every
 * EntryLong entry in the space is identified by an unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 September 2000
 * @see EntryLong
 */

public class TSLong extends TSBBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryLong entry, entryTemplate;

    /** Constructor for TSLong
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     *        method, called when remote event is rised.
     * @param entryID the unique ID that identifies the Long Entry
     */
    public TSLong(JavaSpace space, SpaceActionHandler spaceAction, String entryID) {
        super();
        initTSLong(space, spaceAction, entryID);
    }

    /** initialize TSLong
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     *        method; called when remote event is raised.
     * @param entryID the unique ID that identifies the Long Entry
     * @return <tt>true</tt> if internal states are sucessfully reinitialized.
     */
    public boolean initTSLong(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        try{
            initTSBase(space, spaceAction, entryID);
            entry = new EntryLong(entryID);
            entryTemplate = new EntryLong(entryID);
            eventRegistration = null;
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Remove all existing entries from space */
    public void cleanSpace()
    {
        cleanSpace1(null);
    }

    /** Remove all existing entries from space
     * @param txn a valid transaction handle */
}
```

```

public void cleanSpace1(Transaction txn)
{
    Object test;
    do{
        try{
            test = space.takeIfExists(entryTemplate,txn,1000);
            //print("remove [" + entryID + "] from space");
        }catch (Exception e){
            e.printStackTrace();
            test = null;
        }
    }while(test != null);
}

//*****
//*** Write Entry ***
//*****

/** Writes value to space
 * @param value the entry value (Long)
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write1(long value)
{
    return write3( value, this.writeLeaseTime, null);
}

/** Writes value to space
 * @param value the long value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write2(long value, long lease)
{
    return write3( value, lease, null);
}

/** Writes value to space
 * @param value the long value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write3(long value, long lease, Transaction txn)
{
    try{
        entry = new EntryLong(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Update Entry ***
//*****

/** Updates an existing entry value
 * @param value the entry value (Long) to be updated
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update1(long value)
{
    return update3( value, this.updateLeaseTime, null);
}

/** Updates an existing entry value
 * @param value the long value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully updated. */

```

```

public boolean update2(long value, long lease)
{
    return update3( value, lease, null);
}

/** Updates an existing entry value
 * @param value the long value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update3(long value, long lease, Transaction txn)
{
    try{
        cleanSpace1(txn);
        entry = new EntryLong(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
/** Read Entry ***
//*****

/** Reads an entry value
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public long readIfExists()
{
    return readIfExists2(this.readTimeOut, null);
}

/** Reads an entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public long readIfExists1(long timeOut)
{
    return readIfExists2(timeOut, null);
}

/** Reads an entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public long readIfExists2(long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryLong) space.readIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryLong.longValue();
        }
        else
            return -1;
    }
    catch (Exception e){
        e.printStackTrace();
        return -1;
    }
}

```

```

//*****
//*** Take Entry ***
//*****

/** Takes an entry value; entry will be removed from space
 * @return <tt>-1</tt> if entry is not available, otherwise
 *         the entry value. */
public long takeIfExists()
{
    return takeIfExists1(this.takeTimeOut);
}

/** Takes an entry value; entry will be removed from space
 * @return <tt>-1</tt> if entry is not available, otherwise
 *         the entry value. */
public long takeIfExists1( long timeOut)
{
    return takeIfExists2(timeOut, null);
}

/** Takes an entry value; entry will be removed from space
 * @param timeOut the maximum waiting time when reading the entry
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>-1</tt> if entry is not available, otherwise
 *         the entry value. */
public long takeIfExists2( long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryLong) space.takeIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryLong.longValue();
        }
        else
            return -1;
    }
    catch (Exception e){
        e.printStackTrace();
        return -1;
    }
}

//*****
//*** Notify ***
//*****

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent()
{
    return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 *         method; called when remote event is raised.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent1(SpaceActionHandler spaceAction)
{
    return startEvent3(spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry

```

```

* that matches the entry ID is added into the space.
* @param spaceAction the class that implements the actionPerformed(ActionEvent e)
*      method; called when remote event is raised.
* @param lease the amount of time for notification to remain active
* @return <tt>true</tt> if notification is enabled. */
public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
{
    return startEvent3(spaceAction, lease , null);
}

/** Start notification; remote event will be raised if any entry
* that matches the entry ID is added into the space.
* @param spaceAction the class that implements the actionPerformed(ActionEvent e)
*      method; called when remote event is raised.
* @param lease the amount of time for notification to remain active
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>true</tt> if notification is enabled. */
public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
{
    try{
        // Remove earlier event notification
        if(eventRegistration != null)
            eventRegistration.getLease().cancel();

        // Register new event notification
        SpaceEventListener listener = new SpaceEventListener(TS_LONG, entryID,
space, spaceAction);
        eventRegistration = space.notify(entryTemplate, txn, listener, lease, null);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Stop notification
* @return <tt>true</tt> if notification is disabled. */
public boolean stopEvent(){
    try{
        if( eventRegistration != null)
        {
            eventRegistration.getLease().cancel();
            eventRegistration = null;
        }
        return true;
    }
    catch (Exception e){
        return false;
    }
}

//*****
/** Transaction Methods ***
//*****

/** Writes entry value with transaction
* @param value the long value that is to be written to space
* @param lease the amount of time entry is placed in space
* @param this.transaction must be a valid transaction handle
* @return <tt>true</tt> if entry is successfully written to space. */
public boolean transWrite(long value, long lease)
{
    return write3(value, lease, this.getTransactionHandler());
}

