BINARY SPACE
RELIABLE SPACE SYSTEMS

# SatView™ Automation Scripts Interface

All information is subject to change without notice and does not represent a commitment on the part of **BINARY SPACE**.
Release 1.08 (January 2016)

## Table of Contents


**1. Introduction**
**2. Coding Guidelines**
**3. Arguments**
**4. Libraries**
**5. Interfaces**
**6. Samples**


## Appendix


**A. Acceptance**


## Table of Figures


| Figure | Description |
|---|---|
| 1.-1 | Automation Script Example |


## Document Change Log

| Issue | Revision | Date | Affected | Reason for change |
|---|---|---|---|---|
| 1 | 1-5 | October 2005 | All | New document |
| 1 | 6 | September 2009 | Chapter 5.1.<br>Chapter 5.2. | Added task scheduler interface and telemetry data functions |
| 1 | 7 | May 2011 | Chapter 5.2. | New telemetry data functions '**GetPastValueTime**', '**Lock**' and '**Unlock**' |
| 1 | 8 | January 2016 | Chapter 5.3. | Added satellite tracking interface, location pass as well as interlink prediction support |


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

Automation scripts can be a useful tool to help automating some processes in particular in the area of telemetry data visualization, analyzing or reporting. These scripts are written in the programming language JScript .NET® and require a compilation before they can be used. It is also possible to test them offline first in order to guarantee a correct functioning. The execution of automation scripts can be triggered by a time schedule, *System Alerts*, *Telemetry Events* or by changes inside monitored directories. The following figure shows an example how automation scripts can be used within SatView™:

