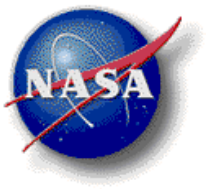
Core Flight System Command and Data Dictionary Tool Developer's Guide

Engineering Directorate
Software, Robotics, and Simulation Division

Version 2.0.24
November 2020



National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas 77058-3696

Table of Contents

1.0 Overview

The Core Flight System Command and Data Dictionary (CCDD) is a software tool for managing the command and telemetry data for CFS and CFS applications. CCDD is written in Java™ and interacts with a PostgreSQL database, so it can be used on any operating system that supports the Java Runtime Environment (JRE) and PostgreSQL. CCDD is released as open source software under the NASA Open Source Software Agreement, version 1.3, and is hosted on GitHub.

This document describes how scripts can be developed to extract data from the CCDD databases. It also provides example code in the supported scripting languages and various methods of extracting data to create the desired data products.

Questions or comments concerning this document or the CCDD application can be addressed to:

Johnson Space Center
Software, Robotics, and Simulation Division
Spacecraft Software Engineering Branch, Mail Code ER6
Houston, TX 77058

2.0 Scripts

The CCDD application's script interface is the mean by which a project's data, stored in the database, is made available for manipulation by the user, primarily for formatting the data to create output files. CCDD supports the use of JVM-based scripting languages. Five of these languages, JavaScript, Python, Ruby, Groovy, and Scala, have been tested with the CCDD application, though any of the other compliant scripting languages should work as well. A language must be installed before it can be used by CCDD. The CCDD Installation Guide provides details on the library files required for using each of the five tested scripting languages. The CCDD About dialog displays a list of the installed scripting languages. Examples of the use of scripts to produce output files include the creation of:

- C header files for CFS applications
- CFS Housekeeping copy table
- ITOS record and display files
- JSON files for CTF tool

Scripts may be executed from within the tool (see CCDD User's Guide) or from the command line (see Section 3.0).

The scripts have access to the project data via a set of script data access methods written in Java. Additional methods are provided for displaying dialog boxes (both output and input), opening and writing to an output file, and making direct queries to the database. The methods are called from within a script using the method name and, dependent on the language, prepended by the class name ccdd or ccdds:

```
<ccdd or ccdds.>methodName(arguments…)
```

where `methodName` is the name of the script data access method (function) and `argument` are the parameters required by the particular method. `ccdd` is a reference to the non-static version of the script data access class, whereas `ccdds` is a static reference to the non-static class' methods. JavaScript scripts require the non-static reference in Java 8, but can use either in Java 7. Ruby scripts require the non-static reference, but Python and Groovy scripts can use either. Scala scripts must use the static reference; however, the class name is not used in the script. Details on the script data access methods are provided in Table 5-2.

In order to access these methods the script requires that the data access class (non-static or static version) be imported; the import statement format is dependent on the scripting language. The following paragraphs show the import statement required to be included in the script file for each of the tested scripting languages, as well as an example of using the script data access methods. For each scripting language the example accomplishes the same result and assumes one or more structure tables are associated with the script (see CCDD Users' Guide for information on associating scripts with data tables). First, the script opens an output file names "`myFileName`". Then the names of the structures present in the structure table(s) supplied to the script are stored in an array named "`structNames`". A loop is then performed to write each structure's name to the output file. Finally, the output file is closed and the script terminates, returning control to the CCDD application. A status message is written to the event log to indicate script completion.

If an error occurs, preventing successful script completion, an entry is made in the event log displaying the script name(s) and provides details on the cause of the error. The amount of detail provided depends on the scripting language. This can be improved by the use of exception catching in the script. The syntax is language dependent, but in general one or more sections of the script code is encompassed by a try-catch statement (usually the main portion and not any functions). An exception, caused by an error condition in the script, is caught. Data, such as the execution trace or variable values, can be included in the text that is returned to CCDD as the cause of the failure, which is then included in the event log entry. Information on the specific syntax is given in the following sub-sections.

2.1 JavaScript

JavaScript script files must end with the extension ".js". The JavaScript script must contain the following lines at or near the top of the file (this allows the script to work with both JavaScript 'Rhino' (Java 7 and earlier) and 'Nashorn' (Java 8 and later)):

```
try
{
    load("nashorn:mozilla_compat.js");
}
catch (e)
{
}
importClass(Packages.CCDD.CcddScriptDataAccessHandler);
```

If enhanced error logging is desired then encompass the script code with a try-catch statement as shown below. The throw call output can be replaced or have other text appended if desired.

```
import traceback
try
{
    # Main script steps
    .
    .
}
catch (err)
{
    throw err.name + " " + err.message + " " + err.stack;
}
```

The following is the example script described earlier in this section, written in JavaScript:

```
// Import the script data access method class
try
{
    load("nashorn:mozilla_compat.js");
}
catch (e)
{
}
importClass(Packages.CCDD.CcddScriptDataAccessHandler);

// Open the output file
var file = ccdd.openOutputFile("myFileName");

// Get the array of structure names
var structNames = ccdd.getStructureTableNames();

// Step through each name found
```

```
for (var index = 0; index < structNames.length; index++)
{
// Write the structure name to the output file
ccdd.writeToFileLn(file,
                   "structNames["
                   + index
                   + "] = "
                   + structNames[index]);
}

// Close the output file
ccdd.closeFile(file);
```

2.2 Python

Python script files must end with the extension ".py". The Python script must contain the following line at or near the top of the file:

```
from CCDD import CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a try-except statement as shown below. The raise call output can be replaced or have other text appended if desired.

```
import traceback

try:
    # Main script steps
    .
    .
except:
    raise Exception(traceback.format_exec())
```

The following is the example script described earlier in this section, written in Python:

```
# Import the script data access method class
from CCDD import CcddScriptDataAccessHandler

# Open the output file
file = ccdd.openOutputFile("myFileName")

# Get the array of structure names
structNames = ccdd.getStructureTableNames()

# Step through each name found
for index in range(len(structNames)):

# Write the structure name to the output file
ccdd.writeToFileLn(file, "structNames[" + str(index) + "] = " +
structNames[index])
```

```
# Close the output file
ccdd.closeFile(file)
```

2.2.1 Calling Other Scripts

It may be desirable for the main Python script called by the association to in turn call another Python script. In order for the 'child' script to access the script data access methods the following can be done:

- Create a folder representing the Python script package.
- Place the Python scripts in the package folder. The child script(s) must be placed in the package folder, but the main script does not.
- Create the file `__init__.py` in the package folder. In this file add an import statement for each child script in the form '`from child import *`' where *`child`* is the child script name (minus the .py extension).
- In the child scripts add '`from CCDD import CcddScriptDataAccessHandlerStatic as ccdd`'. Notice that this must be a reference to the static version of the script data access method class. Access method call format in the child script are identical to those in the main script (i.e., `ccdd.methodName()`).
- In the main script add '`import sys`' and '`sys.path.append("path to the package folder")`', then '`import package folder name`'. The system path must include the package folder's location and the update must occur before importing the package.
- To call a function in a child script from the main script use the format '`package folder name.child function name()`'.

2.3 Ruby

Ruby script files must end with the extension "`.rb`". The Ruby script must contain the following line at or near the top of the file:

```
java_import Java::CCDD.CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a `begin-rescue` statement as shown below. The raise call output can be replaced or have other text appended if desired.

```
begin
    # Main script steps
    .
    .
rescue => err
    raise err.message + "; " + err.backtrace.join("; ")
```

```
end
```

The following is the example script described earlier in this section, written in Ruby:

```
# Import the script data access method class
java_import Java::CCDD.CcddScriptDataAccessHandler

# Open the output file
file = $ccdd.openOutputFile("myFileName")

# Get the array of structure names
structNames = $ccdd.getStructureTableNames()

index = 0

# Step through each structure name
structNames.each do |name|

    # Write the structure name to the output file
    $ccdd.writeToFileLn(file, "structNames[#{index}] = #{name}")

    index += 1

end

# Close the output file
$ccdd.closeFile(file)
```

2.4 Groovy

Groovy script files must end with the extension ".groovy". The Groovy script must contain the following line at or near the top of the file:

```
import CCDD.CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a try-catch statement as shown below. The throw call output can be replaced or have other text appended if desired.

```
try
{
    # Main script steps
    .
    .
}
catch (Exception err)
{
    throw new Exception(err.message + "; " + err.getStackTrace())
}
```

The following is the example script described earlier in this section, written in Groovy:

```
// Import the script data access method class
import CCDD.CcddScriptDataAccessHandler

// Open the output file
def file = ccdd.openOutputFile("myFileName")

// Get the array of structure names
def structNames = ccdd.getStructureTableNames()

// Step through each name found
for (def index = 0; index < structNames.length; index++)
{
    // Write the structure name to the output file
    ccdd.writeToFileLn(file,
                       "structNames[" +
                       index +
                       "] = " +
                       structNames[index])
}

// Close the output file
ccdd.closeFile(file)
```

2.5 Scala

Scala script files must end with the extension ".scala". The Scala script must contain the following line at or near the top of the file:

```
import CCDD.CcddScriptDataAccessHandlerStatic._
```

If enhanced error logging is desired then encompass the script code with a `try-catch` statement as shown below. The throw call output can be replaced or have other text appended if desired.

