Definition of the Telecommand Procedure Language (TPL)

All information is subject to change without notice and does not represent a commitment on the part of BINARY SPACE.
Release 1.05 (January 2016)

Table of Contents

Table of Figures

Document Change Log

Issue	Revision	Date	Affected	Reason for change
1	1	March 2007	All	New document
1	2	July 2008	All	Added feature to develop telecommand procedures without graphical interface
1	3	August 2008	Chapter 3.1.	Added support for bandwidth measurement
1	4	May 2011	Chapter 3.1.	Added: '`GetPastValueTime`' function
1	5	January 2016	Chapter 3.3.	Added satellite tracking, pass & interlink functions

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

1. Introduction

The ***Telecommand Procedure Language*** (TPL) is the programming language of SatView™ to implement telecommand procedures. It is a macro extension to the common C++ language. The associated library provides access to all kind of information related to the telemetry data and simplifies the decision if, when and how a telecommand should be released.
A telecommand procedure is usually developed in a graphical way through a *Telecommand Procedure Display* (TPD). This kind of display automatically generates the source code resulting from the thread related flowcharts and provides all features necessary to debug a telecommand procedure.

2. Syntax

The source code of a telecommand procedure always consists of a single major procedure hosting one or more thread blocks:

```
TELECOMMAND PROCEDURE 'Name'[(argument-declaration-list)]
[variable-declarations]
BEGIN
{{THREAD 'Thread-Name'
[TCPARAMETERS T1{,Ti};]
[PARAMETERS P1{,Pi};]
BEGIN
   TPL Code
END}}
END
```

Comments:
The above notation is in *Extended Backus Naur Formalism* (EBNF).

Name	Name of the telecommand procedure		
Thread-Name	Name of a hosted thread		
$T_1, ... T_i$	Telecommand parameter identifiers		
$P_1, ... P_i$	Telemetry parameter identifiers	[*x*]:	*x* occurs once or never
argument-declaration-list	Formal argument list	{*x*}:	*x* occurs zero or more times
variable-declarations	Variable declarations	{{*x*}}:	*x* occurs at least once

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

A telecommand procedure can have zero or more formal arguments. They must be declared with the **Arguments** property page. The same applies to the local variables; they are declared with the **Variables** property page. Any number of threads (but at least one) can be hosted by a telecommand procedure. Their definition takes place via the **Threads** property page. Debugging features are available through the **Debug** property page.

The source code of a telecommand procedure is usually generated automatically when using the available graphical interface. However, in certain situations, it might be of some advantage to slightly modify the automatically generated source code or to write it completely without the support of the graphical interface, a feature which is supported too.

The following guidelines have to be considered when writing or modifying the source code of a telecommand procedure:

Code	Description

`THREAD 'Thread-Name'`
`[TCPARAMETERS T1{,Ti};]`
`[PARAMETERS P1{,Pi};]`
`BEGIN`
`while (WaitThreadActivationExpression())`
`{`
`  // Steps block code (TPL Code)`
 	
`  if (!IsThreadEnabled())`
`  { ResetThread();`
`  continue;`
`  }`
`  return TRUE;`
`}`
`END`
	The source code of each thread consists of a loop waiting for the activation expression to get true first.
	Whenever a thread is going to terminate it should check if it is enabled. If yes, it can exit (by returning true), if no it should re-enter and wait again for the activation expression to become true.
	A thread that terminates due to a failure should always return false.
	Note:
	Do not change this part of the source code.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

```
if (!CheckStepTrigger(TEXT("STEP_1")))
{ AlertStepTriggerFailure(TEXT("STEP_1"));
  return FALSE;
}
AlertStepTriggerSuccess(TEXT("STEP_1"));
if (!CheckStepBody(TEXT("STEP_1")))
{ AlertStepBodyFailure(TEXT("STEP_1"));
  return FALSE;
}

// Step body code (TPL Code)

AlertStepBodySuccess(TEXT("STEP_1"));
if (!CheckStepConfirmation(TEXT("STEP_1")))
{ AlertStepConfirmationFailure(TEXT("STEP_1"));
  return FALSE;
}
AlertStepConfirmationSuccess(TEXT("STEP_1"));
```

For each step inside a thread check the trigger first (use '**CheckStepTrigger**' and alert a failure if necessary with '**AlertStepTriggerFailure**'). If the trigger phase passed successfully this should also be notified (by '**AlertStepTriggerSuccess**'). Before the body phase of a step is started, its pre-execution expression should be checked (by using '**CheckStepBody**') and in case of a failure the appropriate notification should be done (with '**AlertStepBodyFailure**').
The same applies to the confirmation phase of a step (use '**CheckStepConfirmation**', '**AlertStepConfirmationSuccess**' or '**AlertStepConfirmationFailure**').

