BINARY SPACE
RELIABLE SPACE SYSTEMS

Definition of the Derived Parameter Language (DPL)

All information is subject to change without notice and does not represent a commitment on the part of **BINARY SPACE**.
Release 1.06 (January 2016)

Table of Contents

Appendix

Table of Figures

Figure	Description

Document Change Log

Issue	Revision	Date	Affected	Reason for change
1	1	December 1999	All	New document
1	2	September 2001	Chapter 2	Introduction of the '`VOLATILE`' keyword for non-initialized parameters
1	3	August 2004	Chapter 3.1.	Changed parameter status identifiers
1	4	August 2008	Chapter 3.1.	Added support for bandwidth measurement
1	5	May 2011	Chapter 3.1.	Added '`GetPastValueTime`' function
1	6	January 2016	Chapter 3.2.	Added satellite tracking, pass & interlink functions

1. Introduction

The ***Derived Parameter Language*** (DPL) is the programming language of SatView™ developed to ease the implementation of derived parameters. It is a macro extension to the common C++ language and comes with a library containing useful telemetry data functions.

2. Syntax

Code written in DPL always consists of one or more procedures, each associated with a derived parameter. The procedures are executed automatically whenever one or more of the parameters $P_1, ... P_k$ declared inside are updated.

A procedure complying with the following syntax needs to be implemented for every parameter:

```
DERIVATION PROCEDURE P_i
[PARAMETERS P_1{,P_j};]
[VOLATILE P_1{,P_k};]
BEGIN
    DPL Code
END
```

Comments:

The above notation is expressed in the *Extended Backus Naur Formalism* (EBNF).

P_i Derived parameter to be calculated.

$P_1, ... P_j$ Parameters used for the derivation calculation. A parameter declaration P_n can be preceded by the **STATIC** keyword which has the effect that the derived parameter is not re-calculated even if the parameter P_n is updated.

$P_1, ... P_k$ Parameters eventually used for the derivation calculation.

By using the **VOLATILE** keyword the following enumerated parameters are not checked whether or not they are initialized (i.e. do have a value). Each of the specified parameters must be contained in the list declared with the **PARAMETERS** keyword and may have a random value. By default, the procedure is not executed whenever one of the parameters $P_1 ... P_j$ has no value. With the **VOLATILE** keyword specified, the calculation in the procedure is performed although one or more of the parameters $P_1 ... P_k$ have no (or a random) value.

3. Library

3.1. Telemetry Functions

DPL comes with a set of functions providing full access to the telemetry data. Besides including other (derived) parameters into a calculation it also provides a programmatic interface to raw data.

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.

type-specifier **GetValue(***Parameter-Id***)**	Returns the current (calibrated) value of parameter *`Parameter-Id`*. Note: If the parameter occurs more than once within a telemetry unit, the function returns the value of the first occurrence.
type-specifier **GetValue(***Parameter-Id*, INT *nOccurrence***)**	Returns the current (calibrated) value of parameter *`Parameter-Id`* at the occurrence specified by *`nOccurrence`* (>= 0). Note: If an illegal occurrence number is specified the functions return 0.
type-specifier **GetRawValue(***Parameter-Id***)**	Returns the current raw value of parameter *`Parameter-Id`*. Note: If the parameter occurs more than once within a telemetry unit, the function returns the value of the first occurrence.
type-specifier **GetRawValue(***Parameter-Id*, INT *nOccurrence***)**	Returns the current raw value of 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 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 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 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(** _Parameter-Id_**,**INT _nSample_**)**	Returns a past raw value of 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.
CTimeTag **GetPastValueTime(** _Parameter-Id_**,**INT _nSample_**)**	Returns the time associated with a past value of 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 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 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** is returned if the parameter has no value.
UINT **GetStatus(***Parameter-Id*, INT *nOccurrence***)**	Returns the status of the parameter *`Parameter-Id`* for the specified occurrence *`nOccurrence`*. See above for possible values returned by this function. Note: The value **TMPARAMETER_STATUS_NONE** is returned if the parameter has no value or an illegal occurrence number was specified.
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 **`isnan(`**`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. Satellite Tracking, Pass & Interlink Functions