/** Updates an existing entry value with transaction
* @param value the long value that is to be written to space
* @param lease the amount of time entry is placed in space

```

```

* @param this.transaction must be a valid transaction handle
* @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transUpdate(long value, long lease)
{
    return update3(value, lease, this.getTransactionHandler());
}

/** Reads an entry value with transaction
* @param timeOut the maximum waiting time when reading the entry from space
* @param this.transaction must be a valid transaction handle
* @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
*         the entry value. */
public long transRead(long timeOut)
{
    return readIfExists2(timeOut, this.getTransactionHandler());
}

/** Takes an entry value with transaction; entry will be removed from space
* @param timeOut the maximum waiting time when reading the entry
* @param this.transaction must be a valid transaction handle
* @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
*         the entry value. */
public long transTake( long timeOut)
{
    return takeIfExists2(timeOut, this.getTransactionHandler());
}
}

```

9. TSHash.java

```
package tuplespace.core;

import tuplespace.entries.*;
import java.util.HashMap;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSHash</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryHash entry from space. Every
 * EntryHash entry in the space is identified by an unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 * @see EntryHash
 */

public class TSHash extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryHash entry;
    private EntryTemplate entryTemplate;
    private HashMap container;
    /** Constructor for TSHash
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is rised.
     * @param entryID the unique ID that identifies the Hash Entry
     */
    public TSHash(JavaSpace space, SpaceActionHandler spaceAction, String entryID) {
        super();
        initTSHash(space, spaceAction, entryID);
    }

    /** initialize TSHash
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is raised.
     * @param entryID the unique ID that identifies the Hash Entry
     * @return <tt>true</tt> if internal states are sucessfully reinitialized.
     */
    public boolean initTSHash(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        try{
            initTSBase(space, spaceAction, entryID);
            container = new HashMap();
            entry = new EntryHash(entryID);
            entryTemplate = new EntryHash(entryID);
            eventRegistration = null;
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Remove all existing entries from space */
    public void cleanSpace()
    {
        cleanSpace1(null);
    }
    /** Remove all existing entries from space
```

```

* @param txn a valid transaction handle */
public void cleanSpace1(Transaction txn)
{
    Object test;
    do{
        try{
            test = space.takeIfExists(entryTemplate,txn,1000);
            //print("remove [" + entryID + "] from space");
        }catch (Exception e){
            e.printStackTrace();
            test = null;
        }
    }while(test != null);
}

//*****
//*** HashMap Operation ***
//*****
public boolean clearContainer(){
    container = new HashMap();
    return true;
}

public HashMap getContainer(){
    return container ;
}

public void getContainer(HashMap map){
    container = map;
}

public boolean setBoolean(String id, boolean value){
    try{
        container.put(id,new Boolean(value));
        return true;
    }
    catch(Exception e){
        return false;
    }
}

public boolean setInteger(String id, int value){
    try{
        container.put(id,new Integer(value));
        return true;
    }
    catch(Exception e){
        return false;
    }
}

public boolean setFloat(String id, float value){
    try{
        container.put(id,new Float(value));
        return true;
    }
    catch(Exception e){
        return false;
    }
}

public boolean setLong(String id, long value){
    try{
        container.put(id,new Long(value));
        return true;
    }
    catch(Exception e){
        return false;
    }
}

```

```

public boolean setDouble(String id, double value){
    try{
        container.put(id,new Double(value));
        return true;
    }
    catch(Exception e){
        return false;
    }
}

public boolean setString(String id, String content){
    try{
        container.put(id,content);
        return true;
    }
    catch(Exception e){
        return false;
    }
}

public boolean remove(String id){
    if(container.containsKey(id)){
        container.remove(id);
        return true;
    }
    else{
        return false;
    }
}

public int getInteger(String id){
    Integer item;
    try{
        item = (Integer) container.get(id);
        return item.intValue();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return -1;
    }
}

public boolean getBoolean(String id){
    Boolean item;
    try{
        item = (Boolean) container.get(id);
        return item.booleanValue();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return false;
    }
}

public float getFloat(String id){
    Float item;
    try{
        item = (Float) container.get(id);
        return item.floatValue();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return -1;
    }
}

public long getLong(String id){
    Long item;
    try{
        item = (Long) container.get(id);

```

```

        return item.longValue();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return -1;
    }
}

public double getDouble(String id){
    Double item;
    try{
        item = (Double) container.get(id);
        return item.doubleValue();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return -1;
    }
}

public String getString(String id){
    String item;
    try{
        item = (String) container.get(id);
        return item;
    } catch (Exception e) {
        System.err.println(e.getMessage());
        return "Error finding ID["+id+"] in Hash Map";
    }
}

/***** Write Entry *****/
/** Writes value to space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write()
{
    return write2( this.writeLeaseTime, null);
}

/** Writes value to space
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write1( long lease)
{
    return write2( lease, null);
}

/** Writes value to space
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write2( long lease, Transaction txn)
{
    try{
        entry = new EntryHash(this.entryID, container);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/*****

```

```

//*** Update Entry
    ***
//*****

/** Updates an existing entry value
 * @param value the entry value (Hash) to be updated
 * @return <tt>true</tt> if entry is successfully updated. */
public boolean update()
{
    return update2( this.updateLeaseTime, null);
}

/** Updates an existing entry value
 * @param value the double value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is successfully updated. */
public boolean update1( long lease)
{
    return update2( lease, null);
}