Note:
Do not change or delete these step related functions as they are responsible for updating the flowcharts when the telecommand procedure is executing. Only the step body code should be modified.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

Without the graphical interface:

- Only simple data types (no classes or arrays) are allowed as arguments. Variables declared inside the **`[variable-declarations]`** section must also have a simple data type. Local variables of a thread however (as part of the TPL code) may be of any type.
- Full debugging support is available for variables declared inside the **`[variable-declarations]`** section only.
- Global subroutines are allowed and should be placed between the code for the various threads.

3. Library

Various library functions supported by the TPL provide easy access to the telemetry data characteristics. Together with the standard mathematical compiler libraries almost any calculation can be performed. The library is grouped into functions related to the action-object steps and others providing support in the area of telemetry data processing.

3.1. Telemetry Functions

The following functions are supported:

Function	Description
CString **GetTMUnitTag(**void**)**	Returns the identifier of the telemetry unit that is currently processed. Note: When the Packet Telemetry Standard (CCSDS 102.0-B-2) is supported the function returns the name of the telemetry packet. For a telemetry format based standard, it results in a string with the syntax: 'FORMAT: *n*' where *n* is the frame number.
CTimeTag **GetTMUnitTime(**void**)**	Returns the time associated with the telemetry unit. Note: The time identifies the moment when the telemetry unit was received on ground including eventual corrections to compensate any delays caused by the ground segment.

UINT **GetTMUnitID(**void**)**	Returns the number of the telemetry unit. Note: When the Packet Telemetry Standard (CCSDS 102.0-B-2) is supported the function returns the *On Board Reference Time* (OBRT) of the telemetry packet. For a telemetry format based standard, it returns the number known as format counter.
BOOL **GetTMUnitData(**INT *nBytePos*, BYTE &*nValue***)**	Returns the value of a byte at the specified location `nBytePos` (>= 0) in the variable `nValue`. Note: The function returns TRUE if the specified location is valid, FALSE otherwise.
BOOL **GetTMUnitData(**INT *nBytePos*, INT *nBitPos*,INT *nLength*, ULONGLONG &*nValue***)**	Returns the value of data at the specified location `nBytePos` (>= 0), `nBitPos` (0 <= `nBitPos` < 8), `nLength` (1 <= `nLength` <= 64) in the variable `nValue`. Note: The function returns TRUE if the specified location is valid, FALSE otherwise.
WORD **GetTMUnitQuality(**void**)**	Returns the data quality indication of the telemetry unit. It may be a combination of one or more of the following values: **`TMUNIT_DATAQUALITY_GOOD`** **`TMUNIT_DATAQUALITY_BAD`** **`TMUNIT_SEQUENCEQUALITY_GOOD`** **`TMUNIT_SEQUENCEQUALITY_BAD`** **`TMUNIT_TIMECORRELATION_GOOD`** **`TMUNIT_TIMECORRELATION_BAD`** Note: The value **`TMUNIT_DATAQUALITY_NONE`** is returned in case of an error.
BOOL **SetValue(***Parameter-Id*, INT *nOccurrence*,*type-specifier nValue***)**	Sets the specified telecommand parameter `Parameter-Id` equal to the supplied (calibrated) value `nValue` at the occurrence `nOccurrence` (>= 0).

type-specifier **GetValue(***Parameter-Id***)**	Returns the current (calibrated) value of telemetry/telecommand parameter `Parameter-Id`. Note: If the parameter occurs more than once within a telemetry/telecommand unit, the function returns the value of the first occurrence.
type-specifier **GetValue(***Parameter-Id***,** INT *nOccurrence***)**	Returns the current (calibrated) value of telemetry/telecommand parameter `Parameter-Id` at the occurrence specified by `nOccurrence` (>= 0). Note: If an illegal occurrence number is specified the functions return 0.
BOOL **SetRawValue(***Parameter-Id***,** INT *nOccurrence*,*type-specifier nValue***)**	Sets the specified telecommand parameter `Parameter-Id` equal to the supplied raw value `nValue` at the occurrence `nOccurrence` (>= 0).
type-specifier **GetRawValue(***Parameter-Id***)**	Returns the current raw value of telemetry/telecommand parameter `Parameter-Id`. Note: If the parameter occurs more than once within a telemetry/telecommand unit, the function returns the value of the first occurrence.
type-specifier **GetRawValue(***Parameter-Id***,** INT *nOccurrence***)**	Returns the current raw value of telemetry/telecommand parameter `Parameter-Id` at the occurrence specified by `nOccurrence` (>= 0). Note: If an illegal occurrence number is specified the functions return 0.
CTimeTag **GetValueTime(***Parameter-Id***)**	Returns the time associated with the current value of telemetry parameter `Parameter-Id`. Note: If the parameter occurs more than once within a telemetry unit, the function returns the value of the first occurrence.

