

Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	518349 
TITLE	
STK/MATLAB Planning Tools \ (SMPT\) Package for Efficient Programming	
System Number:	
Patron Number:	
Requester:	
Notes:	
DSIS Use only:	
Deliver to: CL	

THIS PAGE IS LEFT BLANK

THIS PAGE IS LEFT BLANK

**DEPARTMENT OF NATIONAL DEFENCE
CANADA**

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT)

DOR(JOINT) RESEARCH NOTE RN 2002/05

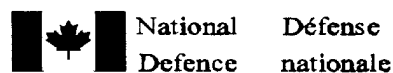
**STK/MATLAB PLANNING TOOLS (SMPT)
PACKAGE FOR EFFICIENT PROGRAMMING**

By

Dr. J. Chan

NOVEMBER 2002

OTTAWA, CANADA



OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT)

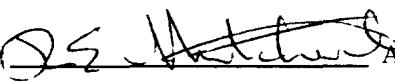
DOR(JOINT) RESEARCH NOTE RN 2002/05

**STK/MATLAB PLANNING TOOLS (SMPT)
PACKAGES FOR EFFICIENT PROGRAMMING**

by

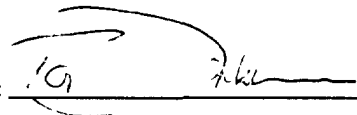
Dr. J. Chan

Recommended by:



R.E. Mitchell
JSORT/TL

Approved by:



R.G. Dickinson
DOR(Joint)

Directorate Research Notes are written to document material which does not warrant or require more formal publication. The contents do not necessarily reflect the views of ORD or the Canadian Department of National Defence.

OTTAWA, ONTARIO

NOVEMBER 2002

ABSTRACT

Satellite Tool Kit (STK) is a sophisticated software tool for modelling and analyzing a wide variety of scenarios involving satellites, sensors, mobile vehicles and stationary facilities. Using the STK/Connect module, the capability of STK can be expanded considerably through the use of MATLAB to perform, for example, trade space analysis studies. By executing some MATLAB routines many times (through a while- or for-loop) with slightly different instructions, a larger number of scenarios can be generated for analysis.

Although the STK/Connect module for MATLAB provides some useful MATLAB functions for building a scenario, they are limited to basic operations. This work describes a new set of MATLAB functions that simplify scenario development and provide the ability to model some common military activities. Examples of these activities would be specifying aircraft search patterns or instructing one vehicle to chase another. The paper also introduces a common framework in MATLAB that can be used to reduce the complexity of large STK scenario development efforts.

The MATLAB functions described in this paper are provided to the readers on a CD. They are offered to the military OR community to aid in the development and analysis of military scenarios using STK. Ideally, the user community will further validate the utility of these functions and perhaps add to this library of functions.

TABLE OF CONTENTS

Abstract..... i

Table of Contents ii

List of Tablesiii

Glossaryiv

I - Introduction1

 Satellite Tool Kit..... 1

 STK/Connect and MATLAB 2

 STK/MATLAB Planning Tools Package..... 2

II - STK and MATLAB Objects.....4

 SMPT Objects in the STK Workspace..... 4

 SMPT Objects in the MATLAB Workspace 5

 Global Variables..... 5

 Object Identifiers..... 7

III - Objects and Information Manupulation 10

 stkDriver..... 12

 Basic Functions 12

 Special Functions 17

IV - Discussion 19

References..... 20

ANNEX A – MATLAB Source Code..... A-1

 stkDriver..... A-3

ANNEX B – ExampleB-1

 stkDriver..... B-3

 demoISR..... B-6

LIST OF TABLES

TABLE I: FUNCTIONS FOR OBJECT CREATION..... 12

TABLE II: FUNCTIONS FOR WAYPOINTS MANIPULATION..... 14

TABLE III: SIMPLE UTILITY FUNCTIONS 15

TABLE IV: FUNCTIONS FOR DATA RETRIEVAL..... 16

TABLE V: SMPT SPECIAL FUNCTIONS 17

GLOSSARY

AGI	Analytical Graphics, Inc.
GUI	Graphical User Interface
LLA	Latitude, Longitude and Altitude
SMPT	STK/MATLAB Planning Tools
SSCN	Space Surveillance Catalog Number
STK	Satellite Tool Kit
TLE	Two-Line Element
UAV	Unmanned Aerial Vehicle

STK/MATLAB PLANNING TOOLS (SMPT) PACKAGE FOR EFFICIENT PROGRAMMING

I - INTRODUCTION

SATELLITE TOOL KIT

1. The Satellite Tool Kit (STK) is a product of Analytical Graphics, Inc. (AGI) [1]. It is widely used in the aerospace and defence industry because of its specialized analytical capability and ease of use. It provides an environment in which one can build a scenario and run a simulation on it. The software automatically performs many important calculations like travel time, position and time at contact point, acceleration/deceleration, and so on.
2. The basic STK application is free of charge. The company makes its profit by releasing many add-on modules for the basic application. These modules include STK/PRO, STK/VO, STK/Connect, STK/Chain, STK/Coverage, STK/Terrain, etc. They enhance the software in different areas. For example, the modules STK/VO and STK/Advanced VO are perhaps the most impressive ones for most people because they provide a 3-dimensional view of any scenario. Another example is the STK/Coverage module which computes the access coverage and other related calculations by a moving or stationary sensor. The STK/Terrain module allows users to import terrain data into a scenario for calculation refinement¹.
3. An STK scenario is usually built by using a graphical user interface (GUI). This GUI allows ones to add objects like facilities, satellites, missiles, sensors, etc in a scenario. Mobile objects are moved either by using propagators (for satellites, missiles, etc.) or by adding waypoints (for aircrafts, ships, etc.) to the objects. Once a scenario has been put together, users can perform analysis on it with the help of the analysis modules.

¹ AGI only provides the module, not the data. Geomatics information for DND purposes can be readily obtained from CF J2 Geo

STK/CONNECT AND MATLAB

4. Although STK is very powerful, it is not a perfect application. Consider the following simple scenario: A helicopter fires a projectile grenade at a stationary target. This scenario has three STK objects, namely an aircraft, a missile and a target. It is so simple that most people can put the objects together within a minute but the scenario could be unrealistic if the point of firing is beyond the line of sight between the helicopter and the target. STK can tell when the target is within the range but it is not as simple as saying 'if see the target, then fire.' The STK GUI simply does not support execution of any conditional statement.

5. Random number generation, which is an essential capability for many programming designs, is another example of an STK deficiency. One can cope with these problems by using STK with third party applications such as C, JAVA, or even MATLAB [2, 3]. They are well developed to meet most programming design challenges. In order to use STK with other applications, one must use an add on module called STK/Connect. This module allows another application to gain control of STK. In other words, STK control instructions can be issued by the application².

6. In the STK community, the STK/Connect module usually works with MATLAB because of its popularity, simplicity and sophistication. In fact, MATLAB has been adopted by the Joint Staff Operational Research Team in ORD to support its work on C4ISR using STK [4]. For these reasons, MATLAB is used in this paper as the third party application of choice to drive the STK computational engine.