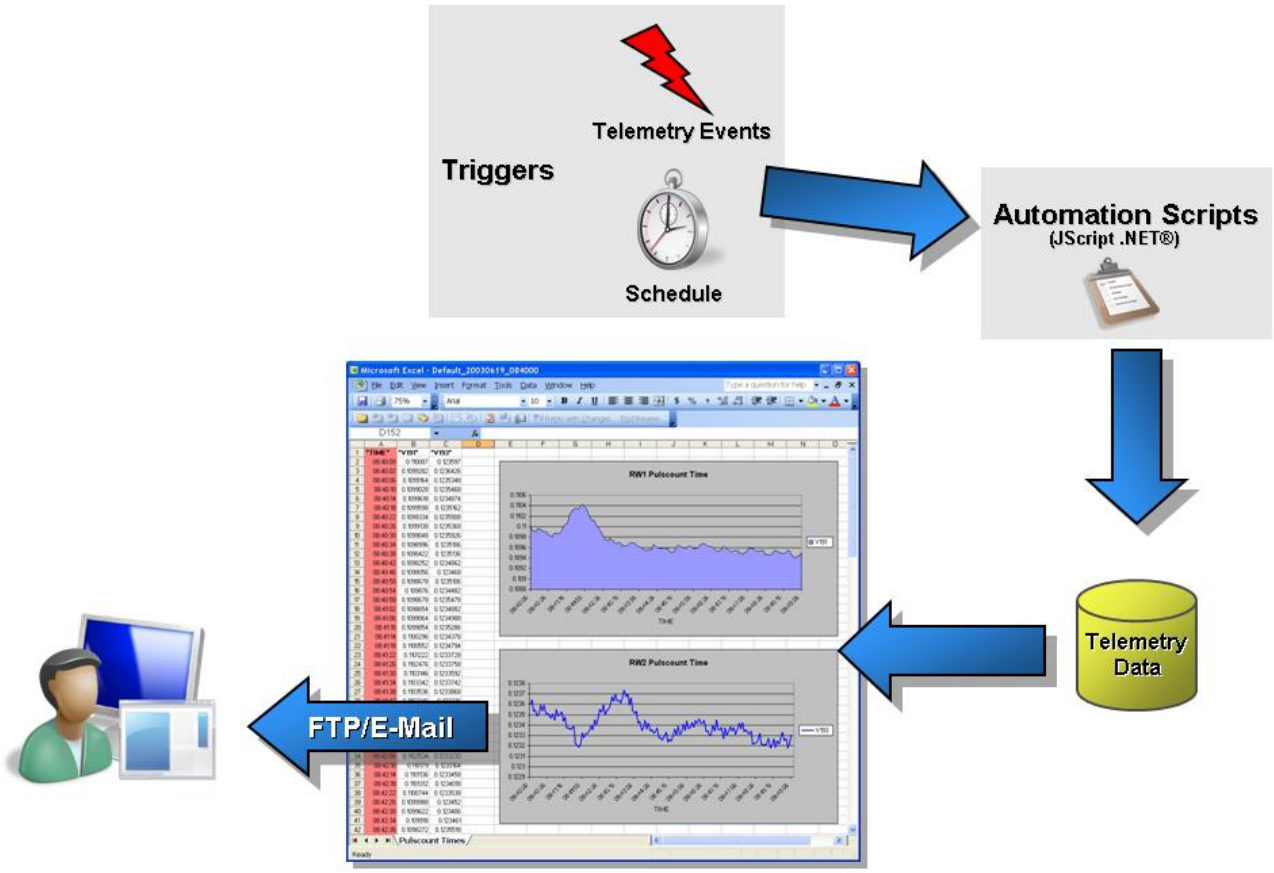


Figure 1.-1 Automation Script Example

## 2. Coding Guidelines

The following points must be considered when writing an automation script:

- Good knowledge about coding practices with JScript .NET®.
- Always use a **try/catch** exception handler in order to handle any faults:

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

```
try
{
   JScript .NET® code
}
catch (exception)
{
   Cleanup code
}
```

Multiple **try**/**catch** blocks are allowed but must be nested in a way that only one remains at the outermost level.

- Add the following code segment at the beginning of the last (outermost) catch block:

```
catch (exception)
{
   // Provide an error reason
   Error.Insert(0,exception.message);
   Cleanup code
   return Exit(false); // A trailing return statement is optional but can be used to indicate a failure when false is provided as an argument to the Exit function
}
```

This code returns an error reason and hence simplifies the code debugging.

- Close all open items (like files, workbooks etc.) in the **catch** block.
- It is advisable to prevent any required user interaction while an automation script is executing. For automating Microsoft® Excel®, for example, check for an existing copy of the output file first before saving it:

```
// Sample for Microsoft® Excel®
if (File.Exists(OutputFile.ToString())) Book.Save();
else Book.SaveAs(OutputFile.ToString());
```

This prevents any undesired input prompt asking if the file should be overwritten.

## 3. Arguments

When an automation script is called, several arguments are passed to it that can be used within the automation script.

| Argument | Type | Description | Calling Context |
|---|---|---|---|
| **Name** | String *<in>* | Contains the name of the calling telemetry parameter extraction or of the telemetry report. | Telemetry Parameter Extractions<br>Telemetry Reports |

| | | | |
|---|---|---|---|
| | | Note:<br>While debugging an automation script an empty string is supplied (manual execution). | |
| **Script** | String<br><*in*> | Supplies the name and title of the automation script in the format:<br>***Name***<*Tabulator*>***Title*** | Telemetry Parameter Extractions<br>Telemetry Reports |
| **InputFile** | String<br><*in*> | Contains a file path supplied by the telemetry report profile that should be processed as input. | Telemetry Reports |
| **OutputFile** | String<br><*in*> | Contains a file path supplied by the telemetry report profile that should be used for the resulting output. | Telemetry Reports |
| **Alert** | String<br><*in*> | Contains the name of the *System Alert* that caused the automation script to be executed.<br>Note:<br>This string is empty if a telemetry event triggered the automation script, if the execution was scheduled or started by a file trigger. | Telemetry Reports |
| **Event** | String<br><*in*> | Contains the name of the *Telemetry Event* that triggered the execution of the automation script.<br>Note:<br>This string is empty if a system alert triggered the automation script, if the execution was scheduled or started by a file trigger. | Telemetry Reports |
| **Message** | String<br><*in*> | Contains a message specific to the situation.<br>Note:<br>This variable is empty if neither a system alert nor a telemetry event triggered the automation script. | Telemetry Reports |
| **ExtraInfo** | String<br><*in*> | Contains additional information related to the message. Whenever a system alert is triggered the automation script it may contain one of these strings:<br>**System Alert** / **ExtraInfo**<br>Completion of history file backups<br>Failures during history file backups: The path name of the history file that was backup (not the resulting file).<br>Completion of history file extractions<br>Failures during history file extractions: The path name of the extracted history file. | Telemetry Reports |