CTimeTag **GetValueTime(***Parameter-Id*, INT *nOccurrence***)**	Returns the time associated with the current value of telemetry parameter `Parameter-Id` at the occurrence specified by `nOccurrence` (>= 0). Note: If an illegal occurrence number is specified the functions return 0.
BOOL **SetTimeValue(***Parameter-Id*, INT *nOccurrence*, CONST CTimeTag &*tTime*)	Sets the specified telecommand parameter `Parameter-Id` equal to the supplied time value `tTime` at the occurrence `nOccurrence` (>= 0).
CTimeTag **GetTimeValue(***Parameter-Id*)	Returns the current time value of telecommand parameter `Parameter-Id`. Note: If the parameter occurs more than once within a telecommand unit, the function returns the value of the first occurrence.
CTimeTag **GetTimeValue(***Parameter-Id*, INT *nOccurrence*)	Returns the current time value of telecommand parameter `Parameter-Id` at the occurrence specified by `nOccurrence` (>= 0). Note: If an illegal occurrence number is specified the functions return 0.
type-specifier **GetPastValue(***Parameter-Id*, INT *nSample*)	Returns a past (calibrated) value of telemetry parameter `Parameter-Id`. The variable `nSample` specifies how many samples in the past the value should be from. Note: If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.
type-specifier **GetPastRawValue(***ParameterId*, INT *nSample*)	Returns a past raw value of telemetry parameter `Parameter-Id`. The variable `nSample` specifies how many samples in the past the value should be from. Note: If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

CTimeTag **GetPastValueTime(***Parameter-Id*, INT *nSample*)	Returns the time associated with a past value of telemetry parameter `Parameter-Id`. The variable *nSample* specifies how many samples in the past the value should be from. Note: If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.
type-specifier **CalculateValueAverage(** *Parameter-Id*,INT *nSamples*)	Returns the raw average value of the last `nSamples` samples of telemetry parameter `Parameter-Id`. Note: If less than `nSamples` samples have been collected, the function returns a floating average of the samples already encountered.
UINT **GetStatus(***Parameter-Id*)	Returns the status of the telemetry/telecommand parameter `Parameter-Id` which may be a combination of the following values: **TMPARAMETER_STATUS_GOOD** **TMPARAMETER_STATUS_BAD** **TMPARAMETER_STATUS_NOLIMIT** **TMPARAMETER_STATUS_SOFTLIMIT** **TMPARAMETER_STATUS_HARDLIMIT** **TMPARAMETER_STATUS_DELTALIMIT** **TMPARAMETER_STATUS_VALID** **TMPARAMETER_STATUS_INVALID** Note: The value **TMPARAMETER_STATUS_NONE**/ **TCPARAMETER_STATUS_NONE** is returned if the telemetry/telecommand parameter has no value. Expect the following identifiers to be returned for telecommand parameters: **TCPARAMETER_STATUS_NOLIMIT** **TCPARAMETER_STATUS_SOFTLIMIT** **TCPARAMETER_STATUS_HARDLIMIT** **TCPARAMETER_STATUS_DELTALIMIT**

UINT **GetStatus(***Parameter-Id*, INT *nOccurrence***)**	Returns the status of the telemetry/telecommand parameter `Parameter-Id` for the specified occurrence `nOccurrence`. See above for possible values returned by this function. Note: The value **`TMPARAMETER_STATUS_NONE`**/ **`TCPARAMETER_STATUS_NONE`** is returned if the telemetry/telecommand parameter has no value or an illegal occurrence number was specified. Expect the following identifiers to be returned for telecommand parameters: **`TCPARAMETER_STATUS_NOLIMIT`** **`TCPARAMETER_STATUS_SOFTLIMIT`** **`TCPARAMETER_STATUS_HARDLIMIT`** **`TCPARAMETER_STATUS_DELTALIMIT`**
double **GetTotalTMBandwidth()**	Returns the total amount of bits per second available for the telemetry data (including the protocol overhead for the telemetry unit). Note: A value of 'NAN' is returned when no bandwidth information is available. Use the macro **`isnan(`**`double f`**`)`** to check for that result.
double **GetAvailableTMBandwidth()**	Returns the currently unused bandwidth as a number between 0 and 1. Note: A value of 'NAN' is returned when no bandwidth information or measurement is available. Use the macro **`isnsn(`**`double f`**`)`** to check for that result.
double **GetMaxDiagnosticTMBandwidth()**	Returns the maximum of bits per second currently available for diagnostic purposes, dumps or reports. Note: A value of 'NAN' is returned when no bandwidth information is available. Use the macro **`isnan(`**`double f`**`)`** to check for that result.

double **GetAvailableDiagnosticTMBandwidth()**	Returns the bandwidth currently available for diagnostic purposes, dumps or reports as a number between 0 and 1. Note: A value of 'NAN' is returned when no bandwidth information or measurement is available. Use the macro **`isnan(`**`double f`**`)`** to check for that result.
CTimeTag **GetLastTMBandwidthMeasurementTime()**	Returns the time of the last bandwidth measurement. Note: A time equal to 0 is returned when no bandwidth information or measurement is available.