```
try
{
    # Main script steps
    .
    .
}
catch
{
    case err: Exception => throw new Exception(err.message + "; " +
              err.getStackTrace().mkString("; "))
}
```

The following is the example script described earlier in this section, written in Scala:

```
// Import the script data access method class
import CCDD.CcddScriptDataAccessHandlerStatic._

// Open the output file
var file = openOutputFile("myFileName")

// Get the array of structure names
var structNames = getStructureTableNames()

// Step through each name found
for (index <- 0 to structNames.length - 1)
{
    // Write the structure name to the output file
    writeToFileLn(file,
                  "structNames[" +
                  index +
                  "] = " +
                  structNames(index))
}

// Close the output file
closeFile(file)
```

3.0 Script Execution from Command Line

The CCDD command line option, execute, allows running scripts without use of the GUI. The script file and data table association must be specified on the command line. The command format is:

```
<script_name[:table[+table2[+...[+tableN]]][;...]]>
```

Groups can be used in place of, or along with tables. Each referenced group name must be preceded by 'Group:' in order to be recognized as a group. For example:

```
script_name:Group:group_name
```

The project database, host, user, and password (if required) command line options must be specified as part of the execute option in order to access the project's database. If not specified, the last project database, user, and host accessed by the application in the most recent session is used. The script name must include its file path if the script is not located within the folder from which the CCDD application is executed. If multiple scripts are provided in the same execute command then the individual associations must be separated by a semi-colon "`;`" and the entire string containing the associations for that "`execute`" command must be bounded by single or double quotes. Multiple "`execute`" commands in the same command line command can be used as well to execute multiple script associations; the format for each is as described above. If multiple script associations are specified then these are run serially in the order they appear in the command line command.

Even though the GUI is not displayed, the event log is generated and all events (success, fail, command, and status events) are written to the log file. Information, warning, and error dialogs are not displayed; instead the text for these dialogs is output to the standard output (information) and standard error (warning and error) streams. Dialogs within a script requiring user input, however, are displayed, and script execution pauses until the dialog is dealt with.

When script execution completes the CCDD application terminates. The application returns a status indicating if the scripts executed successfully: 0 if all script execution succeeded, or 1 if any script did not complete successfully.

Following are examples of running scripts from the command line. Note that in these examples, CCDD is an alias that executes the application with all the necessary class paths, etc.. The first example demonstrates executing the script `myScript` with no associated tables:

```
CCDD -project myProject -host localHost \
     -user myUser -password myPassword \
     -execute myScript
```

The next example executes `myScript` using the data from the table `myTable` (and its child tables, if applicable):

```
CCDD -project myProject -host 192.168.1.1 -port 5432 \
     -user myUser -password myPassword \
     -execute myScript:myTable
```

The third example executes `myScript` using the data from the tables `myTable1` and `myTable2` (and their child tables, if applicable):

```
CCDD -project myProject -user myUser -password myPassword \
     -execute myScript:myTable1+myTable2
```

The last example executes `myScript1` using the data from the tables `myTable1` and `myTable2` (and their child tables, if applicable), then executes the script `myScript2` using the data from the table `myTable3` (and its child tables, if applicable):