STK/MATLAB PLANNING TOOLS PACKAGE

7. The author started using STK and MATLAB when trying to determine the path of a vehicle that chased a moving target. This path must satisfy the condition that the chasee always lies on the tangent of the chaser's path at the chaser's position. In other words, the chaser always heads towards the chasee. When the final STK/MATLAB program *chase* was completed, it was long and complicated. The author realized that if another scenario involved some chasing activities, it would be very difficult to modify the *chase* program to model the new scenario due to its complexity. Therefore the author started breaking the *chase* program into many modules, and eventually splitting the original program into many different functions. Each function served a specific purpose within a larger framework. In this way,

² Detail of this module is only available from the STK software. It can be accessed through Help/Module Help/Connect Help

the chasing activities can be modeled whenever a corresponding MATLAB function is called. Moreover the effort spent on building the *chase* program would not be wasted because some of the original code could be re-used while building the other scenarios.

8. Modulization of the *chase* program turns out to be straightforward compared with the next step of the development. When the number of functions increases, it becomes more difficult to put them within a single framework. For example, determining the location of a ship requires a time as input but locating a (stationary) target does not need this parameter as input. One can create either two functions to find the position for ships and targets, or one function that can handle ships and targets. The author chose the second approach in order to make the functions user-friendly. Moreover the inclusion of more STK object types in the framework adds extra challenge as well. Thus finding a framework that is flexible enough for different STK objects and yet simple enough to be user-friendly is very time-consuming. After much dissatisfaction, the author chose a framework that relied on two global MATLAB variables for communication between the functions. The data-storing variable is flexible enough for future expansion as well.

9. In fact AGI includes some MATLAB functions in the STK/Connect module. These functions are in the *mexConnect* and *AeroToolbox* toolboxes. The author also used them to build his STK/MATLAB functions. Generally speaking, these AGI MATLAB functions deal with some elementary STK operations that are vital in building a scenario.

10. The functions in the STK/MATLAB Planning Tools (SMPT) package are the first step in building a library of MATLAB code that simplifies the programming. By using these functions, a complicated STK scenario can be put together in a short MATLAB scenario M-file. Since an STK scenario is not always self-explanatory, a scenario M-file can be served as a detailed documentation of the scenario to explain the CONOPS behind the model. After all, once a scenario is built, the users can analyse it by both the STK GUI and MATLAB.

11. In the rest of this document, the SMPT functions will be discussed. The next chapter explains the workspace of the two applications, and Chapter III gives a brief explanation of the functions. Some of the MATLAB source code is given in Annex A for reference. The code is designed for STK version 4.3 and MATLAB version 6.5. An example is given in Annex B where an STK scenario is developed from MATLAB code using the SMPT functions. A complete set of source code can be found in a CD attached with this document.

II - STK AND MATLAB OBJECTS

12. As a tool for building scenarios, STK lacks the capability of making any logical decision. On the other hand, MATLAB, which is a general computation software, is not sophisticated enough to determine the motion of a vehicle, especially for satellite motion. The idea of using STK along with MATLAB can bring the best of the two applications together. However the two applications cannot control one another in both directions. One can operate STK from a MATLAB workspace via the STK/Connect module but cannot control MATLAB from the STK workspace. Ironically this limitation reduces some confusion because whenever the MATLAB workspace is discussed here, we refer to a direct use of the application in the MATLAB workspace. This contrasts with the use of the STK workspace which can be manipulated via MATLAB environment or directly through the STK GUI.

13. Prior to discussing the use of STK and MATLAB in Chapter III, the next two sections examine the STK objects considered in this paper and the method by which STK objects are handled by the SMPT functions. In addition, the method by which the connection between MATLAB and STK is managed by the SMPT functions is also discussed.

SMPT OBJECTS IN THE STK WORKSPACE

14. STK is a powerful tool partly because it can model a wide variety of objects. They can be satellites, ships, ground vehicles, aircraft, missiles, radar, sensors, transmitters, receivers, chain, constellation of satellites and so on. One can put these objects together to build a scenario for simulation. Objects like chain and coverage allow the users to perform detailed analysis of a scenario. Nevertheless, a scenario must be built before using any advanced modules for analysis.

15. The SMPT package only deals with some STK objects. They include the STK objects of type:

- a. area target;
- b. target;
- c. facility;
- d. ship;
- e. ground vehicle;
- f. aircraft;

- g. satellite;
- h. sensor;
- i. chain;
- j. constellation;
- k. missiles; and
- l. launch vehicles.

That is to say the SMPT package only provides convenient ways of manipulating these STK objects via MATLAB. One can always add other objects, e.g. coverage, to a scenario either through the STK GUI or STK/Connect command from MATLAB. The package simply does not work with the coverage object at all.

SMPT OBJECTS IN THE MATLAB WORKSPACE

16. In the standard STK/MATLAB environment, an STK aircraft ‘Hercules’ is referred to as ‘*/Aircraft/Hercules’ in the MATLAB workspace. In other words, this MATLAB string refers to an object which only exists in the STK workspace. It is important to distinguish between the STK objects and their counterparts in MATLAB because they are two groups of things in two different working environments. Those MATLAB objects that refer to some STK objects are called *object identifiers* in this document. In this section, these identifiers will be discussed in detail. Besides the object identifiers that are generated when the SMPT package is used, other MATLAB variables are created as well. They will be discussed in the next sub-section.

Global Variables

17. The SMPT package uses two MATLAB variables as global variables. Once these variables are properly defined, they can be used to pass information between the MATLAB functions in the package. The two global variables are *conID* and *STKOBJECTS*.

18. The variable *conID* is an integer to identify a connection between MATLAB and STK. This variable is vital to establish the connection between the software functions. Indeed any STK/Connect users must be familiar with this variable but this package takes one step further by setting it to be a global variable. This step allows the STK scenario objects characteristics to be readily accessible to any of the SMPT functions.

19. The global variable *conID* is much simpler than the second one *STKOBJECTS*. The first variable has the data type of integer but *STKOBJECTS* is a one dimensional structure array of size $1 \times N$ where N is the number of objects in the STK scenario. The structure has several data containers or fields which hold the data defining that particular STK object. Some structure fields have entries describing new objects added to the STK scenario but some fields are just empty. All array elements of this structure will have entries in the fields of *Type*, *Name* and *Path*. These entries are MATLAB strings representing the object's STK type, name and STK path. Either the path or the type-name pair is sufficient to identify an STK object. For example, an STK aircraft called 'Hercules' has a type of 'Aircraft', name of 'Hercules' and path of '* /Aircraft/Hercules'. If this aircraft corresponds to the fourth element in *STKOBJECTS*, the structure array has the entries:

```
STKOBJECTS(4).Type = 'Aircraft'
STKOBJECTS(4).Name = 'Hercules'
STKOBJECTS(4).Path = '* /Aircraft/Hercules'
```

20. Different STK object types have different characteristics. Thus different object types in the structure array *STKOBJECTS* require different fields to store the characteristic data. Where a particular field is not needed to describe an object, the field in that array element is left empty. For example, an STK area target, which defines a region on a map, has the fields of *Lat* and *Lon* in the corresponding array element of *STKOBJECTS*. However an STK target requires an extra field of *Alt* to store the altitude of the object in the corresponding array element of *STKOBJECTS*. Nevertheless the area target also has the field *Alt* but it is always empty. Moreover the fields *Lat* and *Lon* for the two objects in the example are completely different because they are two real numbers for a target but their counterparts for an area target are two row vectors of real numbers because they store the boundary points of an area target.

21. The first structure array element *STKOBJECTS(1)* always represents an STK scenario. In other words, the field *STKOBJECTS(1).Type* must be 'Scenario', and the fields *STKOBJECTS(1).Start* and *STKOBJECTS(1).Stop* represent the scenario start and stop time, respectively.