3.2. Action-object Step Functions

The following functions are supported:

3.2.1. Common Step Functions

Function	Description
BOOL **CheckStepTrigger(**LPCTSTR *pszStep*)	Checks the trigger condition of the specified step. Note: This function returns TRUE if no trigger condition exists.
BOOL **CheckStepBody(**LPCTSTR *pszStep*)	Checks the pre-execution expression (if any) of the specified step. Note: This function returns TRUE if no pre-execution expression exists.
BOOL **CheckStepConfirmation(** LPCTSTR *pszStep*)	Checks the confirmation condition of the specified step. Note: This function returns TRUE if no confirmation condition exists.
VOID **AlertStepTriggerSuccess(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL)	Notifies a successful trigger phase of the specified step and supplies an optional message.
VOID **AlertStepTriggerWarning(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL)	Alerts a problematic trigger phase of the specified step and supplies an optional message.

VOID **AlertStepTriggerFailure(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Alerts a failed trigger phase of the specified step and supplies an optional message.
VOID **AlertStepBodySuccess(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Notifies a successful body phase of the specified step and supplies an optional message.
VOID **AlertStepBodyWarning(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Alerts a problematic body phase of the specified step and supplies an optional message.
VOID **AlertStepBodyFailure(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Alerts a failed body phase of the specified step and supplies an optional message.
VOID **AlertStepConfirmationSuccess(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Notifies a successful confirmation phase of the specified step and supplies an optional message.
VOID **AlertStepConfirmationWarning(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Alerts a problematic confirmation phase of the specified step and supplies an optional message.
VOID **AlertStepConfirmationFailure(** LPCTSTR *pszStep*, LPCTSTR *pszMessage*=NULL**)**	Alerts a failed confirmation phase of the specified step and supplies an optional message.

3.2.2. Specific Step Functions

Function	Description
BOOL **SetProcedureState(** LPCTSTR *pszName*, UINT *nState***)**	Sets the execution state of the specified telecommand procedure. The following options are available: **TCPROCEDURE_ACTIONSTATE_SUSPEND** **TCPROCEDURE_ACTIONSTATE_RESUME** **TCPROCEDURE_ACTIONSTATE_ABORT**
BOOL **SetThreadState(** LPCTSTR *pszName*, UINT *nState***)**	Sets the execution state of the specified thread. The following options are available: **TCPROCEDURETHREAD_ACTIONSTATE_SUSPEND** **TCPROCEDURETHREAD_ACTIONSTATE_RESUME** **TCPROCEDURETHREAD_ACTIONSTATE_ENABLE** **TCPROCEDURETHREAD_ACTIONSTATE_DISABLE** **TCPROCEDURETHREAD_ACTIONSTATE_ABORT**
BOOL **CallProcedure(** LPCTSTR *pszName*,...**)**	Calls the specified telecommand procedure and waits upon its completion.
BOOL **StartProcedure(** LPCTSTR *pszName*,...**)**	Starts the specified telecommand procedure and continues its execution.

BOOL **SetSynchronizationObject(** LPCTSTR *pszSyncObject*)	Signals the specified synchronization object.
BOOL **WaitSynchronizationObject(** LPCTSTR *pszSyncObject,* DWORD *dwTimeout*)	Waits for the specified synchronization object to get signaled within the indicated amount of milliseconds.
BOOL **WaitTimeInterval(** DWORD *dwTimeInterval*)	Waits the specified amount of milliseconds.
BOOL **WaitAbsoluteTime(** CONST CTimeTag *&tTime*)	Waits until the specified absolute time.
BOOL **InjectAlert(**LPCTSTR *pszAlert*)	Injects an alert as an event. The supplied string must follow these formatting guidelines: **Flag** — **Description** /CATEGORY — Specifies the category to which the message belongs: 'System', 'Spacecraft' or 'User' /TYPE — Specifies the type of the message. Predefined values are: 'Success', 'Informational', 'Warning', 'Error' or 'Scheduled'. Other custom types are also possible. /SUBTYPE — Specifies the subtype of the message. This flag can be used together with the type to perform filtering. /MESSAGE — Specifies the message text. /COMMENT — Specifies any comment associated with the message. /AUDITION — Specifies the name of an existing audition profile to be used when the message is displayed. Note: This option only works with the 'Global Eventbox'. It is ignored for other Eventbox display windows.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