```
CCDD -password myPassword \
     -execute myScript1:myTable1+myTable2,myScript2:myTable3
```

4.0 Overridable XTCE Export Script Methods

The XTCE schema allows for some interpretation as to how data is parsed to the output and provides for many features not covered by the export methods within the CCDD application, so the content of the CCDD-generated XTCE files may not be as desired. A

script can be created to produce the conversion; however, there are certain operations performed in the internal code that would be difficult to replicate in a script - for example, the conversion of the command header structure table to the equivalent command metadata required by the XTCE schema. To make XTCE conversion via a script easier a hybrid of the internal code and script code can be used.

The internal code has "hooks" that allow for several of the key conversion methods to be replaced with code in a script. These hooks are activated when exporting via the XTCE export dialog by selecting the "**Use external methods**" check box, or when the script access method `xtceExport` is called from a script.

Appendix A provides a brief description of each of the XTCE export methods that can be overridden by a script function. The script must use the same function name as shown in the table, and the "**Input(s)**" and "**Output**" columns describe the parameters that are passed to and expected from the script functions. **Appendix A** also includes a flow chart of the methods' calling sequence. If the script doesn't have the function to replace a method then the internal method is used by default.

The user can elect to arrange the conversion process in a manner other than by the predefined methods. Assuming the `xtceExport()` script data access method or the export dialog is used as the conversion initiator, the `addSpaceSystemParameters` and `addSpaceSystemCommands` must be the entry points in the script.

5.0 Data Access Script Methods

Appendix B provides details on each of the project data access methods available for use in the scripts. The first column is the method name. The scripts are automatically assigned a variable, ccdd, which references the class containing the data access methods. When calling one of the access methods from a script the method name must be preceded by `ccdd` (or `ccdds` if using the static methods) for JavaScript, Python, and Groovy, and $ccdd for Ruby; for Scala only the method name is used. The second column is a short description of the access method. The third column in the table gives the method input parameter type(s) and description(s), if any. The fourth column gives the output type and description, if any. The last column indicates the applicability of the method to the project data. See a small table in **Appendix B** for definition of the applicability codes.

Certain methods require that the table type be supplied as a parameter. Convenience methods are provided in these cases for the Structure and Command table types. In place of supplying the table type as a parameter the method name incorporates the table type. For example, the method `getTableData` has accompanying convenience methods `getCommandTableData` and `getStructureTableData`.

6.0 Troubleshooting

You should not assume that CCDD.jar is fully self-contained: some versions of the jar are, and some are not. To be safe, always run CCDD from within its directory in the git repository, rather than copying just the jar file elsewhere. If you really feel the need to move the jar, then make sure the directory CCDD_lib moves with it.

7.0 Known Issues

1. Concurrent operation is not currently supported. Simultaneously interacting with the same project from more than one instance of the CCDD application or via another database access application can result in unexpected results or corruption of the project database.
2. If 32-bit Java 7 is used in a 64-bit Linux environment then the 32-bit compatibility libraries must be installed. The specific libraries are Linux version dependent. As an example, the user's guide cannot be displayed in 64-bit CentOS 6 using the command menu unless the Gnome 32-bit library, libgnome.i686, is installed.
3. In Java 9 and subsequent versions the JAXB libraries are no longer part of the default Java installation. For Java 9 and 10, in order for these libraries to be accessed the option --add-modules java.xml.bind must be added to the CCDD startup command. This 'fix' will no longer be valid beginning with Java 11.
4. When executing the application using the command line -shutdown option the GUI is hidden. However, the Java Swing classes used to generate the GUI must be available during program execution. An example is if the application is executed over a SSH connection – the -X flag must be specified.

Appendix A. Overridable XTCE Export Script Methods

Internal Method / Script Function Name	Description	Input(s)	Output
addCommand	Add a MetaCommand element to the MetaCommandSet. Each MetaCommand element has a BaseMetaCommand (if the command header table is defined), which uses ArgumentAssignment elements to set the application ID and command function code. The ArgumentList and CommandContainer are populated with the command argument information. The script data access method `xtceAddCommand` can be called instead to use the internal method.	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference String: Command header table name SpaceSystemType: Space system reference String: Command name String: Command code name String: Command code String: Name of the application ID String: Application ID boolean: true if this table represents the command header String: Command header table system path String[]: Array of command argument names String[]: Array of of command argument data types String[]: Array of of command argument array sizes; the array item is null or blank if the corresponding argument isn't an array String: Description of the command	

Internal Method / Script Function Name	Description	Input(s)	Output
addContainerReference	Generic function used by both the telemetry and command functions to add a container reference to the specified EntryList element. The script data access method `xtceAddContainerReference` can be called instead to use the internal method.	ObjectFactory: object factory reference EntryListType: Reference to the telemetry or command entry list into which to place the parameter or parameter array container reference(s) String: Parameter name String: Data type String: Parameter array size; null or blank if the parameter isn't an array	
addParameterAndType	Add a structure table variable, if it has a primitive data type, to the ParameterTypeSet and ParameterSet elements. The script data access method `xtceAddParameterAndType` can be called instead to use the internal method.	ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system reference String: Parameter name String: Parameter primitive data type String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or blank if not a bit-wise parameter	

Internal Method / Script Function Name	Description	Input(s)	Output
		String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units String: Minimum parameter value String: Maximum parameter value String: Parameter description int: Size, in characters, of a string parameter; ignored if not a string or character	
addParameterSequenceEntry	Add the structure table variables as EntryList entries in a SequenceContainer element within a ContainerSet element. The script data access method `xtceAddParameterSequenceEntry` can be called instead to use the internal method.	ObjectFactory: Object factory reference String: Telemetry header table name SpaceSystemType: Reference to the space system to which the parameter belongs String: Parameter name String: Data type String: Array size EntryListType: Reference to the entry list into which to place the parameter (for a primitive data type) or container (for a structure data type) reference boolean: true if this table represents the telemetry header or one of its descendants	boolean: true if the parameter's data type references the telemetry header or one of its descendants; otherwise return the flag status unchanged

Internal Method / Script Function Name	Description	Input(s)	Output
addSpaceSystemCommands	Overall command assignment handler. Steps through each row of data in the command table, parses the command's aruments, and calls functions that assign arguments to the ArgumentTypeSet element. For each command in the table a function is called to create the MetaCommand element, which describes the command, within the MetaCommandSet. The script data access method `xtceAddSpaceSystemCommands` can be called instead to use the internal method.	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Command header table name AssociatedColumns[]: Array of AssociatedColumns class instances that have the associated command argument column indices SpaceSystemType: Space system reference String[][]: Table data array int: Command name column index int: Command code column index int: Command description column index boolean: true if this table represents the command header String: Command header table system path String: Command code name String: Name of the application ID String: Application ID	

Internal Method / Script Function Name	Description	Input(s)	Output
addSpaceSystemHeader	Add the Header element to the space system. For the root space system the AuthorSet and NoteSet elements are automatically created and populated with the user, project, creation, and endianness information. The script data access method `xtceAddSpaceSystemHeader` can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space system reference String; Classification attribute String: Validation status attribute String: Version attribute String: Creation time and date	
addSpaceSystemParameters	Overall parameter assignment handler. Steps through each row of data in the structure table and calls functions that assign variables to the ParameterTypeSet, ParameterSet, and ContainerSet elements. If the application ID is provided, the table is a root structure, and it has a variable with a data type referencing the	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system to which the table belongs	

Internal Method / Script Function Name	Description	Input(s)	Output
	telemetry header table, then a BaseContainer element is created with a RestrictionCriteria element set to the application ID. The script data access method `xtceAddSpaceSystemParameters` can be called instead to use the internal method.	String: Table name String[][]: Array containing the table's data int: Variable (parameter) name column index int: Parameter data type column index int: Parameter array size column index int: Parameter bit length column index int: Parameter enumeration column index; -1 if no the table has no enumeration column int: Parameter description column index; -1 if no the table has no description column int: Parameter units column index; -1 if no the table has no units column int: Minimum parameter value column index; -1 if no the table has no minimum column int: Maximum parameter value column index; -1 if no the table has no maximum column boolean: true if this table represents the telemetry header or one of its descendants String: telemetry header table system path; null or blank is none boolean: true if the table is a root structure table String: Name of the telemetry header application ID data field	

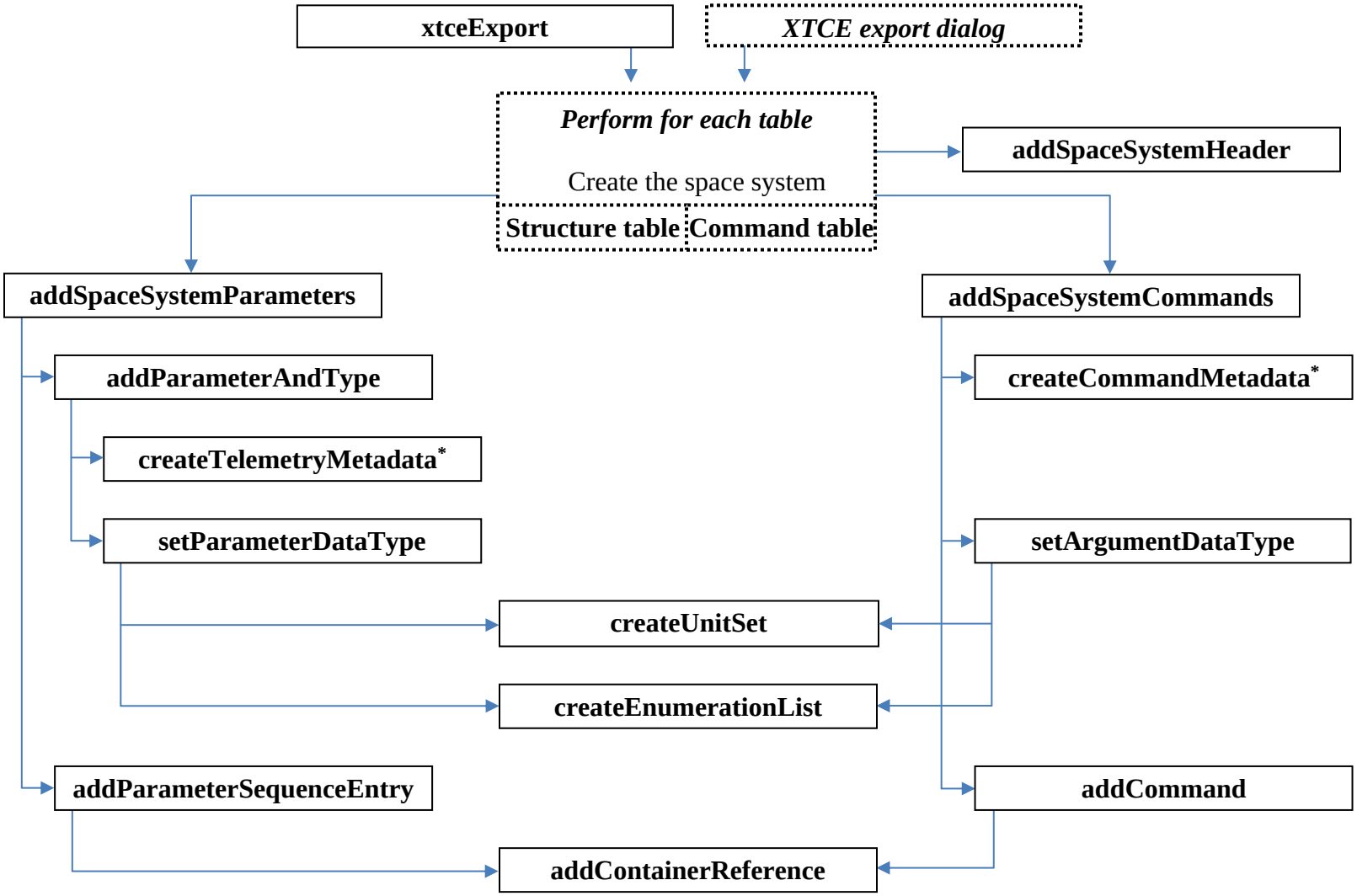
Internal Method / Script Function Name	Description	Input(s)	Output
		String: Telemetry header application ID	
createCommandMetadata	Create the CommandMetaData element within the specified space system. The script data access method `xtceCreateCommandMetadata` can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space reference	
createEnumerationList	Use the provided enumeration information to create an EnumerationList element. The script data access method `xtceCreateEnumerationList` can be called instead to use the internal method.	ObjectFactory: object factory reference SpaceSystemType: Space system reference String: Enumeration in the format <enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]	EnumerationList: Enumeration list for the supplied enumeration string
createTelemetryMetadata	Create the TelemetryMetaData element within the specified space system. The script data access method `xtceCreateTelemetryMetadata` can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space system reference	

Internal Method / Script Function Name	Description	Input(s)	Output
createUnitSet	Use the provided units information to create a UnitSet element. The script data access method `xtceCreateUnitSet` can be called instead to use the internal method.	ObjectFactory: object factory reference String: Parameter or command argument units; null to not specify	UnitSet: Unit set for the supplied units string; an empty unit set if no units are supplied