/** Updates an existing entry value
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is successfully updated. */
public boolean update2( long lease, Transaction txn)
{
    try{
        cleanSpace1(txn);
        entry = new EntryHash(this.entryID, container);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
//*** Read Entry
    ***
//*****

/** Read entry value
 * @return <tt>-1</tt> if entry is not available, otherwise
 * the entry value. */
public boolean readIfExists()
{
    return readIfExists2(this.readTimeOut, null);
}

/** Read entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @return <tt>false</tt> if entry is not available, otherwise
 * the entry value. */
public boolean readIfExists1(long timeOut)
{
    return readIfExists2(timeOut, null);
}

/** Read entry value
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>false</tt> if entry is not available, otherwise
 * the entry value. */
public boolean readIfExists2(long timeOut, Transaction txn)
{

```

```

        this.setResult(false);
        try{
            entry = (EntryHash) space.readIfExists(entryTemplate, txn, timeOut);
            if( entry != null)
            {
                this.setResult(true);
                container = entry.getHashMap();
                return true;
            }
            else
                return false;
        }
        catch (Exception e){
            e.printStackTrace();
            return false;
        }
    }

    /**
     * Take entry value; entry will be removed from space
     * @return <tt>>false</tt> if entry is not available, otherwise
     * the entry value. */
    public boolean takeIfExists()
    {
        return takeIfExists1(this.takeTimeOut);
    }

    /**
     * Take entry value; entry will be removed from space
     * @return <tt>>false</tt> if entry is not available, otherwise
     * the entry value. */
    public boolean takeIfExists1( long timeOut)
    {
        return takeIfExists2(timeOut, null);
    }

    /**
     * Take entry value; entry will be removed from space
     * @param timeOut the maximum waiting time when reading the entry
     * @param txn a valid transaction handle, if transaction is involved
     * @return <tt>>false</tt> if entry is not available, otherwise
     * the entry value. */
    public boolean takeIfExists2( long timeOut, Transaction txn)
    {
        this.setResult(false);
        try{
            entry = (EntryHash) space.takeIfExists(entryTemplate, txn, timeOut);
            if( entry != null)
            {
                this.setResult(true);
                container = entry.getHashMap();
                return true;
            }
            else
                return false;
        }
        catch (Exception e){
            e.printStackTrace();
            return false;
        }
    }

    /**
     * Notify
     */

```

```

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent()
{
    return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent1(SpaceActionHandler spaceAction)
{
    return startEvent3(spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
{
    return startEvent3(spaceAction, lease , null);
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
{
    try{
        // Remove earlier event notification
        if(eventRegistration != null)
            eventRegistration.getLease().cancel();

        // Register new event notification
        SpaceEventListener listener = new SpaceEventListener(TS_HASH, entryID,
space,spaceAction);
        eventRegistration = space.notify(entryTemplate, txn, listener, lease, null);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Stop notification
 * @return <tt>true</tt> if notification is disabled. */
public boolean stopEvent(){
    try{
        if( eventRegistration != null)
        {
            eventRegistration.getLease().cancel();
            eventRegistration = null;
        }
        return true;
    }
    catch (Exception e){

```

```

        return false;
    }
}

//*****
//*** Transaction Methods
//*****

/** Writes entry value with transaction
 * @param value the float value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transWrite( long lease)
{
    return write2(lease,this.getTransactionHandler());
}

/** Updates an existing entry value with transaction
 * @param value the float value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transUpdate( long lease)
{
    return update2(lease,this.getTransactionHandler());
}

/** Reads an entry value with transaction
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public boolean transRead(long timeOut)
{
    this.setResult(false);
    try{
        entry = (EntryHash) space.readIfExists(entryTemplate,
this.getTransactionHandler(), timeOut);
        if( entry != null)
        {
            this.setResult(true);
            container = entry.getHashMap();
            return true;
        }
        else
            return false;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Takes an entry value with transaction; entry will be removed from space
 * @param timeOut the maximum waiting time when reading the entry
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public boolean transTake( long timeOut)
{
    this.setResult(false);
    try{
        entry = (EntryHash) space.takeIfExists(entryTemplate,
this.getTransactionHandler(), timeOut);
        if( entry != null)
        {
            this.setResult(true);
            container = entry.getHashMap();

```

```
        return true;
    }
    else
        return false;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}
```

10. TSQueue.java

```
package tuplespace.core;

import tuplespace.entries.*;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSQueue</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryQueue entry from space. Every
 * EntryQueue entry in the space is identified by an unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 * @see EntryQueue
 */

public class TSQueue extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryQueueItem item, itemTemplate;
    private EntryQueueStatus status, statusTemplate;
    private long bufferSize;

    /** Constructor for TSQueue
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is rised.
     * @param entryID the unique ID that identifies the Queue Entry
     */
    public TSQueue(JavaSpace space, SpaceActionHandler spaceAction, String entryID, long
size) {
        super();
        initTSQueue(space, spaceAction, entryID, size);
    }

    /** initialize TSQueue
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method; called when remote event is raised.
     * @param entryID the unique ID that identifies the Queue Entry
     * @return <tt>true</tt> if internal states are sucessfully reinitialized.
     */
    public boolean initTSQueue(JavaSpace space, SpaceActionHandler spaceAction, String
entryID, long size)
    {
        this.bufferSize = size;
        try{
            initTSBase(space, spaceAction, entryID);
            item = new EntryQueueItem(entryID);
            itemTemplate = new EntryQueueItem(entryID);

            status = new EntryQueueStatus(entryID, size);
            statusTemplate = new EntryQueueStatus(entryID);
            initQueueIndexes();
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Initialize */
    private boolean initQueueIndexes()
```