22. The array element *STKOBJECTS(1)* not only stores scenario characteristic data, it also carries some parameters shared by the other functions. For example, the structure field *STKOBJECT(1) SatDb* contains a file name of a database of satellites. If a satellite is created by the SMPT function *createSatellite*, one should provide a file name to this field. It must be

an ASCII file containing standard two-line element (TLE) sets for each satellite. Records in the file should be sorted by the space surveillance catalog number (SSCN) in ascending order. Alternatively, basic STK provides an unclassified satellite database, which can be found in ...\\STKData\\Databases\\Satellite\\stkSatDb.tce. Nevertheless this field can be ignored if a scenario does not call the function *createSatellite*.

Object Identifiers

23. As was seen before, the structure array index is a natural identifier of any STK object in the MATLAB workspace. In fact, readers with experience in STK/Connect should realize that the identifier of an STK object is always the STK path name of the object. It is obvious that using an integer index is more convenient than using a string of path name.

24. In order to use an index to identify an STK object, the SMPT package utilizes a simple way to assign these indices. For example, if a target is created by the function *createTarget*, an object index will be returned. In the following MATLAB command

```
airport = createTarget('Airport',12,-34,0.56)
```

the variable *airport* receives an integer value from the function *createTarget*, which creates an STK target called 'Airport' at 12° latitude, -34° longitude and 0.56 km altitude. In this way, the variable *airport* can be used as an identifier for the target. Since this object index is simply the structure array index, one can find the value 0.56 at the structure field *STKOBJECTS(airport).Alt*.

25. In the previous example, the function creates not only an STK object but also a new element in the structure array *STKOBJECTS* for the new target. The new element is allocated at *STKOBJECTS(airport)* and contains non-empty entries in the fields of *Type*, *Name*, *Path*, *Lat*, *Lon* and *Alt* with appropriate values. In general, these create-functions create an object in the STK workspace and add a new array element to *STKOBJECTS* in the MATLAB workspace. (A complete list of these create-functions is given in TABLE I in the next chapter.)

26. The identifier can be used in many ways. For example, the latitude of the target created before can be retrieved by the SMPT function

```
getLatitude(airport)
```

where *airport* is the object index.

27. Since STK/Connect uses an object's path to identify the object, all functions in the package accept this way of identification as well. That is to say the command

```
getLatitude('*/Target/Airport')
```

is equivalent to the one in the previous paragraph.

28. In fact, the STK path name is only the second kind of identifier supported by the functions in the package. Consider the following SMPT MATLAB commands:

```
structureAirport = getObject(airport)  
removeObject(structureAirport)  
airport = createTarget(structureAirport)
```

A new 1×1 MATLAB structure array *structureAirport* is created by the function *getObject*. It contains all information about the object *airport* with non-empty structure fields of *Type*, *Name*, *Path*, *Lat*, *Lon* and *Alt*. (The array element *STKOBJECTS(airport)* has an empty field of *SatDb* and other empty fields but the function *getObject* copies only the non-empty fields to *structureAirport*.) The second command line uses this structure array as an identifier to delete the structure array element *STKOBJECTS(airport)* and the corresponding object in the STK workspace. It is equivalent to the command *removeObject(airport)*. After executing the second command, only the array element *STKOBJECTS(airport)* is removed from the MATLAB workspace; the new structure *structureAirport* remains intact because it is only used as an identifier in the second command to point to the appropriate element in *STKOBJECTS*. The last command uses the information residing in *structureAirport* to create a new target which is identical to the one just deleted by the second command. In this example, *structureAirport* works not only as an object identifier but also a definition provider. It is important to notice that the values of *airport* in the first and last commands are not necessarily the same.

29. Finally, the last type of identifier can be illustrated by the following MATLAB commands.

```
controlTower = createFacility('c:\ControlTower.f')  
getLatitude('c:\ControlTower.f')
```

In this example, a pre-defined external file is used to create an STK facility called 'ControlTower'. The second command line shows that the file name is used as an identifier for the facility. This line is equivalent to the command *getLatitude(controlTower)*.

30. Of the four different ways of identifying an STK object in the MATLAB workspace, the most convenient ones are the index identifiers and the path identifiers. In the MATLAB workspace, an integer index is always easy to use. On the other hand, the path identifier is the only way admitted by the standard STK/Connect module. Thus allowing more than one way of identifying an object simply expands the flexibility and ease of use. Users can choose a way that they feel most comfortable with.

31. The other two ways of identifying an object, namely structure identifiers and file name identifiers, are quite different from the other two. A structure not only can identify an STK object but also can store its characteristic data. It can be used to define an STK object as well. Similarly, the file name identifier also defines an object in addition to identifying it. It serves a dual role as was seen in paragraph 29. The details of this will be discussed in the next chapter.

III - OBJECTS AND INFORMATION MANUPULATION

32. As was shown in the last chapter, information about an STK object is stored in the MATLAB structure array *STKOBJECTS* but there are many different fields in it. Retrieving the information from these fields can be cumbersome. In some cases, some useful information is not even stored in the MATLAB workspace at all. For example, the position of a ship at a certain time must be calculated because this piece of information is not readily available. Sometimes instead of retrieving the information, users would like to change the state of some objects. For example, moving an object around or halting a vehicle at a certain time. The SMPT package of MATLAB functions can carry out some of these more common activities. .

33. Functions in the package can be divided into several groups, depending upon their usage. The syntax of all functions can be found by typing the command

help <function name>

For example, the MATLAB command

help getObjectPath

returns the following lines:

getObjectPath Return a string of path name of an object

USAGE 'objectPath' = getObjectPath(objectIndex)
 'objectPath' = getObjectPath('fileName')
 'objectPath' = getObjectPath(objectStruct)
 'objectPath' = getObjectPath(objectIndex, warningSwitch)
 'objectPath' = getObjectPath('fileName', warningSwitch)
 'objectPath' = getObjectPath(objectStruct, warningSwitch)

objectIndex - Integer index of an object
 'fileName' - String name of an STK file
 objectStruct - Structure with some fields identifying an object
 warningSwitch - Integer of 1 or 0 to turn on or off the warning message generated by
 this function
 'objectPath' - String name of an object path

The MATLAB structure "objectStruct" must have the field "Path" or the fields "Type" and "Name" in order to identify the object. If the object is a sensor, the

structure must have the field "Path".

If the integer "warningSwitch" is ignored, the default action is turning on the warning messages. Only the integer 0 can turn off this action.

Each function comes with a brief explanation of the syntax and usage. Therefore the syntax of individual functions will not be explained in this document.

34. As was shown in the previous example, the same function allows three different ways of identifying an object. This is common to all functions in the package. In other words, users are not restricted to a particular form of entering the input. This is important because the same information can always manifest itself in more than one way. Moreover some information has no standard way of representation. For example, when it comes to the position, the latitude, longitude and altitude can be represented by three numbers. However it is equally logical to think of them as a 3-dimensional vector. Thus the functions also allow entering a position as three separated numbers or one 3-dimensional vector, either a column or row vector. In this way, it not only increases the flexibility of using the functions, it also increases the tolerance of the software to user error.