setArgumentDataType	Add a command table command argument as an argument type in the ArgumentTypeSet element. The argument type is based on the argument's data type. An array variable generates two entries, one to describe the array type, and a second to describe the data type of the array elements. The script data access method `xtceSetArgumentDataType` can be called instead to use the internal method.	boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Command header table name SpaceSystemType: Space system reference String: Command argument name; null to not specify String: Command argument data type; null to not specify String: Command argument array size; null or blank if the argument isn't an array String: Command argument bit length String: Command argument enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify	NameDescriptionType: Command description of the type corresponding to the primitive data type with the specified attributes set

Internal Method / Script Function Name	Description	Input(s)	Output
		String: Command argument units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Command argument description ; null to not specify int: String size in bytes; ignored if the command argument does not have a string data type	
setParameterDataType	Add a structure table variable as a parameter type in the ParameterTypeSet element. The parameter type is based on the variable's data type. An array variable generates two entries, one to describe the array type, and a second to describe the data type of the array elements. The script data access method `xtceSetParameterDataType` can be called instead to use the internal method.	ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system String: Parameter name; null to not specify String: Data type; null to not specify String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or empty if not a bit-wise parameter	

Internal Method / Script Function Name	Description	Input(s)	Output
		String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Parameter description; null to not specify int: Size, in characters, of a string parameter; ignored if not a string or character	

XTCE Export Script Method Flowchart



***** *Called once per space system, and only if a telemetry/command parameter exists in the table*

Appendix B. Data Access Script Methods

Code	Method Applicability
O	Method returns information with respect to only those tables associated with the script. This includes every method with a row number input.
A	Method returns information for any or all tables, not just those associated with the script.
S	Method returns information from the telemetry or application schedulers, so is not dependent on the associated tables.
N	Method is not table related, so is not dependent on the associated tables.

The codes above are used in the last column of the table for data access script methods below.

Method Name	Description	Input(s)	Output	*
closeFile	Close the specified output file	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method		N
formatArrayIndex	Convert an integer array containing the size of each array dimension into a string in the format `[#]<[#]<...>>`	int[]: Array of integers containing the size of each array dimension	String: Array size in the format `[#]<[#]<...>>`	N
getApplicationMessageDefinitionTable	Get the application scheduler message definition table		String[]: Array containing the message definition table information	S
getApplicationNames	Get the array containing the groups that represent CFS applications		String[]: Array containing names of the groups that represent CFS applications	A
getApplicationScheduleDefinitionTable	Get the specified entry in the application scheduler schedule definition table	int: Row index for the entry in the schedule definition table	String[][]: Array containing the specified entry in the schedule definition table	S
getApplicationScheduleDefinitionTableDefines	Get the array of defined parameters for the schedule definition table		String[]:Two-dimensional array containing the defined parameters	S
getArrayFromString	Divide the supplied string into an array using the supplied separator character or string, and trim any leading or trailing white space characters from each array member	String: String to separate into an array String: Character string to use to delineate the separation point(s) between columns. The separator is eliminated from the array members	String[]: Array representing the substrings in the supplied text after being parsed using the separator; returns null if the input text is empty	N

getArrayFromString	Divide the supplied string into a two-dimensional array (columns and rows) using the supplied separator characters or strings, and trim any leading or trailing white space characters from each array member	String: String to separate into an array String: Character string to use to delineate the separation point(s) between columns. The separator is eliminated from the array members String: Character string to use to delineate the separation point(s) between rows. The separator is eliminated from the array members. Use null if only one row is supplied	String[][]: Two-dimensional array representing the substrings in the supplied text after being parsed using the separator; returns null if the input text is empty	N
getArrayIndexFromSize	Get the integer array containing the size of each array dimension from the supplied array size string	String: Array size in the format `[#]<[#]<...>>` or `#<,#<...>>`	int[]: Array of integers containing the size of each array dimension	N
getAssociatedGroupNames	Get the array of group names referenced in the script association		String[]: Array containing the group names referenced in the script association; empty array if no groups are referenced	O
getBaseDataType	Get the base type for the specified data type	String: Name of the primitive data type	String: Base type for the specified data type; returns null if the data type doesn't exist or isn't a primitive type	N
getCDataType	Get the C type for the specified data type	String: Name of the primitive data type	String: C type for the specified data type; returns null if the data type doesn't exist or isn't a primitive type	N

getCheckBoxDialog	Display a dialog containing one or more check boxes. The user must press the Okay button to accept the check box input(s), or Cancel to close the dialog without accepting the input	String: Text to display above the check box(es) String[][]: Array containing the text and optional descriptions for the radio buttons to display in the dialog	boolean[]: An array containing the status for the check box(es) if the Okay button is pressed; returns null if no check box information is supplied or if the Cancel button is pressed	N
getCommandCode	Get the command code (as a string) at the specified row in the command data	int: Table data row index	String: Command code (as a string) at the specified row in the command data; null if the row index is invalid	O
getCommandInformation	Get an array containing the name, code, argument variable name(s), and command table for every command in the project database		String[][]: Array containing the name, code, argument variable name(s), and command table for every command. The array is sorted by command name; if the same then by command code; if the same then by table name	A
getCommandName	Get the command name at the specified row in the command data	int: Table data row index	String: Command name at the specified row in the command data; null if the row index is invalid	O
getCommandTableColumnNames	Get the column names for the table referenced on the specified row of the command table data	int: Command table data row index	String[]: Array containing the names of the columns of the command table referenced in the specified row of the command table data	O

getCommandTableData	Get the command table data at the row and column indicated, with any macro replaced by its corresponding value. The column is specified by name and is not case sensitive. Convenience method for getTableData that assumes the table type is "command"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified command table's array at the row and column name provided, with any macro replaced by its corresponding value; returns null if an instance of the command table type doesn't exist	O
getCommandTableDataFieldValues	Get the data field value for all command tables that have the specified data field	String: Data field name	String: Array of command table names and the data field value; returns an empty array if the field name is invalid (i.e., no command table has the data field)	O
getCommandTableDataWithMacros	Get the command table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive. Convenience method for getTableDataWithMacros that assumes the table type is "command"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified command table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the command table type doesn't exist	O
getCommandTableNameByRow	Get the command table name to which the specified row's data belongs. Convenience method for getTableNameByRow that assumes the table type is "command"	int: Table data row index	String: Command table name to which the current row's parameter belongs; returns a blank if an instance of the command table type or the row doesn't exist	O

getCommandTableNames	Get the array of all command table names in the table data. Convenience method for getTableNames that specifies the table type as "command"		String[]: Array of all command table names; returns an empty array if an instance of the command table type doesn't exist	O
getCommandTableNumRows	Get the number of rows of data in the command table. Convenience method for getTableNumRows that assumes the table type is "command"		int: Number of rows of data in the table for the table type "command"; return -1 if an instance of the command table type doesn't exist	O
getCommandTableRowIndices	Get an array of row numbers in the command table data that belong to the specified command table. Convenience method that assumes the table type is "Command"	String: Table name	Integer[]: Array ofthe command table data row numbers that belong to the specified command table; returns an empty array if the coammnd table name doesn't exist	O
getCommandTypeNameByRow	Get the table type name referenced in the specified row of the command table type data. Convenience method for getTypeNameByRow that specifies the table type as "command". The data for all command types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	int: Table data row index	String: Command table type name to which the current row's parameter belongs; returns a blank if an instance of the command table type or the row doesn't exist	O
getCopyTableColumnNames	Get the copy table column names		String[]: Array containing the copy table column names	S

getCopyTableEntries	Get the copy table for the messages of the specified data stream	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String: Name of the message ID name data field (e.g., 'Message ID name') boolean: true to combine memory copy calls for consecutive variables in the copy table boolean: false to retain any macros in the variable names; true to replace any macros with their corresponding values	String[][]: Array containing the copy table entries; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getCopyTableEntries	Get the copy table for the messages of the specified data stream	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String[][]: Array containing string array entries giving the structure table path+name and the table's associated message ID name boolean: true to combine memory copy calls for consecutive variables in the copy table boolean: false to retain any macros in the variable names; true to replace any macros with their corresponding values	String[][]: Array containing the copy table entries; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S

getCopyTableEntries WithMacros	Get the copy table for the messages of the specified data stream. Any macro embedded in a variable name is left in place	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String: Name of the message ID name data field (e.g., 'Message ID name') boolean: true to combine memory copy calls for consecutive variables in the copy table	String[][]: Array containing the copy table entries with any macro embedded in a variable name left in place; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getCopyTableEntries WithMacros	Get the copy table for the messages of the specified data stream. Any macro embedded in a variable name is left in place	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String[][]: Array containing string array entries giving the structure table path+name and the table's associated message ID name boolean: true to combine memory copy calls for consecutive variables in the copy table	String[][]: Array containing the copy table entries with any macro embedded in a variable name left in place; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getDatabaseQuery	Perform a query on the currently open database	String: PostgreSQL-compatible database query statement	String[][]: Two-dimensional array representing the rows and columns of data returned by the database query; returns null if the query produces an error, or an empty array if there are no results	N