```

{
    try{
        status = (EntryQueueStatus) space.read(statusTemplate,null,2000);
        if(status == null ){
            cleanSpace();
            status = new EntryQueueStatus(entryID,bufferSize);
            space.write(status,null, Lease.FOREVER);
            print("Initialized! TSQueue");
            return true;
        }
        else
            return false;
    }catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Remove all existing entries from space */
public void cleanSpace()
{
    cleanSpace1(null);
}

/** Remove all existing entries from space
 * @param txn a valid transaction handle */
public void cleanSpace1(Transaction txn)
{
    Object test;
    do{
        try{
            test = space.takeIfExists(statusTemplate,txn,2000);
            print("removing stack Start Index from space");
        }catch (Exception e){
            e.printStackTrace();
            test = null;
        }
    }while(test != null);

    do{
        try{
            test = space.takeIfExists(itemTemplate,txn,2000);
            print("removing stack item from space");
        }catch (Exception e){
            e.printStackTrace();
            test = null;
        }
    }while(test != null);
}

/** Remove all existing entries from space */
public void cleanItem(long index)
{
    cleanItem1(null, index);
}

/** Remove all existing entries from space
 * @param txn a valid transaction handle */
public void cleanItem1(Transaction txn, long index)
{
    Object test;
    do{
        try{
            EntryQueueItem itemTmp = new EntryQueueItem(entryID);
            itemTmp.position = new Long(index);
            test = space.takeIfExists(itemTmp,txn,1000);
            print("removing stack item from space");
        }catch (Exception e){

```

```

        e.printStackTrace();
        test = null;
    }
    }while(test != null);
}

/** Remove all existing entries from space */
public void cleanStatus()
{
    cleanStatus1(null);
}

/** Remove all existing entries from space
 * @param txn a valid transaction handle */
public void cleanStatus1(Transaction txn)
{
    Object test;
    do{
        try{
            EntryQueueStatus itemTmp = new EntryQueueStatus(entryID);
            test = space.takeIfExists(itemTmp,txn,1000);
            print("removing stack item from space");
        }catch (Exception e){
            e.printStackTrace();
            test = null;
        }
    }while(test != null);
}

public String read1(long index){
    try{
        itemTemplate.position = new Long(index);
        item = (EntryQueueItem) space.readIfExists(itemTemplate,null,2000);
        if(item != null){
            print("Queue item[" + index + "] -> " + item.content);
            return item.content;
        }
        else
            return null;
    } catch(Exception e){
        e.printStackTrace();
        return null;
    }
}

/** Updates an existing entry value
 * @param value the String value that is to be updated
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully updated. */
private boolean update3(String msg, long lease, Transaction txn, long index)
{
    try{
        cleanItem1(txn, index);
        item = new EntryQueueItem(entryID,index,msg);
        space.write(item, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

public String take(){
    return takel(null);
}

public String takel( Transaction txn){
    long startIndex;

```

```

try{
    status = (EntryQueueStatus) space.readIfExists (statusTemplate,txn,2000);
    if(status != null){
        if(!status.isEmpty()){
            startIndex = status.startIndex.longValue();
            itemTemplate.position = new Long(startIndex);
            item = (EntryQueueItem) space.takeIfExists(itemTemplate,txn,2000);
            status.incrementStartIndex ();
            cleanStatus1(txn);
            space.write(status,txn, Lease.FOREVER);
            print("Queue item[" + startIndex + "] -> " + item.content);
            return item.content;
        }
        return null;
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}

public boolean write1(String msg){
    return write2(msg,null);
}

public boolean write2(String msg, Transaction txn){
    long nextIndex;
    try{
        status = (EntryQueueStatus) space.readIfExists(statusTemplate,txn,2000);
        if(status != null ){
            if( !status.isFull()){
                status.incrementEndIndex();
                nextIndex = status.endIndex.longValue();
                print("Written! " + msg + " to space" + " Index->" + nextIndex);
                update3(msg, Lease.FOREVER, txn,nextIndex);
                cleanStatus1(txn);
                space.write(status,txn, Lease.FOREVER);
                return true;
            }
        }
        return false;
    } catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

public void printQueue(){
    try{
        status = (EntryQueueStatus) space.read(statusTemplate,null,2000);
        print("Start Index : " + status.getStartIndex() + "      End Index : " +
status.getEndIndex());
        for(long i = status.getStartIndex(); i <= status.getEndIndex();i++)
        {
            itemTemplate.position = new Long(i);
            item = (EntryQueueItem) space.readIfExists(itemTemplate,null,2000);
            if(item != null)
                print("Queue item[" + i + "] -> " + item.content);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}

//*****

```

```

    /*** Notify
        ***
    /*******
    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent()
    {
        return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
    }

    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
    * method; called when remote event is raised.
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent1(SpaceActionHandler spaceAction)
    {
        return startEvent3(spaceAction, this.notifyLeaseTime, null );
    }

    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
    * method; called when remote event is raised.
    * @param lease the amount of time for notification to remain active
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
    {
        return startEvent3(spaceAction, lease , null);
    }

    /** Start notification; remote event will be raised if any entry
    * that matches the entry ID is added into the space.
    * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
    * method; called when remote event is raised.
    * @param lease the amount of time for notification to remain active
    * @param txn a valid transaction handle, if transaction is involved
    * @return <tt>true</tt> if notification is enabled. */
    public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
    {
        try{
            // Remove earlier event notification
            if(eventRegistration != null)
                eventRegistration.getLease().cancel();