| | | | |
|---|---|---|---|
| | | Completion of memory dumps<br>Failures during the processing of memory dumps: The path name of all dump files that were generated by the memory dump.<br>Note: Multiple file names are separated by a semicolon.<br>Completion of telemetry data extractions<br>Failures during telemetry data extractions: The path name of the file that was generated by the telemetry data extraction.<br>Failures during telemetry data archiving: The path name of the history file that could not be updated.<br>Successful log-in of a PFLP client<br>Log-out of a PFLP client<br>Refused log-in of a PFLP client: The name and IP address of the client separated by a *Tabulator*.<br>Successful log-in of a TPEP client<br>Log-out of a TPEP client<br>Refused log-in of a TPEP client: The name and IP address of the client separated by a *Tabulator*. | |
| **ReportTime** | String<br>*<in>* | Contains the time when the trigger that initiated the execution of the automation script occurred. The time is expressed in seconds since January 1, 1970. | Telemetry Parameter Extractions<br>Telemetry Reports |
| **EventLog** | String<br>*<out>* | In contrast to the previous arguments this one is used to return information back which is then displayed as an event in the *Global Eventbox*.This can be achieved by assigning the appropriate text to the **EventLog** (StringBuilder) variable like, for example:<br>`EventLog.Insert(0,"Show this text inside the global eventbox.");`<br>Note:<br>The supplied text will be shown only if the automation script executed successfully. There are additional flags allowing a more complete control over how the text is displayed in the eventbox: | Telemetry Parameter Extractions<br>Telemetry Reports |

| 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.<br>Note:<br>This option only works with the *Global Eventbox*. It is ignored for other Eventbox display windows. |
| **/NOTIFICATION** | Specifies the name of an existing *Notification Profile* to be used when the message is displayed.<br>Note:<br>This option only works with the *Global Eventbox*. It is ignored for other Eventbox display windows. |

| | |
|---|---|
| **/COLOR** | Specifies the color of the message text.<br>Note:<br>The color is expressed as a 24-bit value: *RGB(red,green,blue)*. |
| **/BLINK** | Indicates if the message should blink. |

**Examples:**

```
EventLog.Insert(0,"Show this text
inside the global eventbox.");
```

is identical with:

```
EventLog.Insert(0,"/CATEGORY:'System'
/TYPE:'Informational' /MESSAGE:'Show
this text inside the global
eventbox.'");
```

Adding the '/BLINK' flag makes it blinking:

```
EventLog.Insert(0,"/CATEGORY:'System'
/TYPE:'Informational' /MESSAGE:'Show
this text inside the global
eventbox.' /BLINK");
```

Displaying the message in red color is achieved by adding the '/COLOR' flag:

```
EventLog.Insert(0,"/CATEGORY:'System'
/TYPE:'Informational' /MESSAGE:'Show
this text inside the global
eventbox.' /BLINK /COLOR:255");
```

Specifying a subtype can be useful to perform filtering:

```
EventLog.Insert(0,"/CATEGORY:'System'
/TYPE:'Informational'
/SUBTYPE:'Script100' /MESSAGE:'Show
this text inside the global
eventbox.'");
```

Note:
All text values must always be enclosed with single quotes.