getDataStreamNames	Get a string array containing all of the data stream names in the project		String[]: Array containing the unique data stream names	N
getDataTypeDefinitions	Get the array containing the user-defined data type names and their corresponding C-language, size (in bytes), and base data type values		String[][]: Array where each row contains a user-defined data type name and its corresponding C-language, size (in bytes), and base data type values	N
getDataTypeSizeInBits	Get the number of bits for the specified data type	String: Name of the structure or primitive data type	int: Number of bits required to store the data type; returns 0 if the data type doesn't exist	N
getDataTypeSizeInBytes	Get the number of bytes for the specified data type	String: Name of the structure or primitive data type	int: Number of bytes required to store the data type; returns 0 if the data type doesn't exist	N

getDateAndTime	Get the current time and date in the form: *dow mon dd hh*:*mm*:*ss zzz yyyy* where: *dow* is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat) *mon* is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) *dd* is the day of the month (01 through 31), as two decimal digits *hh* is the hour of the day (00 through 23), as two decimal digits *mm* is the minute within the hour (00 through 59), as two decimal digits *ss* is the second within the minute (00 through 61, as two decimal digits *zzz* is the time zone (and may reflect daylight saving time) *yyyy* is the year, as four decimal digits		String: Current date and time	N
getEnumTableNames	Retrieve the names of tables that have a type of Enum		String[]: Table names	O
getEnumTableData	Retrieve the data associated with the supplied enum table name	String: Table Name	String[]: Table data	O

getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by underscores, and with the data types removed	int: Table data row index	String: The variable's full path and name with each variable in the path separated by an underscore, and with the data types removed; returns a blank if the row is invalid	O
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the supplied separator character(s)	int: Table data row index String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s), and with the data types removed; returns a blank if the row is invalid	O
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the supplied separator character(s). Data types may be excluded or retained, based on the input flag. If retained, the data types and variable names are separated by the supplied separator character(s)	int: Table data row index String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the row is invalid	O

getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s) and with the data types removed. In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2)	String: Path to the variable in the format `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[,...]]]` String: Name of the variable in the format `primitiveDataType.variable` String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s) and with the data types removed; returns a blank if the variable path + name doesn't exist in the project database	A
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s). In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2). Data types may be excluded or retained, based on the input flag	String: Path to the variable in the format `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[,...]]]` String: Name of the variable in the format `primitiveDataType.variable` String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the variable path + name doesn't exist in the project database	A

getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s), and with the data types removed. In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2)	String: Variable path + name in the format `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[,...]]],primitiveDataType.variable` String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s), and with the data types removed; returns a blank if the variable path + name doesn't exist in the project database	A
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s). In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2). Data types may be excluded or retained, based on the input flag. Any macro embedded in the variable name is expanded.	String: Variable path + name in the format `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[,...]]],primitiveDataType.variable` String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the variable path + name doesn't exist in the project database. Any macro embedded in the variable name is expanded	A

getFullVariableNameRaw	Get the full name of the variable in the specified row of the structure datain the application's native format, which includes the variables in the structure path separated by commas, and with the data type and variable names separated by periods.	int: Table data row index	String: The variable's full path and name with each variable in the path separated by a comma, and with each data type and variable name separated by a period; returns a blank if the row is invalid	O
getGroupDataFieldDescription	Get the description of the data field for the specified group's specified data field.	String: Group name String: Data field name	String: Data field's description; returns a blank if the group name or data field name is invalid	N
getGroupDataFieldNames	Get the name(s) of the data field(s) associated with the specified group	String: Name of the group to which the field is a member	String: Array of the data field names associated with the specified group; returns an empty array if the group name is invalid or the group has no data fields	N
getGroupDataFieldValue	Get the contents of the data field for the specified group's specified data field	String: Name of the group for which the field is a member String: Data field name	String: Data field value; returns a null if the group name or field name is invalid	N
getGroupDescription	Get the description for the specified group	String: Group name	String: Description for the specified group; blank if the group has no description or the group doesn't exist	N

getGroupFields	Get the data field information for the specified group	String: Group name	String[][]: Array containing the data field information for the specified group; an empty array if the group has no data fields, or the group doesn't exist. Each row in the array describes a single data field in the format: field name, description, size, input type, required (true or false), applicability, value	N
getGroupNames	Get an array of all group names	boolean: true to get only the groups that represent a CFS application; false to get all groups	String[]: Array containing the group names (application groups only if the input flag is true); returns an empty array if no groups exist	N
getGroupTables	Get an array containing the table members, including the member table ancestor tables, for the specified group	String: Group name	String[]: Array containing the table members for the specified group; an empty array if the group has no table members, or the group doesn't exist	A
getInputDialog	Display a dialog for receiving text input. The user must select `Okay` to accept the input, or `Cancel` to close the dialog without accepting the input	String: Text label to display beside the input text field	String: The text entered in the dialog input field if the `Okay` button is pressed; returns null if no text or white space is entered, or if the `Cancel` button is pressed	N

getITOSEncodedDataType	Convert a primitive data type into its ITOS encoded form	String: Name of the data type; e.g., uint16, double String: ITOS encoding type: **SINGLE_CHAR** to get the single character encoding (e.g., "I" for any integer type) **TWO_CHAR** to get the encoding character with the data type size (e.g., "I4" for a 4-byte integer) **BIG_ENDIAN** to get the encoding as big endian **BIG_ENDIAN_SWAP** to get the encoding as a big endian with byte swapping **LITTLE_ENDIAN** to get the encoding as little endian **LITTLE_ENDIAN_SWAP** to get the encoding as a little endian with byte swapping	String: ITOS encoded form of the data type in the format requested; returns the data type, unmodified, if the data type is a table (i.e., it's a structure), or null if the data type is unrecognized. Example: a data type of "int32" and ITOS encoding type of **LITTLE_ENDIAN** returns "I12345678"	N
getITOSLimitName	Get the ITOS limit name based on the supplied index value	int: 0 = redLow, 1 = yellowLow, 2 = yellowHigh, 3 = redHigh	String: ITOS limit name ("redLow", "yellowLow", "yellowHigh", or "redHigh"); returns blank if the index is invalid	N

getLinkApplicationNames	Get the array containing the application name data field values associated with the specified link's variable members. Each application name is listed only once in the array	String: Name of the application name data field	String[]: Array containing the contents of the specified application name data field associated with each of the tables referenced by the link's variable members	N
getLinkDescription	Return the description for the specified link; returns a blank if the link doesn't exist or the link has no description	String: Data stream name String: Link name	String: Link description; returns a blank if the data stream or link don't exist, or the link has no description	N
getLinkRate	Return the sample rate for the specified link; returns a blank if the link doesn't exist	String: Data stream name String: Link name	String: Text representation of the sample rate, in samples per second, of the specified link. For rates equal to or faster than 1 sample per second the string represents a whole number; for rates slower than 1 sample per second the string is in the form `number of samples / number of seconds`; returns a blank if the data stream or link don't exist	N
getLongestString	Get the character length of the longest string in the supplied string array	String[]: Array of strings Integer: Initial minimum width; null to use zero as the minimum	int: Character length of the longest string in the supplied array; null if an input is invalid	N

getLongestStrings	Get the character length of the longest string for each column in the supplied string array	String[][]: Array of string arrays Integer[]: Initial minimum widths; null to use zero as the minimums	Integer[]:Character length of the longest string in each column of the supplied array; null if any of the inputs is invalid	N
getMacroDefinitions	Get the array containing the macro names and their corresponding values		String[][]: Array where each row contains a macro names and its corresponding value	N
getMessageOwnersNamesAndIDs	Get an array containing every message owner, name, and ID from every table cell, data field (table or group), and telemetry message. Message names and IDs are determined by the input type assigned to the table column or data field		String[][]: Two-dimensional array containing every message owner, name, and ID, sorted by owner name. Each row in the array is an array in the form [owner name], [message name], [message ID]. The owner name is preceded by 'Group:' if the owner is a group, and by "Tlm:' if the owner is a telemetry message	A
getNumberOfTimeSlots	Get the number of time slots for the scheduler definition table		int: Number of time slots for the scheduler definition table table	N
getNumCommandArguments	Get the number of arguments associated with the command table type at the specified row in the command data	int: Row index	int: Number of arguments associated with the command table type at the specified row in the command data	O

getNumCommandArguments	Get the number of arguments associated with the specified command table type	String: Table type (case insensitive)	int: Number of arguments associated with the specified command table type; -1 if the table type is invalid	O
getOutputPath	Get the script output file path set via the program preferences dialog or command line		String: Script output file path; blank if no path has been set	N

getPathByRow	Get the path to which the specified row's data belongs with any embedded macro replaced by its corresponding value	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: The path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]`	O

getPathByRowWithMacros	Get the path to which the specified row's data belongs with any embedded macro(s) left in place	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: The path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]`	O
getProject	Get the project's name		String: Name of the project	N
getProjectDataFieldDescription	Get the description of the data field for the specified project data field	String: Data field name	String: Data field's description; returns a blank if the project data field name is invalid	N

getProjectDataFieldNames	Get the name(s) of the data field(s) associated with the project		String: Array of the data field names associated with the project; returns an empty array if the project has no data fields	N