35. When it comes to the output information, the functions always export the information in a certain format. For example, an object is usually represented by its integer index, time is in string format instead of numerical format, and the position is exported as a column vector. If the users were given a choice of controlling the format of the output, the functions would have needed an extra input argument to accommodate the request. In order to keep the functions as simple as possible, their output format is always the same.

stkDriver

36. This is the only MATLAB file in the package that is not a MATLAB function. It is an ordinary M-file. This M-file contains all necessary STK/Connect commands to initialize the procedures of calling STK from MATLAB. Once the two applications are opened, calling this file, by typing

stkDriver

will establish the necessary linkage between the two applications. This will simplify the initialization process. Users are encouraged to insert their customized M-files near the end of this file.

BASIC FUNCTIONS

37. The first group of basic functions includes those that create STK objects. These functions are shown in TABLE I. The use of *createTarget* and *createFacility* was shown in the examples before.

TABLE I: FUNCTIONS FOR OBJECT CREATION

Function Names	Brief Description
assignObjectIndex	Put an STK object in MATLAB workspace and assign an integer index to it
CreateAircraft	Create an STK aircraft
createAreaTarget	Create an STK area target
createChain	Create an STK chain
createConstellation	Create an STK constellation
createFacility	Create an STK facility
createGroundVehicle	Create an STK ground vehicle
createLaunchVehicle	Create an STK launch vehicle
createMissile	Create an STK missile
createSatellite	Create an STK satellite based on an SSC Number
createSensor	Create an STK sensor
createShip	Create an STK ship
createTarget	Create an STK target

38. The first function *assignObjectIndex* in the table is very different from the rest in the table. The other functions create an STK object from the MATLAB workspace, and return an integer index as an identifier. If a user creates an STK object through the STK GUI, MATLAB does not realize the existence of the new object. This problem motivates the introduction of the function *assignObjectIndex*, which assigns a MATLAB identifier (integer index) to an existing STK object, which has no record in *STKOBJECTS*. Thus after the assignment, the new STK object is registered in the MATLAB workspace, and users can use all other functions in the package with the new object.

39. Consider the following example to illustrate the use of these functions:

```
ac1 = createAircraft('Aircraft1',' 07 May 2002 16:00:00.00',45,-36,0.53,0.07);
createAircraft('c:\Aircraft2.ac');
ac3Obj = getObject(ac1);
ac3Obj.Name = 'Aircraft3';
ac3Obj.Start = '08 May 2002 16:00:00.00';
createAircraft(ac3Obj);
addWaypoints(ac3Obj,46,-35,1,0.08,'update');
```

In this example, the first aircraft 'Aircraft1' is explicitly created at (45°, -36°, 0.53 km) with a speed of 0.07 km/sec at the time 1600 on May 7th. The second aircraft 'Aircraft2' is created by using an external file that defines the aircraft. Finally the aircraft 'Aircraft3' is created at 1600 on May 8th at the same position as that of 'Aircraft1'. A second waypoint of (46°, -35°, 1 km, 0.08 km/sec) is added to this plane by the last command. The definition of 'Aircraft3' is borrowed from the structure array element *STKOBJECTS(ac1)* via using the function *getObject*. In this example, the structure *ac3Obj* is used as a definition of an airplane and an identifier.

40. Another group of functions is for waypoint manipulation. They are summarized in TABLE II. Since waypoints are only well defined for some moving vehicles, these functions only apply to the objects of types “Ship”, “GroundVehicle” and “Aircraft”.

TABLE II: FUNCTIONS FOR WAYPOINTS MANIPULATION

Function Names	Brief Description
addWaypoints	Add new waypoints to the end of a vehicle's waypoints
changeLatLon	Change latitudes and longitudes of an object
changeLLA	Change latitudes, longitudes and altitude of an object
changeWaypoints	Change the last few waypoints of a vehicle
setSpeeds	Store the vehicle speeds in the MATLAB workspace
updateWaypoints	Update the STK waypoints of a platform

41. A waypoint in the package always contains four elements: latitude, longitude, altitude and speed. They are measured in degrees, kilometers and kilometers per second. It is possible that after a waypoint is added or changed by a function in TABLE II, the STK workspace does not acknowledge the change. In this case the last function in the table *updateWaypoints* is needed to tell STK to update the waypoint information. Since the time to change one waypoint in STK workspace is similar to the time to change several waypoints, it is more efficient to change as many waypoints as possible in each call to the STK workspace. Thus changing waypoints in MATLAB workspace and updating waypoints in STK workspace are split into two processes.

42. The group of simple utility functions as shown in TABLE III performs some basic STK or MATLAB calculations. For example in a simple case, the functionality of *checkAccess* can be replaced by an STK/Connect command 'Access'. However this function allows one to use the object indices to refer to the sensor and target in contrast to using the string path names while the STK/Connect command is employed. Another example is the function *datestrSTK*. MATLAB has a built-in function called *datestr*, which is similar to *datestrSTK*. The two functions convert numbers into strings but the converted date by *datestr* is rounded to the nearest second. Since this accuracy is insufficient for STK, the function *datestrSTK* converts a numeric date into a string with accuracy of a hundredth of a second.

TABLE III: SIMPLE UTILITY FUNCTIONS

Function Names	Brief Description
checkAccess	Check the access between detector(s) and a target after certain time
datenumSTK	Convert some date/time strings into the MATLAB numeric format
datestrSTK	Convert some numeric date/time into the STK string format
findCombinedPeriod	Find the union of some time intervals
findOverlappedPeriod	Find the intersection of some time intervals
findSeparation	Find the separation between two points at sea level
findStartTime	Find the latest start time among several vehicles
findStopTime	Find the earliest stop time among several vehicles
findTerrainElevation	Find the terrain elevation at a point or a set of points
findTrack	Find the lat., long. and alt. of a vehicle during a time period
findVehicleTLLASR	Find the time, lat., long., alt., speed and turn radius of a vehicle
ft2km	Convert length from feet into kilometers
listObjects	List all STK objects with their indices
removeObject	Remove an STK object from the scenario

43. As the designer of the package, the author has to admit that it could be difficult to find some data from the MATLAB workspace. For integrity, several functions are introduced to retrieve the information. They are listed in TABLE IV.

TABLE IV: FUNCTIONS FOR DATA RETRIEVAL

Function Names	Brief Description
getAltitude	Return the altitude of an object at certain time
getCruiseSpeed	Return the cruise speed of a vehicle
getLatitude	Return the latitude of an object at certain time
getLongitude	Return the longitude of an object at certain time
getMaxSpeed	Return the maximum speed of a vehicle
getMinSpeed	Return the minimum speed of a vehicle
getObject	Return an STK object as a MATLAB structure
getObjectIndex	Return the integer index of an object
getObjectLLA	Return latitude, longitude and altitude of an object at certain time
getObjectName	Return the name of an object
getObjectPath	Return a string of path name of an object
getObjectSpeed	Return the speed of a ship, ground vehicle or aircraft at certain time
getObjectType	Return the STK object type of an object
getParentPath	Return a string of path name of an object's parent
getTLE	Return the information of a two-line element (TLE) of a satellite

44. Most functions in TABLE III and all functions in TABLE IV are very similar in the sense that they all return some information about an object. Indeed it might be appropriate to put them in one table. Nevertheless the functions in TABLE III always return the information that requires some computational effort. On the other hand, the results returned by the functions in TABLE IV are readily available.