	/NOTIFICATION — Specifies the name of an existing notification profile to be used when the message is displayed. Note: This option only works with the 'Global Eventbox'. It is ignored for other Eventbox display windows.
BOOL **LogMessage(** LPCTSTR *pszFileName*, LPCTSTR *pszMessage*, BOOL *bPlainText*=FALSE, BOOL *bUnicode*=FALSE**)**	Logs a message to the specified file. The following options are available: ***bPlainText*** — **Effect** TRUE — The message is written as a normal str to the file. FALSE — The file is treated like a log file and hence not directly readable any more (for internal use only). ***bUnicode*** — **Effect** TRUE — The message is written to the file as a Unicode encoded string. FALSE — The message is written to the file as an ASCII encoded string.
BOOL **UserInteraction(**UINT *nType*, LPCTSTR *pszMessage*, CONST CStringArray &*szVariables*, CONST CStringArray &*szVariableValues***)**	Initiates a dialog box asking the operator either to acknowledge a message or to set values to a predefined set of variables. The following options are available: **Effect** *Requests the operator to just acknowledge the supplied message, **Asks the operator to set values for the supplied variables (`szVariables`), ***Asks the operator to set values out of a set of predefined values (`szVariableValues`) for the supplied variables (`szVariables`). Note: Multiple values for variables should be separated by TABs inside `szVariableValues`. ***TCPROCEDUREUSERINTERACTIONITEM_ACTION_ACKNOWLEDGE** ****TCPROCEDUREUSERINTERACTIONITEM_ACTION_REQUEST** *****TCPROCEDUREUSERINTERACTIONITEM_ACTION_CHOOSE**

BOOL **SendTCFunction(** LPCTSTR *pszName*, LPCTSTR *pszSequence*, LPCTSTR *pszFunction*, INT *nEntry*, INT *nEntries*, CONST CTimeTag &*tScheduleTime*, CONST CTimeTag &*tReleaseTime*, DWORD *dwReleaseOffset*, CONST CTimeTag &*tExecutionTime*, DWORD *dwExecutionOffset*, INT *nBlockID*, BOOL *bGrouped*, BOOL *bCritical*, UINT *nAckFlags***)**	Sends the specfied telecommand to the scheduler. The following options are available: *pszName* — Specifies the telecommand function. *pszSequence* — Specifies the telecommand sequence that hosts the function; otherwise it is an em string. *nEntry* — Specifies the entry number (> 0) if the telecommand function is part of a sequence; otherwise it is -1. *nEntries* — Specifies the total number of entries that the sequence contains. If the telecommand function is not part of a sequence it is -1. *tScheduleTime* — Contains the time at which this telecommand function was scheduled. This time must be identical for all entries of a telecommand sequence *tReleaseTime* — Contains the time at which this telecommand function should be released. It is 0 for immediate release. *dwReleaseOffset* — Specifies the release time offset with respect to the start of the sequence. If the telecommand function is not part of a sequence it is 0. Note: This offset must be considered in the absolute release time already and is used for display purposes only.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

tExecutionTime	Specifies the time at which the telecommand function should be executed on-board of the spacecraft; otherwise it is 0.
dwExecutionOffset	Specifies the execution time offset with respect to the start of the sequence. If the telecommand function is not part of a sequence it is 0. Note: This offset must be considered in the absolute execution time already and is used for display purposes only.
nBlockID	Specifies the block identifier (>= 0) for a blocked telecommand. If the telecommand function is not part of a sequence or if it is not blocked it is -1.
bGrouped	Indicates if the telecommand function is to be grouped with the previous entry in the sequence. It is FALSE if the telecommand function is not part of a sequence.
bCritical	Indicates if the telecommand function needs to be acknowledged before being released.
nAckFlags	Specifies one or more of the options* below when the telecommand function is part of a sequence; otherwise one or more of these options**. The options indicate if the operator should acknowledge a completed verification step.

	* TCSEQUENCEENTRY_ACKFLAG_ACCEPTANCE TCSEQUENCEENTRY_ACKFLAG_START TCSEQUENCEENTRY_ACKFLAG_PROGRESS TCSEQUENCEENTRY_ACKFLAG_COMPLETION ** TCFUNCTION_ACKFLAG_ACCEPTANCE TCFUNCTION_ACKFLAG_START TCFUNCTION_ACKFLAG_PROGRESS TCFUNCTION_ACKFLAG_COMPLETION

3.2.3. Thread Functions

Function	Description
BOOL **WaitThreadActivationExpression()**	Waits until the thread activation expression returns TRUE. Note: This function returns FALSE whenever the thread execution has been aborted.
VOID **EnableThread**(BOOL *bEnable*)	Enables/disables the current thread.
BOOL **IsThreadEnabled()**	Returns TRUE if the current thread is enabled.
VOID **ResetThread()**	Resets the current thread.