getProjectDataFieldValue	Get the value for the specified project data field	String: Data field name	String: Data field value; returns a null if the project field name is invalid	N
getProjectDescription	Get the project's description		String: Description of the project	N
getProjectFields	Get the data field information for the project		String[][]: Array containing the data field information for the project; an empty array if the project has no data fields. Each row in the array describes a single data field in the format: field name, description, size, input type, required (true or false), applicability, value	N
getPrototypeName	Get the name of the prototype table for the specified table	String: Table name	String: The name of the prototype table for the specified table	N

getRadioButtonDialog	Display a dialog containing radio buttons. The radio buttons are mutually exclusive; only one can be selected at a time. The user must press the Okay button to accept the radio button input, or Cancel to close the dialog without accepting the input	String: Text to display above the radio buttons String[][]: Array containing the text and optional descriptions for the radio buttons to display in the dialog	String: The text for the selected radio button if the Okay button is pressed; returns null if no radio button is selected or if the Cancel button is pressed	N
getRootStructureTableNames	Get the name(s) of the root structure table(s). Convenience method for getRootTableNames that assumes the table type is "structure"		String[]: Array containing the root structure table names; returns an empty array if an instance of the structure table type doesn't exist	O
getRootTableNames	Get the name(s) of the root table(s) for the supplied table type. Note that only structure tables can have child tables so using this method for non-structure tables returns the same list of tables as getTableNames(typeName)	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	String[]: Array containing the root table names for the type specified; returns an empty array if an instance of the table type doesn't exist	O
getScriptName	Get the name of the script file being executed		String: Script file name	N
getStructureArraySize	Get the variable array size at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable array size at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O

getStructureArraySizeWithMacros	Get the variable array size at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable array size at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureBitLength	Get the variable bit length at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable bit length at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureBitLengthWithMacros	Get the variable bit length at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable bit length at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureDataByVariableName	Get the data from the specified "Structure" table in the specified column for the row with the specified variable name, with any macro replaced by its corresponding value. Convenience method for getTableDataByColumnName that assumes the table type is "Structure" and the variable name column is "Variable Name"	String: Full table path, which includes the parent table name and the data type + variable name pairs String: Variable name String: Column name (case insensitive)	String: Contents of the table defined by the table path, variable name, and column name specified; returns null if an instance of the table type, the column name, or the variable name doesn't exist	O

getStructureDataByVariableNameWithMacros	Get the data from the specified "Structure" table in the specified column for the row with the specified variable name, with any macro name(s) left in place. Convenience method getTableDataByColumnName that assumes the table type is "Structure" and the variable name column is "Variable Name"	String: Full table path, which includes the parent table name and the data type + variable name pairs String: Variable name String: Column name (case insensitive)	String: Contents of the table defined by the table path, variable name, and column name specified, with any macro name(s) left in place; returns null if an instance of the table type, the column name, or the variable name doesn't exist	O
getStructureDataType	Get the variable data type at the specified row in the structure data	int: Table data row index	String: Variable data type at the specified row in the structure data; null if the row index is invalid	O
getStructureDescription	Get the variable description at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable description at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid or no column has the 'Units' input type	O
getStructureDescriptionWithMacros	Get the variable description at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable description at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid or no column has the 'Units' input type	O

getStructureEnumerations	Get the variable enumeration(s) at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Array containing the variable enumeration(s) at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureEnumerationsWithMacros	Get the variable enumeration(s) at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Array containing the variable enumeration(s) at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureParentRowByChildRow	Get the row index in the structure data for the first entry associated with the parent structure of the entry on the specified row of the structure data. The subsequent rows of the parent structure are not necessarily contiguous in the structure data. If a variable in the structure has a structure data type then the child structure's rows are inserted within the rows of the parent. Use getStructurePathByRow() (or variant) to determine the structure to which a specific row belongs	int: Table data row index	int: The row index in the structure data for the first entry associated with the parent structure of the entry on the specified row of the structure data ; -1 if there is no parent associated with the specified row or no structure data exists	O

getStructurePathByRow	Get the structure path to which the specified row's data belongs with any embedded macro replaced by its corresponding value. Convenience method that assumes the table type is "structure"	int: Table data row index	String: The structure path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]`	O

getStructurePathByRowWithMacros	Get the structure path to which the specified row's data belongs with any embedded macro(s) left in place. Convenience method that assumes the table type is "structure"	int: Table data row index	String: The structure path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: `rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]`	O
getStructureRates	Get the variable rate(s) at the specified row in the structure data	int: Table data row index	String: Array containing the variable rate(s) at the specified row in the structure data; null if the row index is invalid	O

getStructureTableColumnNames	Get the column names for the table referenced on the specified row of the structure table data	int: Structure table data row index	String[]: Array containing the names of the columns of the structure table referenced in the specified row of the structure table data	O
getStructureTableData	Get the structure table data at the row and column indicated, with any macro replaced by its corresponding value. The column is specified by name and is not case sensitive. Convenience method for getTableData that assumes the table type is "structure"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified structure table's array at the row and column name provided, with any macro replaced by its corresponding value; returns null if an instance of the structure table type doesn't exist	O
getStructureTableDataFieldValues	Get the data field value for all structure tables that have the specified data field	String: Data field name	String: Array of structure table names and the data field value; returns an empty array if the field name is invalid (i.e., no structure table has the data field)	O
getStructureTableDataWithMacros	Get the structure table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive. Convenience method for getTableDataWithMacros that assumes the table type is "structure"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified structure table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the structure table type doesn't exist	O

getStructureTableITOSPathByRow	Get the structure path to which the specified row's data belongs, formatted for use in an ITOS record statement, and with any macro name replaced by its corresponding value	int: Table data row index	String: The path to the current row's parameter formatted for use in an ITOS record statement and with any macro name replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a period and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by an underscore. The format is: `rootTable[.variable1[.variable2[...]]]`	O

getStructureTableITOSPathByRowWithMacros	Get the structure path to which the specified row's data belongs, formatted for use in an ITOS record statement, and with any macro name(s) left in place	int: Table data row index	String: The path to the current row's parameter formatted for use in an ITOS record statement and with any macro name(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a period and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by an underscore. The format is: `rootTable[.variable1[.variable2[...]]]`	O
getStructureTableNameByRow	Get the prototype structure table name to which the specified row's data belongs. Convenience method for getTableNameByRow that assumes the table type is "structure"	int: Table data row index	String: Prototype structure table name to which the current row's parameter belongs; returns a blank if an instance of the structure table type or the row doesn't exist	O
getStructureTableNames	Get array of all prototype structure table names referenced in the table data. Convenience method that specifies the table type as "structure"		String[]: Array of all prototype structure table names; returns an empty array if an instance of the structure table type doesn't exist	O

getStructureTableNumRows	Get the number of rows of data in the structure table. Convenience method for getTableNumRows that assumes the table type is "structure"		int: Number of rows of data in the table for the table type "structure"; return -1 if an instance of the structure table type doesn't exist	O
getStructureTablePaths	Get array of all structure table names, including paths for child structure tables, referenced in the table data. Convenience method that specifies the table type as "structure"		String[]: Array of all structure table names, including paths for child structure tables; returns an empty array if an instance of the structure table type doesn't exist	O
getStructureTableRowIndices	Get an array of row numbers in the structure table data that belong to the specified structure table. Convenience method that assumes the table type is "Structure"	String: Full table path	Integer[]: Array ofthe structure table data row numbers that belong to the specified structure table; returns an empty array if the structure table path doesn't exist	O
getStructureTablesByReferenceOrder	Get an array containing the names of the prototype structures in the order in which they are referenced; that is, the structure array is arranged so that a structure appears in the array prior to a structure that references it		String[]: Array containing the names of the prototype structures in the order in which they are referenced	O

getStructureTableVariablePathByRow	Get the structure path to which the specified row's data belongs, showing only the root structure and variable names and with any embedded macro replaced by its corresponding value. This format is used when referencing a structure table's data fields	int: Table data row index	String: The path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a comma and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by a comma. The format is: `rootTable[,variable1[,variable2[...]]]`	O

getStructureTableVariablePathByRowWithMacros	Get the structure path to which the specified row's data belongs and with any embedded macro(s) left in place, showing only the root structure and variable names. This format is used when referencing a structure table's data fields	int: Table data row index	String: The path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a comma and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by a comma. The format is: `rootTable[,variable1[,variable2[...]]]`	O
getStructureTypeNameByRow	Get the table type name referenced in the specified row of the structure table type data. Convenience method for or getTypeNameByRow that specifies the table type as "structure". The data for all structure types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	int: Table data row index	String: Structure table type name to which the current row's parameter belongs; returns a blank if an instance of the structure table type or the row doesn't exist	O