SPECIAL FUNCTIONS

45. So far the MATLAB functions discussed before manipulate the STK objects in the same way that the STK GUI does. They can simplify the process of scenario building using MATLAB. For example, waypoints can be added to a vehicle through the SMPT function *addWaypoints*, or using *checkAccess* to see if two objects can see one another. However the users must decide where to put a waypoint before using *addWaypoints*, or decide what action to take after *checkAccess* detects an object. The functions discussed in the previous section do not decide where is the appropriate position for the next waypoint, nor what an object should do after detecting a target.

46. In reality, when a pilot was told to patrol over a region, he would not be given any detailed instructions like where the starting point in the region was, or what the flight pattern should be. However when modelling such a patrol flight, the STK users must fill in all these details before the aircraft can take off. These details could be tedious and sometimes even exacting³. In order to relieve the difficulty, the package includes a few special functions to perform some standard activities in some pre-determined ways. These functions are summarized in TABLE V.

TABLE V: SMPT SPECIAL FUNCTIONS

Functions	Brief Description
chaseVehicle	Modify and add waypoints of a vehicle that chases other vehicles
followVehicle	Add waypoints to a vehicle that meets and then follows another vehicle
patrolRegion	Add waypoints to a vehicle which patrols over a region
stopVehicle	Stop a vehicle at certain time
visitTargets	Add waypoints to a vehicle that visits multiple targets
wander	Create latitudes and longitudes for a wandering route

³ For example, if an aircraft is flying towards a moving ship, the ship must always lay on the tangent of the flight path. Adding the waypoints of such flight path is very demanding.

47. All these functions have some restrictions. For example, if the track pattern is 'perimeter' or 'spiral' for *patrolRegion*, the users cannot choose the orientation of the track. The function will decide patrolling the region in either clockwise or counter-clockwise direction in order to minimize turning of the vehicle. These functions are designed for simplifying the programming for some commonly used activities. They could be a starting point for rewriting other customized functions for specific purposes.

48. The function *visitTargets* allows a vehicle to visit each target once on a list of targets. If there are several targets on the list, there will be more than one way to go through the visits. The order of visits is determined in such a way that the total travel distance is minimized. Determining the order for the least travel distance is an open problem called the Traveling Salesman Problem. This problem quickly becomes unmanageable when the number of targets increases⁴. The package comes with a function called *travelSalesmanProblem* to determine the optimal travel order in the Traveling Salesman Problem [5]. The function employs a Monte Carlo algorithm [5] to search for the solution. However, there is no guarantee that the returned solution is the optimal solution due to the nature of the problem and limited computer resources.

⁴ If there are n targets, there will be $n!$ different ways of visiting them. The total distance of each visit is a sum of $(n-1)$ sub-distances. Thus the optimal order is the one that yields the least total distance among the $n!$ possible ways. If there are 3 targets, there will be 6 possible ways of visit. If the target number is raised to 6, the number of possible ways becomes 720. Another two-fold to 12 targets yields 479001600 combinations.

IV - DISCUSSION

49. Creating STK scenarios by using MATLAB is much more difficult than doing it with STK alone via the STK GUI. It is however easy to see that the utility of combining these two applications will repay the initial effort. By introducing the SMPT package and offering it to the modelling and simulation community, it is hoped that this will reduce the difficulties inherent in modelling complex scenarios. By using the functions in the package, programmers can reduce the effort and time in creating a scenario. This is especially useful when users can call some functions to generate some frequently occurring activities. This helps in understanding the logic flow within the program and in utilizing repeated use of some routines.

50. While the number of STK users is increasing within DND, the demand of sophisticated STK scenarios, or even STK/MATLAB programs is also on the rise. Users will be able to easily exchange STK scenarios for collaborative analysis or repeating legacy analyses. With this reuse of information, the problem of releasability of information emerges. This will be true even within the DND environment. If STK is used alone, it will be difficult to separate classified data from a scenario. This problem can be alleviated if MATLAB and STK are used together. Classified information can then be imported to an unclassified STK/MATLAB model to generate the classified STK model. Separating code from data also allows STK/MATLAB programmers of different projects to share functions. Hence the library of the functions can increase to cover a wider area of interest for the entire modelling and simulation community.

51. It is evident that STK does not provide any simple way for the users to document a scenario. For example, one can witness a missile launched at 1600 but he does not have any idea why the missile was launched at that time. For a more complicated scenario, this problem becomes worse. Since a MATLAB program is not difficult to read, one can understand the CONOPS behind a scenario by reading its STK/MATLAB code. This makes the code an ideal documentation of an STK scenario and alleviates considerable effort in reusing the scenario for follow-on work.

REFERENCES

1. <http://www.stk.com/>
2. <http://www.mathworks.com/>
3. *Mastering MATLAB 5: A Comprehensive Tutorial and Reference*, D. Hanselman and B. Littlefield, Prentice Hall, 1998.
4. *C4ISR Modelling Capability using STK and MATLAB: A Practical demonstration of Capability*, R.E. Mitchell, DOR(Joint) Research Note RN 2001/12, November 2001.
5. *Operations Research. Applications and Algorithms*, 3rd Ed., W.L. Winston, Duxbury Press, 1994.

ANNEX A
DOR(JOINT) RESEARCH NOTE RN 2002/05
NOVEMBER 2002

ANNEX A – MATLAB SOURCE CODE

1. The source code of the M-file *stkDriver* is included in this annex. The source code of the functions that are listed in TABLE I to TABLE V is not given here; it is put in a CD attached with this document for convenience. The source code of *stkDriver* and the function *travelSalesmanProblem* is also included in the CD.

THIS PAGE INTENTIONALLY LEFT BLANK

stkDriver

```

% stkDriver          This M-file creates an STK scenario without any objects (a plain scenario)
%
% This M-file creates a plain STK scenario
%
% This program was created by Jim Chan
% DGOR/JSORT
% NDHQ Ottawa
% (613) 996-6511
% October 2001
%-----%

clear all;
clc;
tic

stkInit

global conID STKOBJECTS;

%-----%

randomState = round(sum(100*clock));
disp([filename,': Random number state: ',num2str(randomState)]);
rand('state',randomState);
%rand('state',0);

scenarioName      = 'MyScenario';
scenarioStartTime = floor(now);
scenarioStopTime  = floor(now) + 1;
epochTime        = scenarioStartTime;
satelliteDatabase = [stkHomeDir,'\STKData\Databases\Satellite\stkSatDb.tce'];

%-----%

hostPort = stkDefaultHost;
conID    = stkOpen(hostPort);
if ( stkValidScen )

```

```

disp([filename,': Removing old STK scenario.']);
stkUnload('*/*');
end
disp([filename,': New scenario "',scenarioName,'" is being created.']);
stkNewObj('/', 'Scenario', scenarioName);

scenarioStartTime = datestrSTK(scenarioStartTime);
scenarioStopTime = datestrSTK(scenarioStopTime);
epochTime = datestrSTK(epochTime);
scenarioPath = ['/Scenario/', scenarioName];
stkSetTimePeriod(scenarioStartTime, scenarioStopTime, 'GREGUTC');
stkSetEpoch(epochTime, 'GREGUTC');
stkSyncEpoch;
stkExec(conID, ['Animate ', scenarioPath, ' SetAnimationMode XRealTime']);
stkExec(conID, ['Animate ', scenarioPath, ' XRealTimeMultiplier 1.0']);
%stkExec(conID, ['Animate ', scenarioPath, ' SetAnimationMode Normal']);
stkExec(conID, ['Animate ', scenarioPath, ' SetValues "', scenarioStartTime, " ', ...
'60 0.1 " ', scenarioStopTime, " '"]);
stkExec(conID, ['Animate ', scenarioPath, ' Start End']);
stkExec(conID, ['Animate ', scenarioPath, ' Reset']);
stkExec(conID, ['SetUnits ', scenarioPath, ' Kilometer Second Gregorian']);

STKOBJECTS(1).Type = 'Scenario';
STKOBJECTS(1).Name = scenarioName;
STKOBJECTS(1).Path = scenarioPath;
STKOBJECTS(1).Start = scenarioStartTime;
STKOBJECTS(1).Stop = scenarioStopTime;
if ( exist('satelliteDatabase') )
    STKOBJECTS(1).SatDb = satelliteDatabase;
else
    STKOBJECTS(1).SatDb = [];
end
%-----
%
% Customized MATLAB M-files are put here.
%-----