## 4. Libraries

In order to minimize the code size for automation scripts it is possible to create script libraries. Such libraries consist of one or more functions that can be called from other automation scripts.
The following procedure call must be performed to call functions located within different automation scripts:

```
function LoadAndExecute(Module : String,Function : String,pArguments :
  Object[]) : Object
```

| Argument | Description |
|---|---|
| *Module* | Specifies the path name of the automation script that contains the function to be called |
| *Function* | Specifies the name of the function to be called |
| *pArguments* | Specifies an array of arguments to be supplied to the function |

Note:
Automation scripts consisting of exportable functions only (no global variables) can be marked as libraries by adding the keyword '**EXPORTS LIBRARY**' at the beginning of the code (not mandatory).
Examples:

```
var index : UInt32 = 0;
var pArguments : Object[] = new Object[3]; // Preparing the arguments to be
passed to the function in the library
pArguments[0] = String(Service);
pArguments[1] = String(Temps2);
pArguments[2] = String("NEXT_ODR");
index = LoadAndExecute("SCR10000.DLL","Return_SR",pArguments); //
Calling the automation script '10000'
```

# 5. Interfaces

Automation scripts provide access to an internal task scheduler and to the telemetry data through a set of dedicated functions.

## 5.1. Task Scheduler Functions

Automation scripts offer access to a task scheduler for telemetry reports and telemetry data extractions. This gives automation scripts the capability to schedule them programmatically. The following table summarizes all functions related to the scheduling of automation scripts:

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>function EnumTelemetryReports(<br>    Names : ArrayList,<br>    IDs : ArrayList,<br>    StartTimes : ArrayList,<br>    Intervals : ArrayList,<br>    StopTimes : ArrayList) : int</td><td>Enumerates and returns the count of telemetry reports currently scheduled.<br>Note:<br>This function enumerates only those telemetry reports that were scheduled with the function below.<br>Argument — Description<br>Names — The name of the telemetry reports.<br>IDs — The identifier of the telemetry reports.<br>StartTimes — The time when the telemetry reports are executed for the first time.<br>Intervals — The interval in 100ns ticks between the times when the telemetry reports are executed.<br>Note:<br>Is 0 when the telemetry reports run only once.<br>StopTimes — The time when the telemetry reports run for the last time.<br>Note:<br>Is 0 when the telemetry reports run only once.</td></tr>
<tr><td>function AddTelemetryReport(<br>    Name : String,<br>    StartTime : DateTime,<br>    Interval : TimeSpan,<br>    StopTime : DateTime) : uint</td><td>Adds a telemetry report to the schedule and returns its associated identifier.<br>Note:<br>The name of the telemetry report to be scheduled must exist as a profile. The enumerated scheduled telemetry reports support a precision of 1s only.<br>Argument — Description<br>Name — The name of the telemetry report.<br>StartTime — The time when the telemetry report should be executed for the first time.</td></tr>
</table>

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

| | |
|---|---|
| | *Interval* — The interval in 100ns ticks between the times when the telemetry report is executed.<br>Note:<br>Must be 0 when the telemetry report should run only once.<br>*StopTime* — The time when the telemetry report should run for the last time.<br>Note:<br>Must be 0 when the telemetry report should run only once. |
| function **DeleteTelemetryReport(***ID* : uint**)** : boolean | Deletes a telemetry report from the schedule.<br>**Argument** — **Description**<br>*ID* — Identifier of the telemetry report to remove from the schedule.<br>Note:<br>Whenever a scheduled telemetry report expires (> *StopTime*), it is automatically removed from the schedule. |
| function **EnumTelemetryDataExtractions(**<br>*Names* : ArrayList,<br>*IDs* : ArrayList,<br>*ScheduleTimes* : ArrayList,<br>*StartTimes* : ArrayList,<br>*StopTimes* : ArrayList**)** : int | Enumerates and returns the count of telemetry data extractions currently scheduled.<br>Note:<br>This function enumerates only those telemetry data extractions that were scheduled with the function below.<br>**Argument** — **Description**<br>*Names* — The name of the telemetry data extractions.<br>*IDs* — The identifier of the telemetry data extractions.<br>*ScheduleTimes* — The time when the telemetry data extractions execute.<br>*StartTimes* — The start time of the telemetry data extractions range.<br>*StopTimes* — The stop time of the telemetry data extractions range. |