            // Register new event notification
            itemTemplate = new EntryQueueItem(entryID);
            SpaceEventListener listener = new SpaceEventListener(TS_QUEUE, entryID,
space,spaceAction);
            eventRegistration = space.notify(itemTemplate, txn, listener, lease, null);
            return true;
        }
        catch (Exception e){
            e.printStackTrace();
            return false;
        }
    }

    /** Stop notification
    * @return <tt>true</tt> if notification is disabled. */
    public boolean stopEvent(){
        try{
            if( eventRegistration != null)
            {
                eventRegistration.getLease().cancel();
                eventRegistration = null;
            }
        }
    }

```

```

        }
        return true;
    }
    catch (Exception e){
        return false;
    }
}

public long getStartIndex(){
    try{
        status = (EntryQueueStatus) space.readIfExists(statusTemplate,null,2000);
        return status.getStartIndex();
    } catch(Exception e){
        return -1;
    }
}

public long getEndIndex(){
    try{
        status = (EntryQueueStatus) space.readIfExists(statusTemplate,null,2000);
        return status.getEndIndex();
    } catch(Exception e){
        return -1;
    }
}
}

```

11. TSSString.java

```
package tuplespace.core;

import tuplespace.entries.*;
import net.jini.space.JavaSpace;
import net.jini.core.entry.Entry;
import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import net.jini.core.event.EventRegistration;

/**
 * The <code>TSSString</code> class implements the methods for reading, writing,
 * updating, notifying and retrieving EntryString entry from space. Every
 * EntryString entry in the space is identified by a unique ID (entryID).
 *
 * A subclass that implements the SpaceActionHandler interface has to loaded
 * during initialization if remote event notification is used.
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 October 2000
 * @see EntryString
 */

public class TSSString extends TSBase implements java.io.Serializable,
    SpaceEventRegistration, TSConstants{
    private EntryString entry, entryTemplate;

    /** Constructor for TSSString
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method, called when remote event is raised.
     * @param entryID the unique ID that identifies the String Entry
     */
    public TSSString(JavaSpace space, SpaceActionHandler spaceAction, String entryID) {
        super();
        initTSSString(space, spaceAction, entryID);
    }

    /** initialize TSSString
     * @param space JavaSpace handle
     * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
     * method; called when remote event is raised.
     * @param entryID the unique ID that identifies the String Entry
     * @return <tt>true</tt> if internal states are successfully reinitialized.
     */
    public boolean initTSSString(JavaSpace space, SpaceActionHandler spaceAction, String
entryID)
    {
        try{
            initTSBase(space, spaceAction, entryID);
            entry = new EntryString(entryID);
            entryTemplate = new EntryString(entryID);
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    /** Remove all existing entries from space */
    public void cleanSpace()
    {
        cleanSpace1(null);
    }

    /** Remove all existing entries from space
     * @param txn a valid transaction handle */
    public void cleanSpace1(Transaction txn)
    {

```

```

Object test;
do{
    try{
        test = space.takeIfExists(entryTemplate,txn,1000);
        //print("remove [" + entryID + "] from space");
    }catch (Exception e){
        e.printStackTrace();
        test = null;
    }
}while(test != null);
}

//*****
/** Write Entry
    ***
//*****

/** Writes value to space
 * @param value the entry value (String)
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write1(String value)
{
    return write3( value, this.writeLeaseTime, null);
}

/** Writes value to space
 * @param value the String value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write2(String value, long lease)
{
    return write3( value, lease, null);
}

/** Writes value to space
 * @param value the String value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean write3(String value, long lease, Transaction txn)
{
    try{
        entry = new EntryString(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
/** Update Entry
    ***
//*****

/** Updates an existing entry value
 * @param value the entry value (String) to be updated
 * @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update1(String value)
{
    return update3( value, this.updateLeaseTime, null);
}

/** Updates an existing entry value
 * @param value the String value that is to be updated
 * @param lease the amount of time entry is placed in space

```

```

* @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update2(String value, long lease)
{
    return update3( value, lease, null);
}

/** Updates an existing entry value
* @param value the String value that is to be updated
* @param lease the amount of time entry is placed in space
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>true</tt> if entry is sucessfully updated. */
public boolean update3(String value, long lease, Transaction txn)
{
    try{
        cleanSpace1(txn);
        entry = new EntryString(this.entryID, value);
        space.write(entry, txn, lease);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

//*****
/** Read Entry
***
//*****

/** Reads an entry value
* @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
* the entry value. */
public String readIfExists()
{
    return readIfExists2(this.readTimeOut, null);
}

/** Reads an entry value
* @param timeOut the maximum waiting time when reading the entry from space
* @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
* the entry value. */
public String readIfExists1(long timeOut)
{
    return readIfExists2(timeOut, null);
}

/** Reads an entry value
* @param timeOut the maximum waiting time when reading the entry from space
* @param txn a valid transaction handle, if transaction is involved
* @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
* the entry value. */
public String readIfExists2(long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryString) space.readIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryString;
        }
    }
    else
        return "No Entry Found!";
    catch (Exception e){
        e.printStackTrace();
        return "Exception in method !";
    }
}

```

```

    }
}

//*****
//*** Take Entry
//*****

/** Takes an entry value; entry will be removed from space
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 * the entry value. */
public String takeIfExists()
{
    return takeIfExists1(this.takeTimeOut);
}