```

```
disp([mfilename,': Scenario "',scenarioName,'" is completed.']);  
%stkSaveObj(scenarioPath);  
stkClose(conID);  
  
toc
```

THIS PAGE INTENTIONALLY LEFT BLANK

ANNEX B
DOR(JOINT) RESEARCH NOTE RN 2002/05
NOVEMBER 2002

ANNEX B – EXAMPLE

1. This annex contains an example of using the SMPT functions to build a scenario. A scenario M-file called *demoISR* is attached at the end of this annex. This M-file is executed by a modified *stkDriver* routine, which is also attached here. (The changed lines in the modified *stkDriver* are in bold face compared with the one on page A-3.) The attached CD contains the source code of *demoISR* but not the source code of the modified *stkDriver*.
2. In this example, an unmanned aerial vehicle (UAV) is sent from a home base to patrol over a region. Some intruders will come across the region trying to reach a port. Their tracks are randomly created. The UAV is equipped with a sensor to detect the intruders. If an intruder is detected, the UAV will follow it. Half an hour after the detection, an interceptor will be sent from the home base to intercept the intruder. Since the intruder is far from the home base, the interceptor cannot detect it. It relies on the UAV that provides the location of the intruder while tailing it. If the interceptor can catch up with the intruder, it hovers over the interception point for a while, then returns to the home base. (The intruder is eliminated from the scenario after being caught.) On the other hand, once the interceptor catches the intruder, the UAV stops tailing and resumes the patrol all over again. In some cases, if the intruder reaches the port before being caught by the interceptor, the interceptor abandons the pursuit and returns to the home base, and the UAV stops following and restarts the patrol.
3. It is conceivable that when the UAV resumes patrol after tailing an intruder, it could detect another intruder. When this happens, the UAV follows the intruder as before. Assuming that the home base has only one interceptor, if the interceptor is not in the air, it will be sent from the home base thirty minutes later to chase the intruder. If the interceptor is on its way home, it will be immediately changed its heading to pursuit the intruder. If the interceptor is hovering above a previously detected intruder for investigation, no interceptor will pursuit the newly detected intruder. After the investigation, the interceptor starts chasing the newly detected intruder if it has not yet reached the port. When the interceptor catches the intruder, the UAV stops tailing and starts another round of patrol over the region.
4. The process repeats until the UAV completes patrolling the whole region without

detecting any new intruders, or the UAV reaches the limit of its endurance. In either case, the UAV is programmed to return to the home base.

5. As was explained before, this MATLAB scenario file *demoISR* can create very different events from one execution to another execution. The variation comes from the randomness nature of the intruders' tracks, and the hovering duration. This makes different encounters possible between the UAV, intruders and interceptor. By running this scenario file many times⁵, one can obtain the statistics for the detection rate of the UAV, or the distribution of the interception rate, or other statistics of interest. The analysis of the scenario is not included in *demoISR* because it is beyond the scope of the discussion.

6. Readers are reminded that the algorithm of the scenario file *demoISR* is quite complicated. However the complication comes from the fact that the CONOPS, as explained before, allows many different possible encounters. As each possible encounter has to be taken care of, *demoISR* becomes hard to read. Nevertheless, without using the SMPT functions, complicated CONOPS like this one is very difficult to implement.

⁵ Instead of running the M-file many times, one can modify it by adding a loop in appropriate places. In this way, many STK objects no longer need to create and destroy over and over again

stkDriver

```

% stkDriver          This M-file creates an STK scenario without any objects (a plain scenario)
%
% This M-file creates a plain STK scenario
%
% This program was created by Jim Chan
% DGOR/JSORT
% NDHQ Ottawa
% (613) 996-6511
% October 2001
%-----%

clear all;
clc;
tic

stkInit

global conID STKOBJECTS;

%-----%

randomState = round(sum(100*clock));
disp([filename,': Random number state: ',num2str(randomState)]);
rand('state',randomState);
%rand('state',0);

scenarioName = 'demoISR';
scenarioStartTime = floor(now);           % Midnight of today
scenarioStopTime = floor(now) + 40/24;    % 40 hours later
epochTime = scenarioStartTime;
satelliteDatabase = [stkHomeDir,'\STKData\Databases\Satellite\stkSatDb.tce'];

%-----%

hostPort = stkDefaultHost;
conID = stkOpen(hostPort);

```

```

if ( stkValidScen )
    disp([filename,': Removing old STK scenario.']);
    stkUnload('*/*');
end
disp([filename,': New scenario "',scenarioName,'" is being created.']);
stkNewObj('/', 'Scenario', scenarioName);

scenarioStartTime = datestrSTK(scenarioStartTime);
scenarioStopTime = datestrSTK(scenarioStopTime);
epochTime = datestrSTK(epochTime);
scenarioPath = ['/Scenario/', scenarioName];
stkSetTimePeriod(scenarioStartTime, scenarioStopTime, 'GREGUTC');
stkSetEpoch(epochTime, 'GREGUTC');
stkSyncEpoch;
stkExec(conID, ['Animate ', scenarioPath, ' SetAnimationMode XRealTime']);
stkExec(conID, ['Animate ', scenarioPath, ' XRealTimeMultiplier 1.0']);
%stkExec(conID, ['Animate ', scenarioPath, ' SetAnimationMode Normal']);
stkExec(conID, ['Animate ', scenarioPath, ' SetValues "', scenarioStartTime, " ', ...
    '60 0.1 "', scenarioStopTime, "']);
stkExec(conID, ['Animate ', scenarioPath, ' Start End']);
stkExec(conID, ['Animate ', scenarioPath, ' Reset']);
stkExec(conID, ['SetUnits ', scenarioPath, ' Kilometer Second Gregorian']);

STKOBJECTS(1).Type = 'Scenario';
STKOBJECTS(1).Name = scenarioName;
STKOBJECTS(1).Path = scenarioPath;
STKOBJECTS(1).Start = scenarioStartTime;
STKOBJECTS(1).Stop = scenarioStopTime;
if ( exist('satelliteDatabase') )
    STKOBJECTS(1).SatDb = satelliteDatabase;
else
    STKOBJECTS(1).SatDb = [];
end
%-----
%
% Customized MATLAB M-files are put here.