<table>
<tr>
<td>function AddTelemetryDataExtraction(<br>Name : String,<br>ScheduleTime : DateTime,<br>StartTime : DateTime,<br>StopTime : DateTime) : uint</td>
<td>Adds a telemetry data extraction to the schedule and returns its associated identifier.<br>Note:<br>The name of the telemetry data extraction to be scheduled must exist as a profile. The enumerated scheduled telemetry data extractions support a precision of 1s only.
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>Name</td><td>The name of the telemetry data extraction.</td></tr>
<tr><td>ScheduleTime</td><td>The time when the telemetry data extraction should start.</td></tr>
<tr><td>StartTime</td><td>The start time of the telemetry data extraction range.</td></tr>
<tr><td>StopTime</td><td>The stop time of the telemetry data extraction range.</td></tr>
</table>
</td>
</tr>
<tr>
<td>function DeleteTelemetryDataExtraction(<br>ID : uint) : boolean</td>
<td>Deletes a telemetry data extraction from the schedule.
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>ID</td><td>Identifier of the telemetry data extraction to remove from the schedule</td></tr>
</table>
</td>
</tr>
</table>

Note:
When testing an automation script that manipulates the task scheduler, a list of all scheduled tasks is shown in the '**Debug**' pane (at the bottom) after the script has terminated. The tasks, however, are only listed but not executed at the indicated schedule time.

### 5.2. Telemetry Data Functions

Automation scripts support an interface to the telemetry data which gives them the capability to take decisions depending on the current telemetry data.
The following table summarizes all functions related to the telemetry data interface:

| Function | Description |
|---|---|
| function **GetValue(***szTag* : String,<br>*nSample* : int**)** : Object | Returns the current (calibrated) value of telemetry parameter *szTag* at occurrence *nSample* (≥ 0).<br>Note:<br>If an illegal occurrence number is specified the function returns an empty object. |

| | |
|---|---|
| function **GetRawValue(***szTag* : String, *nSample* : int**)** : Object | Returns the current raw value of telemetry parameter *szTag* at the occurrence specified by *nSample* (≥ 0).<br>Note:<br>If an illegal occurrence number is specified the function returns an empty object. |
| function **GetValueTime(***szTag* : String, *nSample* : int**)** : DateTime | Returns the time associated with the value of telemetry parameter *szTag* at the occurrence specified by *nSample* (≥ 0).<br>Note:<br>If an illegal occurrence number is specified the function returns an empty object. |
| function **GetPastValue(***szTag* : String, *nSample* : int**)** : Object | Returns a past (calibrated) value of telemetry parameter *szTag*. The variable *nSample* specifies how many samples in the past the value should be from.<br>Note:<br>If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.<br>Use the function '**SetPastValueSamples**' in order to specify the number of samples to keep. |
| function **GetPastRawValue(***szTag* : String, *nSample* : int**)** : Object | Returns a past raw value of telemetry parameter *szTag*. The variable *nSample* specifies how many samples in the past the value should be from.<br>Note:<br>If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.<br>Use the function '**SetPastValueSamples**' in order to specify the number of samples to keep. |
| function **GetPastValueTime(***szTag* : String, *nSample* : int**)** : DateTime | Returns the time associated with a past value of telemetry parameter *szTag*. The variable *nSample* specifies how many samples in the past the value should be from.<br>Note:<br>If a parameter occurs more than once within a telemetry unit, each occurrence is counted as a sample.<br>Use the function '**SetPastValueSamples**' in order to specify the number of samples to keep. |