/** Takes an entry value; entry will be removed from space
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 * the entry value. */
public String takeIfExists1( long timeOut)
{
    return takeIfExists2(timeOut, null);
}

/** Takes an entry value; entry will be removed from space
 * @param timeOut the maximum waiting time when reading the entry
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 * the entry value. */
public String takeIfExists2( long timeOut, Transaction txn)
{
    this.setResult(false);
    try{
        entry = (EntryString) space.takeIfExists(entryTemplate, txn, timeOut);
        if( entry != null)
        {
            this.setResult(true);
            return entry.entryString;
        }
        else
            return "No Entry Found!";
    }
    catch (Exception e){
        e.printStackTrace();
        return "Exception in method !";
    }
}

//*****
//*** Notify
//*****

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent()
{
    return startEvent3(this.spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.

```

```

* @return <tt>true</tt> if notification is enabled. */
public boolean startEvent1(SpaceActionHandler spaceAction)
{
    return startEvent3(spaceAction, this.notifyLeaseTime, null );
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent2(SpaceActionHandler spaceAction, long lease )
{
    return startEvent3(spaceAction, lease , null);
}

/** Start notification; remote event will be raised if any entry
 * that matches the entry ID is added into the space.
 * @param spaceAction the class that implements the actionPerformed(ActionEvent e)
 * method; called when remote event is raised.
 * @param lease the amount of time for notification to remain active
 * @param txn a valid transaction handle, if transaction is involved
 * @return <tt>true</tt> if notification is enabled. */
public boolean startEvent3(SpaceActionHandler spaceAction, long lease , Transaction
txn)
{
    try{
        // Remove earlier event notification
        if(eventRegistration != null){
            try {
                Lease l = eventRegistration.getLease();
                l.cancel();
            }catch (Exception e){}
        }
        // Register new event notification
        SpaceEventListener listener = new SpaceEventListener(TS_STRING, entryID,
space,spaceAction);
        eventRegistration = space.notify(entryTemplate, txn, listener, lease, null);
        return true;
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

/** Stop notification
 * @return <tt>true</tt> if notification is disabled. */
public boolean stopEvent(){
    try{
        if( eventRegistration != null)
        {
            try {
                Lease l = eventRegistration.getLease();
                l.cancel();
            }catch (Exception e){}
            eventRegistration = null;
        }
        return true;
    }
    catch (Exception e){
        return false;
    }
}

//*****
//*** Transaction Methods
***

```

```

//*****

/** Writes entry value with transaction
 * @param value the String value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transWrite(String value, long lease)
{
    return write3(value, lease, this.getTransactionHandler());
}

/** Updates an existing entry value with transaction
 * @param value the String value that is to be written to space
 * @param lease the amount of time entry is placed in space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>true</tt> if entry is sucessfully written to space. */
public boolean transUpdate(String value, long lease)
{
    return update3(value, lease, this.getTransactionHandler());
}

/** Reads an entry value with transaction
 * @param timeOut the maximum waiting time when reading the entry from space
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public String transRead(long timeOut)
{
    return readIfExists2(timeOut, this.getTransactionHandler());
}

/** Takes an entry value with transaction; entry will be removed from space
 * @param timeOut the maximum waiting time when reading the entry
 * @param this.transaction must be a valid transaction handle
 * @return <tt>"Exception in method !"</tt> if entry is not available, otherwise
 *         the entry value. */
public String transTake( long timeOut)
{
    return takeIfExists2(timeOut, this.getTransactionHandler());
}
}

```

C. ENTRIES PACKAGE

1. EntryBoolean.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryBoolean</code> is the entry template for type - Boolean
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryBoolean implements Entry {
    public String entryID;
    public Boolean entryBoolean;

    public EntryBoolean(){
    }

    public EntryBoolean(String entryID){
        this.entryID = entryID;
    }

    public EntryBoolean(String entryID, boolean entryValue){
        this.entryID = entryID;
        this.entryBoolean = new Boolean(entryValue);
    }
}
```

2. EntryBytes.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryBytes</code> is the entry template for type - Bytes
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryBytes implements Entry {
    public String entryID;
    public Byte[] entryBytes;

    public EntryBytes(){
    }

    public EntryBytes(String entryID){
        this.entryID = entryID;
    }

    public EntryBytes(String entryID, Byte[] entryValue){
        this.entryID = entryID;
        this.entryBytes = entryValue;
    }
}
```

3. EntryClass.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryClass</code> is the entry template for type - Class
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryClass implements Entry {
    public String entryID;
    public String codeURL;
    public String className;

    public EntryClass(){
    }

    public EntryClass(String entryID){
        this.entryID = entryID;
    }

    public EntryClass(String entryID, String codeURL, String className){
        this.entryID = entryID;
        this.codeURL = codeURL;
        this.className = className;
    }
}
```

4. EntryDouble.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryDouble</code> is the entry template for type - Double
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryDouble implements Entry {
    public String entryID;
    public Double entryDouble;

    public EntryDouble(){
    }

    public EntryDouble(String entryID){
        this.entryID = entryID;
    }

    public EntryDouble(String entryID, double entryValue){
        this.entryID = entryID;
        this.entryDouble = new Double(entryValue);
    }
}
```

5. EntryFloat.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryFloat</code> is the entry template for type - Float
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
```

```

*/

public class EntryFloat implements Entry {
    public String entryID;
    public Float entryFloat;

    public EntryFloat(){
    }

    public EntryFloat(String entryID){
        this.entryID = entryID;
    }

    public EntryFloat(String entryID, float entryValue){
        this.entryID = entryID;
        this.entryFloat = new Float(entryValue);
    }
}