An extensive interface is provided by the DPL 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 <b>CalculateSpacecraftOrbitLongitude(</b>
LPCTSTR <i>pszSpacecraft</i>,UINT <i>nNORADID</i>,
CONST CTimeKey &<i>tTime</i><b>)</b></td><td>Returns the longitude of the specified spacecraft &lt;<i>pszSpacecraft</i>,<i>nNORADID</i>&gt; at the time <i>tTime</i>.
📝 Note:
• This function is available for Earth-centric spacecraft only
• 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 longitude will be between 0...360 degrees
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraft</i></td><td>The name of spacecraft.</td></tr>
<tr><td><i>nNORADID</i></td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td><i>tTime</i></td><td>The time for which the longitude should be calculated.</td></tr>
</table></td></tr>
<tr><td>double <b>CalculateSpacecraftOrbitLatitude(</b>
LPCTSTR <i>pszSpacecraft</i>,UINT <i>nNORADID</i>,
CONST CTimeKey &<i>tTime</i><b>)</b></td><td>Returns the latitude of the specified spacecraft &lt;<i>pszSpacecraft</i>,<i>nNORADID</i>&gt; at the time <i>tTime</i>.
📝 Note:
• This function is available for Earth-centric spacecraft only
• 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 latitude will be between -90...90 degrees
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraft</i></td><td>The name of spacecraft.</td></tr>
<tr><td><i>nNORADID</i></td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td><i>tTime</i></td><td>The time for which the latitude should be calculated.</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td>double <b>CalculateSpacecraftOrbitAltitude(</b>
LPCTSTR <i>pszSpacecraft</i>,UINT <i>nNORADID</i>,
CONST CTimeKey &amp;<i>tTime</i><b>)</b></td>
<td>Returns the altitude of the specified spacecraft &lt;<i>pszSpacecraft</i>,<i>nNORADID</i>&gt; at the time <i>tTime</i>.
📝 Note:
• This function is available for Earth-centric spacecraft only
• 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 altitude will be > 0 km
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraft</i></td><td>The name of spacecraft.</td></tr>
<tr><td><i>nNORADID</i></td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td><i>tTime</i></td><td>The time for which the altitude should be calculated.</td></tr>
</table>
</td>
</tr>
<tr>
<td>double <b>CalculateSpacecraftOrbitVelocity(</b>
LPCTSTR <i>pszSpacecraft</i>,UINT <i>nNORADID</i>,
CONST CTimeKey &amp;<i>tTime</i><b>)</b></td>
<td>Returns the velocity of the specified spacecraft &lt;<i>pszSpacecraft</i>,<i>nNORADID</i>&gt; at the time <i>tTime</i>.
📝 Note:
• This function is available for Earth-centric spacecraft only
• 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 > 0 km/s
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraft</i></td><td>The name of spacecraft.</td></tr>
<tr><td><i>nNORADID</i></td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td><i>tTime</i></td><td>The time for which the velocity should be calculated.</td></tr>
</table>
</td>
</tr>
</table>

CSpacecraftPosition **CalculateSpacecraftPosition(** LPCTSTR *pszSpacecraft*,UINT *nNORADID*, CONST CTimeKey &*tTime*)	Returns the position (relative to the Sun) of the specified spacecraft <*pszSpacecraft*, *nNORADID*> at the time *tTime*. Note: • For Earth-centric spacecraft (*nNORADID* <> 0) 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 position will be returned in form of the class 'CSpacecraftPosition'; its members *m_x*, *m_y*, *m_z* contain the position coordinates in km **Argument** — **Description** *pszSpacecraft* — The name of spacecraft. *nNORADID* — The NORAD identifier of the specified spacecraft. *tTime* — The time for which the position (relative to the Sun) should be calculated.
CSpacecraftVelocity **CalculateSpacecraftVelocity(** LPCTSTR *pszSpacecraft*,UINT *nNORADID*, CONST CTimeKey &*tTime*)	Returns the velocity (relative to the Sun) of the specified spacecraft <*pszSpacecraft*, *nNORADID*> at the time *tTime*. Note: • For Earth-centric spacecraft (*nNORADID* <> 0) 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 returned in form of the class 'CSpacecraftVelocity'; its members *m_x*, *m_y*, *m_z* contain the velocity coordinates in km/s **Argument** — **Description** *pszSpacecraft* — The name of spacecraft. *nNORADID* — The NORAD identifier of the specified spacecraft. *tTime* — The time for which the velocity (relative to the Sun) should be calculated.

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.