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

| | |
|---|---|
| function **SetPastValueSamples(***szTag* : String, *wSamples* : ushort**)** : bool | Specifies the number of samples *wSamples* to keep for the specified telemetry parameter *szTag*.<br>Note:<br>This function needs to be called once only per telemetry parameter (e.g. during initialization) if the number of samples to keep remains constant. |
| function **GetPastValueSamples(***szTag* : String**)** : ushort | Returns the number of samples kept for the specified telemetry parameter *szTag*. |
| function **GetLastUpdateTime(***szTag* : String**)** : DateTime | Returns the time when the telemetry parameter *szTag* was updated the last time. |
| function **GetLastChangeTime(***szTag* : String**)** : DateTime | Returns the time when the telemetry parameter *szTag* changed its value the last time. |
| function **GetStatus(***szTag* : String,*nSample* : int**)** : uint | Returns the status of the telemetry parameter *szTag* which may be a combination of the following values:<br>TMPARAMETER_STATUS_GOOD<br>TMPARAMETER_STATUS_BAD<br>TMPARAMETER_STATUS_NOLIMIT<br>TMPARAMETER_STATUS_SOFTLIMIT<br>TMPARAMETER_STATUS_HARDLIMIT<br>TMPARAMETER_STATUS_DELTALIMIT<br>TMPARAMETER_STATUS_VALID<br>TMPARAMETER_STATUS_INVALID<br>Note:<br>The value TMPARAMETER_STATUS_NONE is returned if the telemetry parameter has no value. |
| function **Lock()** : boolean | Locks the complete telemetry interface and prevents other threads from updating any telemetry parameter. Returns a non-zero value if the function succeeded, zero if not.<br>Note:<br>Use this function with extreme care only as the telemetry data processing is blocked until '**Unlock**' is called. |
| function **Unlock()** : boolean | Unlocks the complete telemetry interface and allows the continuation of the telemetry data processing. Returns a non-zero value if the function succeeded, zero if not. |

### 5.3. Satellite Tracking, Pass & Interlink Functions

Access to satellite tracking services, location pass or satellite interlink predictions is supported by the automation scripts as well.
The following table summarizes all functions related to the satellites interface:

| Function | Description |
|---|---|
| function **CalculateSpacecraftOrbitLongitude(**<br>*Spacecraft* : String,<br>*NORADID* : uint,<br>*Time* : DateTime**)** : double | Returns the longitude of the specified spacecraft <*Spacecraft*,*NORADID*> at the time *Time*.<br>Note:<br>• This function is available for Earth-centric spacecraft only<br>• The parameter *Time* must be within an interval of a few days from current real-time in order to guarantee a precise result<br>• The returned longitude will be between 0...360 degrees<br>**Argument** — **Description**<br>*Spacecraft* — The name of spacecraft.<br>*NORADID* — The NORAD identifier of the specified spacecraft.<br>*Time* — The time for which the longitude should be calculated. |
| function **CalculateSpacecraftOrbitLatitude(**<br>*Spacecraft* : String,<br>*NORADID* : uint,<br>*Time* : DateTime**)** : double | Returns the latitude of the specified spacecraft <*Spacecraft*,*NORADID*> at the time *Time*.<br>Note:<br>• This function is available for Earth-centric spacecraft only<br>• The parameter *Time* must be within an interval of a few days from current real-time in order to guarantee a precise result<br>• The returned latitude will be between -90...90 degrees<br>**Argument** — **Description**<br>*Spacecraft* — The name of spacecraft.<br>*NORADID* — The NORAD identifier of the specified spacecraft.<br>*Time* — The time for which the latitude should be calculated. |

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

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

<table>
<tr>
<td>function CalculateSpacecraftPosition(<br>   Spacecraft : String,<br>   NORADID : uint,<br>   Time : DateTime) : CSatellitePosition</td>
<td>Returns the position (relative to the Sun) of the specified spacecraft <<i>Spacecraft</i>,<i>NORADID</i>> at the time <i>Time</i>.<br>Note:<br>• For Earth-centric spacecraft (<i>NORADID</i> <> 0) the parameter <i>Time</i> must be within an interval of a few days from current real-time in order to guarantee a precise result<br>• The returned position will be returned in form of the class 'CSpacecraftPosition'; its members ‘x’, ‘y’, ‘z’ contain the position coordinates in km
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>Spacecraft</td><td>The name of spacecraft.</td></tr>
<tr><td>NORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td>Time</td><td>The time for which the position (relative to the Sun) should be calculated.</td></tr>
</table>
</td>
</tr>
<tr>
<td>function CalculateSpacecraftVelocity(<br>   Spacecraft : String,<br>   NORADID : uint,<br>   Time : DateTime) : CSatelliteVelocity</td>
<td>Returns the velocity (relative to the Sun) of the specified spacecraft <<i>Spacecraft</i>,<i>NORADID</i>> at the time <i>Time</i>.<br>Note:<br>• For Earth-centric spacecraft (<i>NORADID</i> <> 0) the parameter <i>Time</i> must be within an interval of a few days from current real-time in order to guarantee a precise result<br>• The returned velocity will be returned in form of the class 'CSpacecraftVelocity'; its members ‘x’, ‘y’, ‘z’ contain the velocity coordinates in km/s
<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>Spacecraft</td><td>The name of spacecraft.</td></tr>
<tr><td>NORADID</td><td>The NORAD identifier of the specified spacecraft.</td></tr>
<tr><td>Time</td><td>The time for which the velocity (relative to the Sun) should be calculated.</td></tr>
</table>
</td>
</tr>
</table>