```

6. EntryHash.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;
import java.util.HashMap;

/**
 * The <code>EntryHash</code> is the entry template for type - User Defined
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */

public class EntryHash implements Entry {
    public HashMap map;
    public String entryID;
    public Integer entryInteger;

    public EntryHash(){
    }

    public EntryHash(String entryID){
        this.entryID = entryID;
    }

    public EntryHash(String entryID, HashMap map){
        this.entryID = entryID;
        this.map = map;
    }

    public HashMap getHashMap(){
        return map;
    }

    public void setHashMap(HashMap map){
        this.map = map;
    }
}

```

7. EntryInteger.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryInteger</code> is the entry template for type - Integer
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000

```

```

*/
public class EntryInteger implements Entry {
    public String entryID;
    public Integer entryInteger;

    public EntryInteger(){
    }

    public EntryInteger(String entryID){
        this.entryID = entryID;
    }

    public EntryInteger(String entryID, int entryValue){
        this.entryID = entryID;
        this.entryInteger = new Integer(entryValue);
    }
}

```

8. EntryLong.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryLong</code> is the entry template for type - Long
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryLong implements Entry {
    public String entryID;
    public Long entryLong;

    public EntryLong(){
    }

    public EntryLong(String entryID){
        this.entryID = entryID;
    }

    public EntryLong(String entryID, long entryValue){
        this.entryID = entryID;
        this.entryLong = new Long(entryValue);
    }
}

```

9. EntryListItem.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryListItem</code> is the entry template for List item
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryListItem implements Entry {
    public String entryID;
    public Long position;
    public String content;

    public EntryListItem(){
    }

    public EntryListItem(String entryID){
        this.entryID = entryID;
    }
}

```

```

    public EntryListItem(String entryID, long position, String msg){
        this.entryID = entryID;
        this.position = new Long(position);
        this.content = msg;
    }

    public EntryListItem(String entryID, long position){
        this.entryID = entryID;
        this.position = new Long(position);
    }
}

```

10 EntryListStatus.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;

/**
 * The <code>EntryListStatus</code> is the entry template for List status
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryListStatus implements Entry {
    public String entryID;
    public Long startIndex;
    public Long endIndex;
    public Long maxSize;

    public EntryListStatus(){
    }

    public EntryListStatus(String entryID){
        this.entryID = entryID;
    }

    public EntryListStatus(String entryID, long maxSize){
        this.entryID = entryID;
        this.maxSize = new Long(maxSize);
        this.startIndex = new Long(0);
        this.endIndex = new Long(-1);
    }

    public long getCurSize(){
        return (endIndex.longValue() - startIndex.longValue() +1);
    }

    public long getMaxSize(){
        return maxSize.longValue();
    }

    public long getStartIndex(){
        return startIndex.longValue ();
    }

    public long getEndIndex(){
        return endIndex.longValue ();
    }

    public boolean isEmpty(){
        if(startIndex.longValue() > endIndex.longValue())
            return true;
        else
            return false;
    }
}

```

```

    public boolean isFull(){
        if(maxSize.intValue() <= (endIndex.longValue() - startIndex.longValue()))
            return true;
        else
            return false;
    }

    public boolean incrementStartIndex(){
        if(startIndex.longValue() <= endIndex.longValue()){
            startIndex = new Long(startIndex.intValue()+1);
            return true;
        }else
            return false;
    }

    public boolean incrementEndIndex(){
        if(maxSize.longValue() > (endIndex.longValue() - startIndex.longValue())){
            endIndex = new Long(endIndex.intValue()+1);
            return true;
        }else
            return false;
    }
}

```

11 EntryQueueItem.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;
/**
 * The <code>EntryQueueItem</code> is the entry template for Queue item
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryQueueItem implements Entry {
    public String entryID;
    public Long position;
    public String content;

    public EntryQueueItem(){
    }

    public EntryQueueItem(String entryID){
        this.entryID = entryID;
    }

    public EntryQueueItem(String entryID, long position, String msg){
        this.entryID = entryID;
        this.position = new Long(position);
        this.content = msg;
    }

    public EntryQueueItem(String entryID, long position){
        this.entryID = entryID;
        this.position = new Long(position);
    }
}

```

12 EntryQueueStatus.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;
/**
 * The <code>EntryQueueStatus</code> is the entry template for Queue status
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000

```

```

*/

public class EntryQueueStatus implements Entry {
    public String entryID;
    public Long startIndex;
    public Long endIndex;
    public Long maxSize;

    public EntryQueueStatus(){
    }

    public EntryQueueStatus(String entryID){
        this.entryID = entryID;
    }

    public EntryQueueStatus(String entryID, long maxSize){
        this.entryID = entryID;
        this.maxSize = new Long(maxSize);
        this.startIndex = new Long(0);
        this.endIndex = new Long(-1);
    }

    public long getCurSize(){
        return (endIndex.longValue() - startIndex.longValue() +1);
    }

    public long getMaxSize(){
        return maxSize.longValue();
    }

    public long getStartIndex(){
        return startIndex.longValue ();
    }

    public long getEndIndex(){
        return endIndex.longValue ();
    }

    public boolean isEmpty(){
        if(startIndex.longValue() > endIndex.longValue())
            return true;
        else
            return false;
    }

    public boolean isFull(){
        if(maxSize.intValue() <= (endIndex.longValue() - startIndex.longValue()))
            return true;
        else
            return false;
    }