3.3. Satellite Tracking, Pass & Interlink Functions

An extensive interface is provided by the TPL to support satellite tracking, location pass predictions as well as satellite interlink calculations.

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>double CalculateSpacecraftOrbitLongitude(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td><td>Returns the longitude of the specified spacecraft <pszSpacecraft,nNORADID> at the time tTime.
Note:
• This function is available for Earth-centric spacecraft only
• The parameter tTime must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned longitude will be between 0...360 degrees
<table><tr><th>Argument</th><th>Description</th></tr><tr><td>pszSpacecraft</td><td>The name of spacecraft.</td></tr><tr><td>nNORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr><tr><td>tTime</td><td>The time for which the longitude should be calculated.</td></tr></table></td></tr>
<tr><td>double CalculateSpacecraftOrbitLatitude(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td><td>Returns the latitude of the specified spacecraft <pszSpacecraft,nNORADID> at the time tTime.
Note:
• This function is available for Earth-centric spacecraft only
• The parameter tTime must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned latitude will be between -90...90 degrees
<table><tr><th>Argument</th><th>Description</th></tr><tr><td>pszSpacecraft</td><td>The name of spacecraft.</td></tr><tr><td>nNORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr><tr><td>tTime</td><td>The time for which the latitude should be calculated.</td></tr></table></td></tr>
</table>

<table>
<tr>
<td>double CalculateSpacecraftOrbitAltitude(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td>
<td>Returns the altitude of the specified spacecraft <pszSpacecraft,nNORADID> at the time tTime.
Note:
• This function is available for Earth-centric spacecraft only
• The parameter tTime must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned altitude will be > 0 km
Argument — Description
pszSpacecraft — The name of spacecraft.
nNORADID — The NORAD identifier of the specified spacecraft.
tTime — The time for which the altitude should be calculated.</td>
</tr>
<tr>
<td>double CalculateSpacecraftOrbitVelocity(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td>
<td>Returns the velocity of the specified spacecraft <pszSpacecraft,nNORADID> at the time tTime.
Note:
• This function is available for Earth-centric spacecraft only
• The parameter tTime must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned velocity will be > 0 km/s
Argument — Description
pszSpacecraft — The name of spacecraft.
nNORADID — The NORAD identifier of the specified spacecraft.
tTime — The time for which the velocity should be calculated.</td>
</tr>
</table>

<table>
<tr>
<td>CSpacecraftPosition
CalculateSpacecraftPosition(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td>
<td>Returns the position (relative to the Sun) of the specified spacecraft <<i>pszSpacecraft</i>, <i>nNORADID</i>> at the time <i>tTime</i>.
Note:
• For Earth-centric spacecraft (<i>nNORADID</i> <> 0) the parameter <i>tTime</i> must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned position will be returned in form of the class 'CSpacecraftPosition'; its members <i>m_x</i>, <i>m_y</i>, <i>m_z</i> contain the position coordinates in km
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>pszSpacecraft</td><td>The name of spacecraft.</td></tr>
<tr><td>nNORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td>tTime</td><td>The time for which the position (relative to the Sun) should be calculated.</td></tr>
</table>
</td>
</tr>
<tr>
<td>CSpacecraftVelocity
CalculateSpacecraftVelocity(
LPCTSTR pszSpacecraft,UINT nNORADID,
CONST CTimeKey &tTime)</td>
<td>Returns the velocity (relative to the Sun) of the specified spacecraft <<i>pszSpacecraft</i>, <i>nNORADID</i>> at the time <i>tTime</i>.
Note:
• For Earth-centric spacecraft (<i>nNORADID</i> <> 0) the parameter <i>tTime</i> must be within an interval of a few days from current real-time in order to guarantee a precise result
• The returned velocity will be returned in form of the class 'CSpacecraftVelocity'; its members <i>m_x</i>, <i>m_y</i>, <i>m_z</i> contain the velocity coordinates in km/s
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>pszSpacecraft</td><td>The name of spacecraft.</td></tr>
<tr><td>nNORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td>tTime</td><td>The time for which the velocity (relative to the Sun) should be calculated.</td></tr>
</table>
</td>
</tr>
</table>

TIMETAG **CalculateSpacecraftPassStartTime(**
LPCTSTR *pszSpacecraft*,UINT *nNORADID*,
LPCTSTR *pszLocation*,
double *fLocationLongitude*,
double *fLocationLatitude*,
double *fLocationAltitude*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the begin of the next pass over the location <*pszLocation*,*fLocationLongitude*, *fLocationLatitude*,*fLocationAltitude*> of the specified spacecraft <*pszSpacecraft*, *nNORADID*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStartTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraft	The name of spacecraft.
nNORADID	The NORAD identifier of the specified spacecraft.
pszLocation	The name of pass-over location.
fLocationLongitude	The longitude (deg) of the pass-over location.
fLocationLatitude	The latitude (deg) of the pass-over location.
fLocationAltitude	The altitude (km) of the pass-over location.
tStartTime	Specifies the start time to be used to calculate the next pass over the specified location.
tInterval	Specifies the interval to be used to calculate the next pass over the specified location.