```

demoISR

```
%-----%  
  
disp([mfilename,': Scenario "',scenarioName,'" is completed.']);  
%stkSaveObj(scenarioPath);  
stkClose(conID);  
  
toc
```

demoISR

```

% demoISR          This M-file describes the CONOPS of a simple ISR scenario

% This program was created by Jim Chan
% DGOR/JSORT
% NDHQ Ottawa
% (613) 996-6511
% November 2001
%-----%
%-----%
% Initialize all objects that will be used in this scenario.
% Most of them are created at this time.

uavEndurance      = 36;           % In hours
numIntruders      = 60;           % Number of intruders
numInterceptors   = numIntruders; % Number of interceptors. This is the number of STK aircraft
                                % needed to model several trips of the same interceptor.

lat = [ 45.5522  45.6045  45.2413  44.8843  44.9273  45.1366];
lon = [-61.2488 -60.6831 -60.1976 -60.6751 -61.3371 -61.5458];
aoi = createAreaTarget('Area_of_Interest',lat,lon,'green');

home = createTarget('Home',44.65,-63.6,0.045,'yellow'); % LLA in degrees and kilometers

uavStartTime = datenum(scenarioStartTime);
uav          = createAircraft('UAV',uavStartTime,getObjectLLA(home),0,'white');
setSpeeds(uav,0.02778,0.03611,0.05139); % Speed in kilometers per second
uavAlt       = 7.6;
addWaypoints(uav,getLatitude(home)+0.02,getLongitude(home)+0.02,uavAlt,getCruiseSpeed(uav),'u');

viewWidth      = 2;
swathWidth     = 2;
swathAngle     = atan(swathWidth / 2 / uavAlt) / pi * 180;
swingAngle     = swathAngle - viewWidth / 2;
spinRate       = 0.5;
definition     = ['Rectangular ',num2str(viewWidth),' 2'];

```

```

pointing = ['Spinning 0 0 90 Bidirectional ', num2str(180-swingAngle), ' ', ...
            num2str(180+swingAngle), ' ', num2str(spinRate), ' 0'];

uavSensor = createSensor(uav, 'Swing', definition, pointing, 'white');

target = createTarget('Intruder_Target', 45.27, -61.03, 0, 'yellow'); % LLA in degrees and kilometers

averageInterval = 1.0/24; % Average intruder appearance interval in days
intruderStartTime = datenum(scenarioStartTime);
targetLat = getLatitude(target);
targetLon = getLongitude(target);
for i = 1:numIntruders
    name = ['Intruder_', num2str(i)];
    intruderStartTime = intruderStartTime - averageInterval * log(rand); % Exp distribution
    clear intruderWaypoints;
    [lat lon] = reckon(targetLat, targetLon, km2deg(100), 60+180*rand);
    intruderWaypoints = wander(lat, lon, targetLat, targetLon);
    lat = intruderWaypoints(1,:);
    lon = intruderWaypoints(2,:);
    alt = zeros(1, length(lat));
    spd = 0.006 + 0.004 * rand(1, length(lat));
    intruders(1) = createShip(name, intruderStartTime, lat, lon, alt, spd, 'LineOfSight off', 'cyan');
end

% The interceptor characteristics are defined here. However it is not created at this moment because
% the creation time is unknown at this stage.
interceptorData.Lat = [getLatitude(home) getLatitude(home)+0.02];
interceptorData.Lon = [getLongitude(home) getLongitude(home)+0.02];
interceptorData.Alt = [getAltitude(home) 3.0];
interceptorData.Speeds = [0.06 0.07 0.08]; % Minimum, cruise and maximum speed of the interceptor
interceptorData.Spd = [0 interceptorData.Speeds(3)];

% The sensor characteristics are defined here. However it is not created at this moment because
% its parent platform "interceptor" has not been created yet.
interceptorSensorData.Name = 'SimpleCone';
interceptorSensorData.Definition = 'SimpleCone 35';
interceptorSensorData.Pointing = 'Fixed AzEl 0 60';
interceptorSensorData.Constraint1 = 'MaxElevation 60';
interceptorSensorData.Constraint2 = 'MinElevation 10';
interceptorSensorData.Constraint3 = 'MaxRange 10800';

```

```

%-----%
%
% Fly a UAV to patrol over an area of interest (AOI), where intruders appear frequently.
% When the UAV detects an intruder, it starts following it and notifies home to send an
% interceptor. When the interceptor arrives, the UAV leaves the intruder and start patrolling
% the AOI all over again. The whole process repeats until the UAV completes patrolling the
% whole region or reaches it endurance limit.
%-----%
%
% The UAV intends to fly over the whole region of AOI.
numWaypoints = addWaypoints(uav,44.864097,-60.691764,uavAlt,getCruiseSpeed(uav),'u');
patrolRegion(uav,aoi,'S',getCruiseSpeed(uav),swathWidth);
patrolWaypoints(1,:) = STKOBJECTS(uav).Lat(numWaypoints+1:end); % Store the waypoints of the uav
patrolWaypoints(2,:) = STKOBJECTS(uav).Lon(numWaypoints+1:end); % when it is patrolling over the
patrolWaypoints(3,:) = STKOBJECTS(uav).Alt(numWaypoints+1:end); % AOI. Waypoints before this patrol
patrolWaypoints(4,:) = STKOBJECTS(uav).Spd(numWaypoints+1:end); % are not stored in the array.
%-----%
%
uavReadyTime = uavStartTime; % Time when UAV is ready for detecting intruders
interceptorRestTime = uavReadyTime; % Time when interceptor arrives home for a rest
undetectedIntruders = intruders; % A set of intruders defined before
interceptorTrip = 0;

stopLoop = 0;
while ( ~stopLoop ) % Stop the loop until the UAV detects no intruders
%-----%
%
% The UAV tries to chase the first intruder it detected after the time "uavReadyTime".
theIntruder = 0;
uavContactTime = 1095000; % Correspond to a remote future date 02 Jan 2998 00:00:00
for i = 1:length(undetectedIntruders)
    accessTime = checkAccess(uavReadyTime,uavSensor,undetectedIntruders(i));
    if ( ~isempty(accessTime.Start) )
        numFirstAccessTime = datenum(accessTime.Start{1});
        if ( numFirstAccessTime < uavContactTime )
            theIntruder = undetectedIntruders(i);

```

```

        uavContactTime = numFirstAccessTime;
        theIntruderIndex = i;
    end
end
if ( theIntruder == 0 )
    chaseFlag = 0;
    uavFlightTime = 24 * (datenum(findStopTime(uav)) - uavStartTime);
    if ( uavFlightTime > uavEndurance )
        uavStopTime = uavStartTime + uavEndurance / 24;
        stopVehicle(uav,uavStopTime);
    end
else
    uavFlightTime = 24 * (uavContactTime - uavStartTime);
    if ( uavFlightTime >= uavEndurance )
        chaseFlag = 0;
        uavStopTime = uavStartTime + uavEndurance / 24;
        stopVehicle(uav,uavStopTime);
    else
        undetectedIntruders(theIntruderIndex) = [];
    end
end
%-----%
%
% After detection, the UAV's swing sensor should be in tracking mode.
% A new sensor is added to the UAV to model the tracking mode but please bear in mind that
% there is always only one sensor on board of the UAV.
    definition = ['Rectangular ',num2str(viewWidth),' 1'];
    pointing = ['Targeted Tracking ',getObjectType(theIntruder),'/',...
        getObjectNames(theIntruder)];
    uavTrackSensor = createSensor(uav,'Track',definition,pointing,'white');
    [chaseFlag theIntruder uavChaseStopTime] = chaseVehicle(uav,theIntruder,...
        uavTrackSensor,...
        uavContactTime,...
        getMaxSpeed(uav));
end
end
if ( chaseFlag == 0 )
    stopLoop = 1;
else