getStructureUnits	Get the variable units at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable units at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid or no column has the 'Description' input type	O
getStructureUnitsWithMacros	Get the variable units at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable units at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid or no column has the 'Description' input type	O
getStructureVariableName	Get the variable name at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable name at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureVariableNameWithMacros	Get the variable name at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable name at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O

getTableColumnNames	Get the table column names for the table referenced on the specified row of the table data for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String[]: Array containing the names of the columns of the table type referenced in the specified row of the type's table data	O
getTableColumnNamesByType	Get the table column names for the table type specified	String: Table type name. This is the table's actual type name and not the generic 'Structure' or 'Command' used to access combined structure or command table data	String[]: Array containing the names of the columns of the table type specified	A
getTableData	Get the data at the row and column indicated, with any macro replaced by its corresponding value, for the table type specified. The column is specified by name	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified table's array at the row and column name provided with any macro replaced by its corresponding value; returns null if an instance of the table type, the column name, or the row doesn't exist	O

getTableDataByColumnName	Get the data from the a table in the specified column for the row in the matching column name that contains the matching name, with any macro name replaced by its corresponding value	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Full table path String: Name of the column containing that matching name (case insensitive) String: Text to match in the matching column - this determines the row. The first row in the matching column that matches the matching name determines the row used to retrieve the data value String: Name of the column from which to retrieve the data value (case insensitive)	String: Contents of the table defined by the table type, table path, matching column name, matching name, and data column name specified, with any macro name replaced by its corresponding value; returns null if an instance of the table type, the matching column, the data column, or the matching name doesn't exist	O

getTableDataByColumnNameWithMacros	Get the data from the a table in the specified column for the row in the matching column name that contains the matching name, with any macro name(s) left in place	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Full table path String: Name of the column containing that matching name (case insensitive) String: Text to match in the matching column - this determines the row. The first row in the matching column that matches the matching name determines the row used to retrieve the data value String: Name of the column from which to retrieve the data value (case insensitive)	String: Contents of the table defined by the table type, table path, matching column name, matching name, and data column name specified, with any macro name(s) left in place; returns null if an instance of the table type, the matching column, the data column, or the matching name doesn't exist	O
getTableDataFieldDescription	Get the description of the data field for the specified table's specified data field	String: Table name, including the path if this table references a structure String: Data field name	String: Data field's description; returns a blank if the table name or data field name is invalid	A

getTableDataFieldNames	Get the name(s) of the data field(s) associated with the specified table	String: Name of the table, including the path if this table references a structure, to which the field is a member	String: Array of the data field names associated with the specified table; returns an empty array if the table name is invalid or the table has no data fields	A
getTableDataFieldValue	Get the value for the specified table's specified data field	String: Name of the table, including the path if this table references a structure, for which the field is a member String: Data field name	String: Data field value; returns a null if the table name or field name is invalid	A
getTableDataFieldValues	Get the data field value for all tables that have the specified data field	String: Data field name	String: Array of table names and the data field value; returns an empty array if the field name is invalid (i.e., no table has the data field)	O
getTableDataFieldValues	Get the data field value for all tables of the specified type that have the specified data field	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command". null to include tables of any type String: Data field name	String: Array of table names of the specified type and the data field value; returns an empty array if the field name is invalid (i.e., no table has the data field)	O

getTableDataWithMacros	Get the table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the table type, the column name, or the row doesn't exist	O
getTableDescription	Get the description of the specified table	String: Table name, including the full path for child structure tables	String: Description of the specified table; returns a blank the table doesn't exist	A
getTableDescriptionByRow	Get the description of the table at the row indicated for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Table name for the specified table type to which the current row's parameter belongs; returns a blank if an instance of the table type or the row doesn't exist	O
getTableDataByName	Get the table data related to the specified table	String: Table name	String[][]: 2D String array containing each row of data associated with the specified table. If no data is entered into a specific column the value will be set to null	A

getTableDataByNameAndColumn	Grab the data in every row of the specified column associated with the specified table.	String: Table name String: Column name Boolean: Expand macros	String[]: A string array containing the data in all rows of the specified column. If Expand macros is set to TRUE than all macros will be expanded to their actual value.	A
getTableFieldsByName	Grab all fields, or just the name and value of each field, associated with the specified table.	String: Table name Boolean: Only name and value? If this is set to 'true' then only the name and value of each field will be returned. If it is 'false' then all data related to each field will be returned.	String[][]: 2D array containing all data related to each field or just the name and value of each field.	A
getTableNameByRow	Get the prototype table name for the type specified to which the specified row's parameter belongs	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Prototype table name to which the current row's parameter belongs; return a blank if an instance of the table type or the row doesn't exist	O
getTableNames	Get array of all table names, including paths for child structure tables, referenced in the table data for all table types		String[]: Array of all table names, including paths for child structure tables, referenced in the table data; empty array if no tables exists in the data	O

getTableNames	Get array of all table names, including paths for child structure tables, referenced in the table data of the specified table type	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	String[]: Array of all table names, including paths for child structure tables, represented by the table type (prototype names for child structures); returns an empty array if an instance of the table type doesn't exist	O
getTableNames	Get array of all table names referenced in the table data of the specified table type	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" boolean: true to return only the prototype name for any child structures; false to include the full path for child structures	String[]: Array of all table names, with paths for child structure tables excluded based on the input flag, represented by the table type; returns an empty array if an instance of the table type doesn't exist	O
getTableNumRows	Get the number of rows of data for all table types		int: Number of rows of data for all table types; return 0 if there is no table data	O
getTableNumRows	Get the number of rows of for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	int: Number of rows of data in the table for the table type specified; return -1 if an instance of the table type doesn't exist	O

getTableRowIndices	Get an array of row numbers in the table data for the specified table type that belong to the specified table	String: table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Full table path	Integer[]: Array ofthe structure table data row numbers that belong to the specified structure table; returns an empty array if the structure table path doesn't exist	O
getTelemetryMessageIDs	Get the copy table message names and their corresponding ID values for the specified data stream	String: Data stream name	String: Array containing the copy table message names and ID values; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getTypeDataFieldDescription	Get the description of the data field for the specified table type's specified data field	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Data field name	String: Data field's description; returns a blank if the table type name or data field name is invalid	A
getTypeDataFieldNames	Get the name(s) of the data field(s) associated with the specified table type	String: Name of the table type to which the field is a member	String: Array of the data field names associated with the specified table type; returns an empty array if the table type name is invalid or the table type has no data fields	A

getTypeDataFieldValue	Get the value for the specified table type's specified data field	String: Name of the table type for which the field is a member String: Data field name	String: Data field value; returns a null if the table type name or field name is invalid	A
getTypeNameByRow	Get the table type name referenced in the specified row of the specified table type data. Multiple structure (and command) types are allowed. The data for all structure (command) types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Type name referenced in the specified row of the specified table type data. This the table's actual type name and not the generic 'Structure' or 'Command' used to access combined structure or command table data. Returns a blank if the table type name or row is invalid	O
getTypeNameByTable	Get the the table type name for the specified table	String: Name of the table. For a child structure this includes the path	String: Type name for the specified table. This the table's actual type name and not the generic 'Structure' or 'Command' used to access combined structure or command table data	O
getUser	Get the name of the user executing the script		String: Name of the user executing the script	N
getVariableLinks	Get the array of link names to which the specified variable belongs	String: Variable path and name	String[]: Array containing the links to which the specified variable is a member; returns an empty array if the variable does not belong to a link	N

getVariableOffset	Get the byte offset of the specified variable relative to its parent structure. The variable's path, including parent structure and variable name, is used to verify that the specified target has been located; i.e., not another variable with the same name	String: Parent structure name of the variable being checked String: A comma separated string of each data type and variable name of each variable in the current search path	int: The byte offset to the target variable relative to its parent structure; returns -1 if the parent-variable path combination is invalid	A
getVariablePaths	Get an array containing the path to each parent structure and its variables		String[][]: Array containing the path for each structure variable. The root structures are sorted alphabetically. The variables are displayed in the order of appearance within the structure (parent or child). Any macro is replaced by its corresponding value	A
isArrayMember	Check if the supplied variable name represents an array member	Object: Variable name	boolean: true if the variable name is an array member	N
isDataTypeCharacter	Determine if the supplied data type is a character or string	String: Name of the data type	boolean: true if the supplied data type is a character or string; false otherwise	N
isDataTypeFloat	Determine if the supplied data type is a float or double	String: Name of the data type	boolean: true if the supplied data type is a float or double; false otherwise	N