TIMETAG **CalculateSpacecraftPassStopTime(**
LPCTSTR *pszSpacecraft*,UINT *nNORADID*,
LPCTSTR *pszLocation*,
double *fLocationLongitude*,
double *fLocationLatitude*,
double *fLocationAltitude*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the end of the next pass over the location <*pszLocation*,*fLocationLongitude*, *fLocationLatitude*,*fLocationAltitude*> of the specified spacecraft <*pszSpacecraft*, *nNORADID*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStartTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraft	The name of spacecraft.
nNORADID	The NORAD identifier of the specified spacecraft.
pszLocation	The name of pass-over location.
fLocationLongitude	The longitude (deg) of the pass-over location.
fLocationLatitude	The latitude (deg) of the pass-over location.
fLocationAltitude	The altitude (km) of the pass-over location.
tStartTime	Specifies the start time to be used to calculate the next pass over the specified location.
tInterval	Specifies the interval to be used to calculate the next pass over the specified location.

TIMETAG
CalculateSpacecraftInterlinkStartTime(
LPCTSTR *pszSpacecraftA*,UINT *nNORADIDA*,
LPCTSTR *pszSpacecraftB*,UINT *nNORADIDB*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the begin of the next interlink session between the spacecraft <*pszSpacecraftA*, *nNORADIDA*> and <*pszSpacecraftB*, *nNORADIDB*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStarTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraftA	The name of first spacecraft.
nNORADIDA	The NORAD identifier of the first spacecraft.
pszSpacecraftB	The name of second spacecraft.
nNORADIDB	The NORAD identifier of the second spacecraft.
tStartTime	Specifies the start time to be used to calculate the next interlink session.
tInterval	Specifies the interval to be used to calculate the next interlink session.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

TIMETAG
CalculateSpacecraftInterlinkStopTime(
LPCTSTR *pszSpacecraftA*,UINT *nNORADIDA*,
LPCTSTR *pszSpacecraftB*,UINT *nNORADIDB*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the end of the next interlink session between the spacecraft <*pszSpacecraftA*, *nNORADIDA*> and <*pszSpacecraftB*, *nNORADIDB*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStartTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraftA	The name of first spacecraft.
nNORADIDA	The NORAD identifier of the first spacecraft.
pszSpacecraftB	The name of second spacecraft.
nNORADIDB	The NORAD identifier of the second spacecraft.
tStartTime	Specifies the start time to be used to calculate the next interlink session.
tInterval	Specifies the interval to be used to calculate the next interlink session.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

TIMETAG
CalculateSpacecraftRelaidInterlinkStartTime(
LPCTSTR *pszSpacecraftA*,UINT *nNORADIDA*,
LPCTSTR *pszSpacecraftVia*,UINT *nNORADIDVia*,
LPCTSTR *pszSpacecraftB*,UINT *nNORADIDB*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the begin of the next interlink session between the spacecraft <*pszSpacecraftA*,*nNORADIDA*> and <*pszSpacecraftB*,*nNORADIDB*> via the relais <*pszSpacecraftVia*,*nNORADIDVia*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStartTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraftA	The name of first spacecraft.
nNORADIDA	The NORAD identifier of the first spacecraft.
pszSpacecraftVia	The name of relais spacecraft.
nNORADIDVia	The NORAD identifier of the relais spacecraft.
pszSpacecraftB	The name of second spacecraft.
nNORADIDB	The NORAD identifier of the second spacecraft.
tStartTime	Specifies the start time to be used to calculate the next interlink session.
tInterval	Specifies the interval to be used to calculate the next interlink session.

TIMETAG
CalculateSpacecraftRelaidInterlinkStopTime(
LPCTSTR *pszSpacecraftA*,UINT *nNORADIDA*,
LPCTSTR *pszSpacecraftVia*,UINT *nNORADIDVia*,
LPCTSTR *pszSpacecraftB*,UINT *nNORADIDB*,
CONST CTimeKey &*tStartTime*,
CONST CTimeSpan &*tInterval*)

Returns the end of the next interlink session between the spacecraft <*pszSpacecraftA*,*nNORADIDA*> and <*pszSpacecraftB*,*nNORADIDB*> via the relais <*pszSpacecraftVia*,*nNORADIDVia*> after the time *tStartTime* and within the subsequent *tInterval* interval.