```

```

%-----%
% If an intruder is detected, the UAV follows it and provides the location of it to the base.
    followVehicle(theIntruder,uav,uavTrackSensor,datenumSTK(uavChaseStopTime,-0.01));
%-----%

% When the UAV just detects the intruder, it tells the base to send an interceptor
% to check out the intruder. We assume that the base only has ONE interceptor.
    if ( uavContactTime > interceptorRestTime )
%-----%
% The interceptor is at the home base for a rest. Ready to take-off to chase the intruder.
        interceptorTrip      = interceptorTrip + 1;
        interceptorData.Name  = ['interceptor_Trip',num2str(interceptorTrip)];
        interceptorData.Start = datestrSTK(uavContactTime + 0.5/24); % Preparation: 30 min
        interceptor          = createAircraft(interceptorData,'red');
        interceptorSensor     = createSensor(interceptor,interceptorSensorData,'red');
        interceptorChaseStartTime = findStopTime(interceptor);
        [chaseFlag chasee chaseEndTime] = chaseVehicle(interceptor,theIntruder,...
            [interceptorSensor;uavTrackSensor],...
            interceptorChaseStartTime,...
            getMaxSpeed(interceptor));

        elseif ( uavContactTime >= interceptorHomeTime )
%-----%
% The interceptor is on the way home after checking out a detected intruder.
            [chaseFlag chasee chaseEndTime] = chaseVehicle(interceptor,theIntruder,...
                [interceptorSensor;uavTrackSensor],...
                uavContactTime,...
                getMaxSpeed(interceptor));

        else
%-----%
% The interceptor is checking out a detected intruder.
% It will chase after the newly detected intruder later.

```

```

stopVehicle(interceptor,datestrSTK(interceptorHomeTime,1));
[chaseFlag chasee chaseEndTime] = chaseVehicle(interceptor,theIntruder,...
    [interceptorSensor;uavTrackSensor],...
    interceptorHomeTime,...
    getMaxSpeed(interceptor));

end

%-----%
% The interceptor prepares for its return trip.
%-----%
if ( chaseFlag == 1 )
%-----%
% The interceptor has found the intruder.
% It will hover at the intercept point for a while before going home.
    stopVehicle(theIntruder,chaseEndTime);
    hoverTime      = -2 * log(rand);           % Hover time in hours
    interceptorHomeTime = datenum(patrolRegion(interceptor,getObjectLLA(interceptor),...
        'Hover',getMaxSpeed(interceptor),hoverTime));
else
%-----%
% The interceptor cannot find the intruder. It will go home right away.
    interceptorHomeTime = datenum(chaseEndTime);
end
addWaypoints(interceptor,getLatitude(home)+0.02,getLongitude(home)+0.02,...
    getAltitude(interceptor),getCruiseSpeed(interceptor));
addWaypoints(interceptor,getObjectLLA(home),0,'u');
interceptorRestTime = datenum(findStopTime(interceptor));

%-----%
% Determine the time when the UAV resumes its patrol over the region AOI.
    if ( chaseFlag )

```

```

        accessTime = checkAccess(uavReadyTime,interceptorSensor,theIntruder);
        if ( ~isempty(accessTime.Start) )
            uavReadyTime = datenum(accessTime.Start{1});
        else
            uavReadyTime = datenum(chaseEndTime);
        end
        stopVehicle(uav,uavReadyTime);
    else
        uavReadyTime = datenum(chaseEndTime);
    end
    uavFlightTime = 24 * (uavReadyTime - uavStartTime);
    if ( uavFlightTime >= uavEndurance )
        stopLoop = 1;
    else
        addWaypoints(uav,patrolWaypoints(1:3,:),patrolWaypoints(4,:), 'u');
    end
end
end

%-----%
%
% Remove UAV's tracking sensor, which is not a real sensor on board of the aircraft.
if ( exist('uavTrackSensor') )
    removeObject(uavTrackSensor);
end

%-----%
%
% After the patrol, the UAV returns home.
addWaypoints(uav,getLatitude(home)+0.02,getLongitude(home)+0.02,uavAlt,getCruiseSpeed(uav));
addWaypoints(uav,getObjectLLA(home),0,'u');

```

UNCLASSIFIED
 Security Classification of Form
 (Highest Classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1 ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8) Operational Research Division Department of National Defence Ottawa, Ontario K1A 0K2	2 SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable) UNCLASSIFIED		
3 TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title) STK/MATLAB Planning Tools (SMPT) package for efficient programming			
4 AUTHORS (last name, first name, middle initial) Chan, Jim			
5 DATE OF PUBLICATION (month Year of Publication of document) November 2002	<table style="width: 100%; border: none;"> <tr> <td style="width: 60%; border: none; padding: 5px;"> 6a NO OF PAGES (total containing information. Include Annexes, Appendices, etc.) 51 </td> <td style="width: 40%; border: none; padding: 5px;"> 6b NO OF REFS (total cited in document) 5 </td> </tr> </table>	6a NO OF PAGES (total containing information. Include Annexes, Appendices, etc.) 51	6b NO OF REFS (total cited in document) 5
6a NO OF PAGES (total containing information. Include Annexes, Appendices, etc.) 51	6b NO OF REFS (total cited in document) 5		
7 DESCRIPTIVE NOTES (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) Research Note			
8 SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address)			
9a PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b CONTRACT NO. (if appropriate, the applicable number under which the document was written.)		
10a. ORIGINATOR's document number (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) DOR(Joint) Research Note RN 2002/05	10b OTHER DOCUMENT NOS. (Any other numbers which may be assigned to this document either by the originator or by the sponsor.) ---		
11 DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification) ☒ (X) Unlimited distribution ☐ () Distribution limited to defence departments and defence contractors, further distribution only as approved ☐ () Distribution limited to defence departments and Canadian defence contractors, further distribution only as approved ☐ () Distribution limited to government departments and agencies, further distribution only as approved ☐ () Distribution limited to defence departments, further distribution only as approved ☐ () Other (please specify):			
12 DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)			

UNCLASSIFIED
 Security Classification of Form
 (Highest Classification of Title, Abstract, Keywords)

UNCLASSIFIED

Security Classification of Form

(Highest Classification of Title, Abstract, Keywords)

13 ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual)

Satellite Tool Kit (STK) is a sophisticated software tool for modelling and analyzing a wide variety of scenarios involving satellites, sensors, mobile vehicles and stationary facilities. Using the STK/Connect module, the capability of STK can be expanded considerably through the use of MATLAB to perform, for example, trade space analysis studies. By executing some MATLAB routines many times (through a while- or for-loop) with slightly different instructions, a larger number of scenarios can be generated for analysis.

Although the STK/Connect module for MATLAB provides some useful MATLAB functions for building a scenario, they are limited to basic operations. This work describes a new set of MATLAB functions that simplify scenario development and provide the ability to model some common military activities. Examples of these activities would be specifying aircraft search patterns or instructing one vehicle to chase another. The paper also introduces a common framework in MATLAB that can be used to reduce the complexity of large STK scenario development efforts.

The MATLAB functions described in this paper are provided to the readers on a CD. They are offered to the military OR community to aid in the development and analysis of military scenarios using STK. Ideally, the user community will further validate the utility of these functions and perhaps add to this library of functions.

14 KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project index name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

STK
STK/Connect
MATLAB

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

(Highest classification of Title, Abstract, Keywords)

Canada

518349

0 000 000