<table>
<tr>
<td>TIMETAG
<b>CalculateSpacecraftInterlinkStartTime(</b>
LPCTSTR <i>pszSpacecraftA</i>,UINT <i>nNORADIDA</i>,
LPCTSTR <i>pszSpacecraftB</i>,UINT <i>nNORADIDB</i>,
CONST CTimeKey &amp;<i>tStartTime</i>,
CONST CTimeSpan &amp;<i>tInterval</i><b>)</b></td>
<td>Returns the begin of the next interlink session between the spacecraft &lt;<i>pszSpacecraftA</i>, <i>nNORADIDA</i>&gt; and &lt;<i>pszSpacecraftB</i>, <i>nNORADIDB</i>&gt; after the time <i>tStartTime</i> and within the subsequent <i>tInterval</i> interval.
Note:
<ul>
<li>This function is available for Earth-centric spacecraft only</li>
<li>The parameter <i>tStarTime</i> must be within an interval of a few days from current real-time in order to guarantee a precise result</li>
</ul>
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraftA</i></td><td>The name of first spacecraft.</td></tr>
<tr><td><i>nNORADIDA</i></td><td>The NORAD identifier of the first spacecraft.</td></tr>
<tr><td><i>pszSpacecraftB</i></td><td>The name of second spacecraft.</td></tr>
<tr><td><i>nNORADIDB</i></td><td>The NORAD identifier of the second spacecraft.</td></tr>
<tr><td><i>tStartTime</i></td><td>Specifies the start time to be used to calculate the next interlink session.</td></tr>
<tr><td><i>tInterval</i></td><td>Specifies the interval to be used to calculate the next interlink session.</td></tr>
</table>
</td>
</tr>
</table>

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.

<table>
<tr>
<td>TIMETAG
<b>CalculateSpacecraftRelaidInterlinkStartTime(</b>
LPCTSTR <i>pszSpacecraftA</i>,UINT <i>nNORADIDA</i>,
LPCTSTR <i>pszSpacecraftVia</i>,UINT <i>nNORADIDVia</i>,
LPCTSTR <i>pszSpacecraftB</i>,UINT <i>nNORADIDB</i>,
CONST CTimeKey &<i>tStartTime</i>,
CONST CTimeSpan &<i>tInterval</i><b>)</b></td>
<td>Returns the begin of the next interlink session between the spacecraft &lt;<i>pszSpacecraftA</i>,<i>nNORADIDA</i>&gt; and &lt;<i>pszSpacecraftB</i>,<i>nNORADIDB</i>&gt; via the relais &lt;<i>pszSpacecraftVia</i>,<i>nNORADIDVia</i>&gt; after the time <i>tStartTime</i> and within the subsequent <i>tInterval</i> interval.
Note:
• This function is available for Earth-centric spacecraft only
• The parameter <i>tStartTime</i> must be within an interval of a few days from current real-time in order to guarantee a precise result
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><i>pszSpacecraftA</i></td><td>The name of first spacecraft.</td></tr>
<tr><td><i>nNORADIDA</i></td><td>The NORAD identifier of the first spacecraft.</td></tr>
<tr><td><i>pszSpacecraftVia</i></td><td>The name of relais spacecraft.</td></tr>
<tr><td><i>nNORADIDVia</i></td><td>The NORAD identifier of the relais spacecraft.</td></tr>
<tr><td><i>pszSpacecraftB</i></td><td>The name of second spacecraft.</td></tr>
<tr><td><i>nNORADIDB</i></td><td>The NORAD identifier of the second spacecraft.</td></tr>
<tr><td><i>tStartTime</i></td><td>Specifies the start time to be used to calculate the next interlink session.</td></tr>
<tr><td><i>tInterval</i></td><td>Specifies the interval to be used to calculate the next interlink session.</td></tr>
</table>
</td>
</tr>
</table>

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.

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).
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.

4. Samples

```
DERIVATION PROCEDURE  E015
PARAMETERS  R301;
BEGIN
   E015 = (R301 == TEXT("PEAM")) ? TRUE:FALSE;
END

DERIVATION PROCEDURE  E730
PARAMETERS  E722P,O372,R372;
BEGIN
   E730 = (R372-E722P)/O372;
END

DERIVATION PROCEDURE  E735
PARAMETERS  E864,E865,E866;
BEGIN
   E735 = -0.02163*E864+0.02163*E865+0.052*E866;
END

DERIVATION PROCEDURE  PV407
PARAMETERS  F116;
BEGIN
   PV407 = (F116 == 0x73) ? TRUE:FALSE;
END

DERIVATION PROCEDURE  A015
BEGIN
   A015 = (CalculateSpacecraftOrbitAltitude(TEXT("ISS (ZAYRA)"),25544,
GetTMUnitTime().GetTimeInSeconds()) < 250.0) ? TRUE:FALSE;
END
```

A. Acceptance

This document has been read and accepted by ESA.