Note:

- This function is available for Earth-centric spacecraft only
- The parameter *tStartTime* must be within an interval of a few days from current real-time in order to guarantee a precise result

Argument	Description
pszSpacecraftA	The name of first spacecraft.
nNORADIDA	The NORAD identifier of the first spacecraft.
pszSpacecraftVia	The name of relais spacecraft.
nNORADIDVia	The NORAD identifier of the relais spacecraft.
pszSpacecraftB	The name of second spacecraft.
nNORADIDB	The NORAD identifier of the second spacecraft.
tStartTime	Specifies the start time to be used to calculate the next interlink session.
tInterval	Specifies the interval to be used to calculate the next interlink session.

Note:
All satellite tracking, pass & interlink functions cannot be tested within the SatView™ Editor; they all return 'NAN' (for 'double' data types) and '0' (for 'TIMETAG' data types). When executed within the SatView™ Desktop, the satellite tracking sub-system must be enabled for these functions to return valid results. Furthermore, it must be ensured that access to the Internet is guaranteed.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

Helper Functions:

Function	Description
UINT **ConvertBinaryStringToValue(** LPCTSTR *pszValue*)	Converts a binary encoded string into an unsigned integer.
UINT **ConvertOctalStringToValue(** LPCTSTR pszValue)	Converts a string representing an octal number into an unsigned integer.
UINT **ConvertHexadecimalStringToValue(** LPCTSTR pszValue)	Converts a string representing a hexadecimal number into an unsigned integer.
TIMETAG **ConvertTimeStringToValue(** LPCTSTR *pszTime*)	Converts a string representing a time into a variable of type TIMETAG. Note: The string must be formatted as follows: DD/MM/YYYY HH:MM:SS.nnn

Data Types:

Identifier	Description
type-specifier	Depending on the data type of the telemetry parameter P_i it is either an UINT, INT, double or CString.
parameter-tag	Tag of the telemetry parameter P_i.
CSpacecraftPosition	A class representing the position of a spacecraft (relative to the Sun for all non Earth-centric ones). The following member properties are available: double **m_x** double **m_y** double **m_z**
CSpacecraftVelocity	A class representing the velocity of a spacecraft (relative to the Sun for all non Earth-centric ones). The following member properties are available: double **m_x** double **m_y** double **m_z**
CString	A class representing a string. Consult the *Microsoft® Foundation Class* (MFC) documentation.
CTimeTag TIMETAG	A class representing an absolute time in microseconds since January 1, 1970. Consult the *Microsoft® Foundation Class* (MFC) documentation (see the 'CTime' class).

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

CTimeKey TIMEKEY	A class representing an absolute time in seconds since January 1, 1970. Consult the *Microsoft® Foundation Class* (MFC) documentation (see the 'CTime' class).
CTimeSpan	A class representing a time interval in seconds. Consult the *Microsoft® Foundation Class* (MFC) documentation.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com

4. Samples

```
TELECOMMAND PROCEDURE  '100'
BEGIN
THREAD  'Primary Thread'
BEGIN
  while (WaitThreadActivationExpression()) // Waits until the thread activation expression returns true
  {
    if (!CheckStepTrigger(TEXT("STEP_1"))) // Checks the trigger of the step 'STEP_1' to see if it can be executed
    { AlertStepTriggerFailure(TEXT("STEP_1")); // A failed step trigger is alerted and the thread terminates
      return FALSE; // A thread that terminates with a failure returns false
    }
    for (AlertStepTriggerSuccess(TEXT("STEP_1")); !CheckStepBody(TEXT("STEP_1")); ) // Signals a successful trigger phase and checks if the body phase of the step can be started
    { AlertStepBodyFailure(TEXT("STEP_1"));
      return FALSE;
    }
    if (!SendTCFunction(TEXT("STEP_1"),TEXT(""),TEXT("Z0001"),-1,-1,0,0,0,-1,FALSE,FALSE,0)) // Sends a telecommand
    { AlertStepBodyFailure(TEXT("STEP_1"),TEXT("The telecommand function 'Z0001' could not be scheduled."));
      return FALSE;
    }
    for (AlertStepBodySuccess(TEXT("STEP_1")); !CheckStepConfirmation(TEXT("STEP_1")); ) // Signals a successful body phase and checks if the confirmation phase of the step can be started
    { AlertStepConfirmationFailure(TEXT("STEP_1"));
      return FALSE;
    }
    AlertStepConfirmationSuccess(TEXT("STEP_1"));
    if (!IsThreadEnabled()) // A disabled thread that terminates re-enters
    { ResetThread();
      continue;
    }
    return TRUE; // A thread that terminates successfully returns true
  }
END
END
```

A. Acceptance

This document has been read and accepted by ESA.

In der Weid 3, CH-8122 Binz
Tel: +41 44 8877987, Fax: +41 44 8877989, Email: info@binary-space.com, Web: www.binary-space.com