isDataTypeInteger	Determine if the supplied data type is a signed or unsigned integer	String: Name of the data type	boolean: true if the supplied data type is a signed or unsigned integer; false otherwise	N

isDataTypePrimitive	Determine if the supplied data type is a primitive type	String: Name of the data type	boolean: true if the supplied data type is a primitive; false otherwise	N
isDataTypeString	Determine if the supplied data type is a character string	String: Name of the data type	boolean: true if the supplied data type is a character string; false otherwise	N
isDataTypeUnsignedInt	Determine if the supplied data type is an unsigned integer	String: Name of the data type	boolean: true if the supplied data type is an unsigned integer; false otherwise	N
isGUIHidden	Check if the graphical user interface is not displayed		boolean: true if the GUI is hidden; false if the GUI is visible	N
isStructureShared	Determine if the specified structure is referenced by more than one root structure	String: Prototype name of the structure to check	boolean: true if the specified structure is referenced by more than one root structure; false otherwise	N
isStructureSharedExternally	Determine if the specified structure is referenced by more than one root structure, and that at least one of the structures is not associated with the script	String: Prototype name of the structure to check	boolean: true if the specified structure is referenced by more than one root structure and at least one of these structures is not associated with the script; false otherwise	N
openOutputFile	Open the specified file for writing. The PrintWriter object that is returned is used by the file writing methods to specify the output file	String: Output file path + name	PrintWriter: PrintWriter object; returns null if the file could not be opened	N

parseEnumerationParameters	Divide the supplied enumeration string into the values and labels. The enumeration value/label separator character and the enumerated pair separator character are automatically determined. Any leading or trailing white space characters are removed from each array member	String: Enumeration in the format <enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]	String[][]: Two-dimensional array representing the enumeration parameters ; returns null if the input text is empty or the enumeration separator characters cannot be determined	N
parseMessageNameAndID	Parse the supplied string containing a message name and ID into an array with the name in index 0 and the ID in index 1 (depending on the input string either or both may be blank). If only the name or ID is present in the supplied string the output is based on if the string evaluates to a hexadecimal value (treated as the ID; name is blank) or not (treated as the name; ID is blank)	String: Message name and ID in the format [<message name>] [<message ID>]	String[]:One-dimensional array containing the message name in index 0 and the ID in index 1 (depending on the input string either or both may be blank)	N
showErrorDialog	Display an error dialog showing the supplied text. The dialog's header and icon indicate that the text describes an error condition. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N

showInformationDialog	Display an informational dialog showing the supplied text. The dialog's header and icon indicate that the text describes information useful to the user; e.g., script status. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N
showWarningDialog	Display a warning dialog showing the supplied text. The dialog's header and icon indicate that the text describes a warning condition. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N
writeFailLogEntry	Write the supplied text to the event log with an event type of 'Fail. The log message is prepended with "[`script: scriptFileName`] " where `scriptFileName` is the file name of the script generating the message request	String: Text to output to the event log		N
writeStatusLogEntry	Write the supplied text to the event log with an event type of 'Status. The log message is prepended with "[`script: scriptFileName`] " where `scriptFileName` is the file name of the script generating the message request	String: Text to output to the event log		N

writeSuccessLogEntry	Write the supplied text to the event log with an event type of 'Success'. The log message is prepended with "[`script: scriptFileName`] " where `scriptFileName` is the file name of the script generating the message request	String: Text to output to the event log		N
writeToFile	Write the supplied text to the specified output file PrintWriter object	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Text to write to the output file		N
writeToFileFormat	Write the supplied formatted text in the indicated format to the specified output file PrintWriter object	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Print format string to write to the output file Object…: variable list of arguments referenced by the format specifiers in the format string		N
writeToFileLn	Write the supplied text to the specified output file PrintWriter object and append a line feed character	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Text to write to the output file		N
xmlCleanSystemPath	Replace each invalid character with an underscore and move any leading underscores to the end of each path segment	String: System path in the form `<</>path1</path2<...>>`	String: Path with each invalid character replaced with an underscore and any leading underscores moved to the end of each path segment	N

xmlGetBaseDataType	Convert the primitive data type into the base equivalent used by the xtceSetParameterDataType() and xtceSetArgumentDataType() methods	String: Data type	BasePrimitiveDataType: Base primitive data type corresponding to the specified primitive data type; null if no match	N
xtceAddCommand	Add a command to the command metadata set	SpaceSystemType: Space system reference String: Command name String: Command code String: Application ID boolean: true if this table represents the command header String: Command header table system path String[]: Array of command argument names String[]: Array of of command argument data types String[]: Array of of command argument array sizes; the list item is null or blank if the corresponding argument isn't an array String: Description of the command		N

xtceAddContainerReference	Add a container reference(s) for the telemetry or command parameter or parameter array to the specified entry list	String: Reference to the telemetry or command entry list into which to place the parameter or parameter array container reference(s) String: Parameter name String: Data type String: Parameter array size; null or blank if the parameter isn't an array		N
xtceAddParameterAndType	Add a parameter with a primitive data type to the parameter set and parameter type set	SpaceSystemType: Space system reference String: Parameter name String: Parameter primitive data type String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or blank if not a bit-wise parameter String: Enumeration in the format `<enum label>|<enum value>[|...][,...]`; null to not specify String: Parameter units String: Minimum parameter value String: Maximum parameter value String: Parameter description int: Size, in characters, of a string parameter; ignored if not a string or character		N

xtceAddParameterSequenceEntry	Add the parameter to the sequence container entry list	SpaceSystemType: Reference to the space system to which the parameter belongs String: Parameter name String: Data type String: Array size EntryListType: Reference to the entry list into which to place the parameter (for a primitive data type) or container (for a structure data type) reference boolean: true if this table represents the telemetry header or one of its descendants	boolean: true if the parameter's data type references the telemetry header or one of its descendants; otherwise return the flag status unchanged	N
xtceAddSpaceSystemCommands	Add the command(s) from a table to the specified space system	SpaceSystemType: Space system reference String[][]: Table data array int: Command name column index int: Command code column index int: Command description column index boolean: true if this table represents the command header String: Command header table system path String: Application ID		N

xtceAddSpaceSystem Header	Set the space system header attributes	SpaceSystemType: Space system reference String: Classification attribute String: Validation status attribute String: Version attribute String: Export creation time and date		N

xtceAddSpaceSystem Parameters	Add a structure table's parameters to the telemetry meta data	SpaceSystemType: Space system reference String: Table name String[][]: Array containing the table's data int: Variable (parameter) name column index int: Parameter data type column index int: Parameter array size column index int: Parameter bit length column index int: Parameter enumeration column index; -1 if no the table has no enumeration column int: Parameter description column index; -1 if no the table has no description column int: Parameter units column index; -1 if no the table has no units column int: Minimum parameter value column index; -1 if no the table has no minimum column int: Maximum parameter value column index; -1 if no the table has no maximum column		N

		boolean: true if this table represents the telemetry header or one of its descendants String: Telemetry header table system path; null or blank is none boolean: true if the table is a root structure table String: Telemetry header application ID		
xtceCreateCommandMetadata	Create the space system command metadata	SpaceSystemType: Space system reference		N
xtceCreateEnumerationList	Build an enumeration list from the supplied enumeration string	SpaceSystemType: space system reference String: Enumeration in the format `<enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]`	EnumerationList:	N
xtceCreateTelemetryMetadata		SpaceSystemType: Space system reference		N
xtceCreateUnitSet	Build a unit set from the supplied units string	String: Parameter or command argument units; null to not specify	UnitSet:	N

xtceExport	Export the tables in XTCE XML format to the specified file. This is the main entry point when using a script association to perform the export. It calls the internal method to set up and parse the tables for export	String: Output file name boolean: true if the data is big endian boolean: true if the telemetry and command headers big endian String: Version attribute (for the space system headers) String: Validation status attribute (for the space system headers) String: First level classification attribute (for the space system headers) String: Second level classification attribute (for the space system headers) String: Third level classification attribute (for the space system headers)	boolean: true if an error occurred preventing exporting the project to the file	N

xtceSetArgumentDataType	Set the command argument data type and set the specified attributes	SpaceSystemType: Space system reference String: Command argument name; null to not specify String: Command argument data type; null to not specify String: Command argument array size; null or blank if the argument isn't an array String: Command argument bit length String: Command argument enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Command argument units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Command argument description ; null to not specify int: String size in bytes; ignored if the command argument does not have a string data type String: Text used to uniquely identify data types with the same name; blank if the data type has no name conflict	NameDescriptionType: Command description of the type corresponding to the primitive data type with the specified attributes set	N

xtceSetParameterDataType	Create the telemetry parameter data type and set the specified attributes	SpaceSystemType: Space system reference String: Parameter name; null to not specify String: Data type; null to not specify String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or empty if not a bit-wise parameter String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Parameter description; null to not specify int: Size, in characters, of a string parameter; ignored if not a string or character		N

Appendix C. Acronyms

Term	Definition
CCDD	CFS Command & Data Dictionary
CCSDS	Consultative Committee for Space Data Systems
cFE	Core Flight Executive
CFS	Core Flight System
CPU	Central Processing Unit
CSV	Comma-Separated Values
DBU	Database Backup
EDS	Electronic Data Sheet
GUI	Graphical User Interface
HK	Housekeeping
I/O	Input/Output
ID	Identifier
ITOS	Integrated Test and Operations System
JAR	Java Archive
JDBC	Java DataBase Connectivity
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
L&F	Look and Feel
OID	Object Identifier
OS	Operating System
PDF	Portable Document Format
PNG	Portable Network Graphics
SQL	Structured Query Language

Term	Definition
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XML	Extensible Markup Language
XTCE	XML Telemetric and Command Exchange