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

function **CalculateSpacecraftPassStartTime(**
*Spacecraft* : String,
*NORADID* : uint,
*Location* : String,
*LocationLongitude* : double,
*LocationLatitude* : double,
*LocationAltitude* : double,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the begin of the next pass over the location <*Location*,*LocationLongitude*, *LocationLatitude*,*LocationAltitude*> of the specified spacecraft <*Spacecraft*,*NORADID*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

Note:

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

| Argument | Description |
|---|---|
| *Spacecraft* | The name of spacecraft. |
| *NORADID* | The NORAD identifier of the specified spacecraft. |
| *Location* | The name of pass-over location. |
| *LocationLongitude* | The longitude (deg) of the pass-over location. |
| *LocationLatitude* | The latitude (deg) of the pass-over location. |
| *LocationAltitude* | The altitude (km) of the pass-over location. |
| *StartTime* | Specifies the start time to be used to calculate the next pass over the specified location. |
| *TimeInterval* | Specifies the interval to be used to calculate the next pass over the specified location. |

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

function **CalculateSpacecraftPassStopTime(**
*Spacecraft* : String,
*NORADID* : uint,
*Location* : String,
*LocationLongitude* : double,
*LocationLatitude* : double,
*LocationAltitude* : double,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the end of the next pass over the location <*Location*,*LocationLongitude*, *LocationLatitude*,*LocationAltitude*> of the specified spacecraft <*Spacecraft*,*NORADID*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

Note:

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

| Argument | Description |
|---|---|
| *Spacecraft* | The name of spacecraft. |
| *NORADID* | The NORAD identifier of the specified spacecraft. |
| *Location* | The name of pass-over location. |
| *LocationLongitude* | The longitude (deg) of the pass-over location. |
| *LocationLatitude* | The latitude (deg) of the pass-over location. |
| *LocationAltitude* | The altitude (km) of the pass-over location. |
| *StartTime* | Specifies the start time to be used to calculate the next pass over the specified location. |
| *TimeInterval* | Specifies the interval to be used to calculate the next pass over the specified location. |

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

function
**CalculateSpacecraftInterlinkStartTime(**
*SpacecraftA* : String,
*NORADIDA* : uint,
*SpacecraftB* : String,
*NORADIDB* : uint,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the begin of the next interlink session between the spacecraft <*SpacecraftA*, *NORADIDA*> and <*SpacecraftB*,*NORADIDB*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

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

| Argument | Description |
|---|---|
| *SpacecraftA* | The name of first spacecraft. |
| *NORADIDA* | The NORAD identifier of the first spacecraft. |
| *SpacecraftB* | The name of second spacecraft. |
| *NORADIDB* | The NORAD identifier of the second spacecraft. |
| *StartTime* | Specifies the start time to be used to calculate the next interlink session. |
| *TimeInterval* | Specifies the interval to be used to calculate the next interlink session. |

function
**CalculateSpacecraftInterlinkStopTime(**
*SpacecraftA* : String,
*NORADIDA* : uint,
*SpacecraftB* : String,
*NORADIDB* : uint,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the end of the next interlink session between the spacecraft <*SpacecraftA*, *NORADIDA*> and <*SpacecraftB*,*NORADIDB*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