    public boolean incrementStartIndex(){
        if(startIndex.longValue() <= endIndex.longValue()){
            startIndex = new Long(startIndex.intValue()+1);
            return true;
        }else
            return false;
    }

    public boolean incrementEndIndex(){
        if(maxSize.longValue() > (endIndex.longValue() - startIndex.longValue())){
            endIndex = new Long(endIndex.intValue()+1);
            return true;
        }else
            return false;
    }
}

```

12 EntryStackItem.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;
/**
 * The <code>EntryStackItem</code> is the entry template for Stack item
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryStackItem implements Entry {
    public String entryID;
    public Long position;
    public String content;

    public EntryStackItem(){
    }

    public EntryStackItem(String entryID){
        this.entryID = entryID;
    }

    public EntryStackItem(String entryID, long position, String msg){
        this.entryID = entryID;
        this.position = new Long(position);
        this.content = msg;
    }

    public EntryStackItem(String entryID, long position){
        this.entryID = entryID;
        this.position = new Long(position);
    }
}
```

14 EntryStackStatus.java

```
package tuplespace.entries;
import net.jini.core.entry.Entry;
/**
 * The <code>EntryStackStatus</code> is the entry template for Stack status
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryStackStatus implements Entry {
    public String entryID;
    public Long startIndex;
    public Long endIndex;
    public Long maxSize;

    public EntryStackStatus(){
    }
    public EntryStackStatus(String entryID){
        this.entryID = entryID;
    }

    public EntryStackStatus(String entryID, long maxSize){
        this.entryID = entryID;
        this.maxSize = new Long(maxSize);
        this.startIndex = new Long(0);
        this.endIndex = new Long(-1);
    }

    public long getCurSize(){
        return (endIndex.longValue() - startIndex.longValue() +1);
    }
}
```

```

    public long getMaxSize(){
        return maxSize.longValue();
    }

    public long getStartIndex(){
        return startIndex.longValue ();
    }

    public long getEndIndex(){
        return endIndex.longValue ();
    }

    public boolean isEmpty(){
        if(startIndex.longValue() > endIndex.longValue())
            return true;
        else
            return false;
    }

    public boolean isFull(){
        if(maxSize.intValue() <= (endIndex.longValue() - startIndex.longValue()))
            return true;
        else
            return false;
    }

    public boolean incrementStartIndex(){
        if(startIndex.longValue() <= endIndex.longValue()){
            startIndex = new Long(startIndex.intValue()+1);
            return true;
        }else
            return false;
    }

    public boolean incrementEndIndex(){
        if(maxSize.longValue() > (endIndex.longValue() - startIndex.longValue())){
            endIndex = new Long(endIndex.intValue()+1);
            return true;
        }else
            return false;
    }
}

```

15 EntryString.java

```

package tuplespace.entries;
import net.jini.core.entry.Entry;
/**
 * The <code>EntryString</code> is the entry template for type - String
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class EntryString implements Entry {
    public String entryID;
    public String entryString;
    public String entryType;
    public EntryString(){
        entryType = "STRING";
    }

    public EntryString(String entryID){
        entryType = "STRING";
        this.entryID = entryID;
    }

    public EntryString(String entryID, String entryString){
        entryType = "STRING";
    }
}

```

```

        this.entryID = entryID;
        this.entryString = entryString;
    }
}

```

16 SpaceActionHandle.java

```

package tuplespace.entries;

import java.util.EventListener;
import java.awt.event.*;
/**
 * The <code>SpaceActionHandler</code>
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public interface SpaceActionHandler extends ActionListener {

    public void actionPerformed(ActionEvent e);
    public void fireAction(int eventType, String id);
}

```

17 SpaceEventListener.java

```

package tuplespace.entries;

import tuplespace.entries.*;
import java.rmi.server.*;
import java.rmi.RemoteException;
import net.jini.core.event.*;
import net.jini.space.JavaSpace;
import java.awt.event.ActionListener;
/**
 * The <code>SpaceEventListener</code>
 *
 * @author Kin Boon Kwang
 * @version 1.0, 01 Oct. 2000
 */
public class SpaceEventListener implements RemoteEventListener {
    private int eventType;
    private String eventID;
    private JavaSpace space;
    private SpaceActionHandler action;

    public SpaceEventListener(int eventType, String eventID, JavaSpace space,
        SpaceActionHandler action) throws RemoteException {
        this.eventType = eventType;
        this.eventID = eventID;
        this.space = space;
        this.action = action;
        UnicastRemoteObject.exportObject(this);
    }

    public void notify(RemoteEvent ev) {
        try {
            action.fireAction(eventType, eventID);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Road, Suite 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Road
Monterey, CA 93943-5101

3. Computer and Information Programs Office 1
Code 32
Naval Postgraduate School
833 Dyer Rd., Room 404
Monterey, California 93943-5120

4. Chair, Computer Science Department..... 1
Naval Postgraduate School
833 Dyer Rd.
Monterey, California 93943-5118

5. Dr Luqi 1
Naval Postgraduate School
833 Dyer Rd.
Monterey, California 93943-5118

6. Dr Valdis Berzins 3
Naval Postgraduate School
833 Dyer Rd.
Monterey, California 93943-5118

7. Dr Ge Jun 2
Naval Postgraduate School
833 Dyer Rd.
Monterey, California 93943-5118

8. Mr. Kin Boon Kwang..... 3
Depot Road, Defense Technology Tower A,
Level 20B, Singapore 109679