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

| Argument | Description |
|---|---|
| *SpacecraftA* | The name of first spacecraft. |
| *NORADIDA* | The NORAD identifier of the first spacecraft. |
| *SpacecraftB* | The name of second spacecraft. |
| *NORADIDB* | The NORAD identifier of the second spacecraft. |
| *StartTime* | Specifies the start time to be used to calculate the next interlink session. |
| *TimeInterval* | 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

function
**CalculateSpacecraftInterlinkStartTime(**
*SpacecraftA* : String,
*NORADIDA* : uint,
*SpacecraftVia* : String,
*NORADIDVia* : uint,
*SpacecraftB* : String,
*NORADIDB* : uint,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the begin of the next interlink session between the spacecraft <*SpacecraftA*, *NORADIDA*> and <*SpacecraftB*,*NORADIDB*> via the relais <*SpacecraftVia*,*NORADIDVia*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

Note:

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

| Argument | Description |
|---|---|
| *SpacecraftA* | The name of first spacecraft. |
| *NORADIDA* | The NORAD identifier of the first spacecraft. |
| *SpacecraftVia* | The name of relais spacecraft. |
| *NORADIDVia* | The NORAD identifier of the relais spacecraft. |
| *SpacecraftB* | The name of second spacecraft. |
| *NORADIDB* | The NORAD identifier of the second spacecraft. |
| *StartTime* | Specifies the start time to be used to calculate the next interlink session. |
| *TimeInterval* | 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

function
**CalculateSpacecraftInterlinkStopTime(**
*SpacecraftA* : String,
*NORADIDA* : uint,
*SpacecraftVia* : String,
*NORADIDVia* : uint,
*SpacecraftB* : String,
*NORADIDB* : uint,
*StartTime* : DateTime,
*TimeInterval* : TimeSpan**)** : DateTime

Returns the end of the next interlink session between the spacecraft <*SpacecraftA*, *NORADIDA*> and <*SpacecraftB*,*NORADIDB*> via the relais <*SpacecraftVia*,*NORADIDVia*> after the time *StartTime* and within the subsequent *TimeInterval* interval.

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

| Argument | Description |
|---|---|
| *SpacecraftA* | The name of first spacecraft. |
| *NORADIDA* | The NORAD identifier of the first spacecraft. |
| *SpacecraftVia* | The name of relais spacecraft. |
| *NORADIDVia* | The NORAD identifier of the relais spacecraft. |
| *SpacecraftB* | The name of second spacecraft. |
| *NORADIDB* | The NORAD identifier of the second spacecraft. |
| *StartTime* | Specifies the start time to be used to calculate the next interlink session. |
| *TimeInterval* | 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 'DateTime' 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

## 6. Samples

The following sample illustrates how Microsoft® Excel® can be automated:

```
// Declare the variables
var Excel, Book, Sheet;

// Create the Excel application object
Excel = new ActiveXObject("Excel.Application");

// Hide the application
Excel.Visible = false;

// Hide the assistant
Excel.Assistant.Visible = false;

// Create a new work book
Book = Excel.Workbooks.Add();

try
{   // Configure the workbook
    Sheet = Book.WorkSheets(1);
    Sheet.Activate();
    Sheet.Name = "Pulscount Times";
    Book.WorkSheets("Sheet2").Delete();
    Book.WorkSheets("Sheet3").Delete();

    // Format the rows and columns
    Sheet.Columns(1).NumberFormat = "hh:mm:ss";
    Sheet.Columns(1).Interior.ColorIndex = 22;
    Sheet.Rows(1).NumberFormat = "General";
    Sheet.Rows(1).Font.Bold = true;

    // Read the source data
    var Pos : int;
    var Col : int;
    var Row : int;
    var Index : int;
    var Source : String;
    var Stream : StreamReader;

    Stream = File.OpenText(InputFile);
    for (Row = 1; true; Row = Row+1)
    {    if ((Source = Stream.ReadLine()) != null)
         {    for (Col = 1,Index = 0; Index < Source.Length; Col = Col+1)
              {    if ((Pos = Source.Substring(Index).IndexOf(",")) ≥ 0)
                   {    Sheet.Cells(Row,Col).Value =
Source.Substring(Index,Pos);
                        Index = Index+Pos+1;
                        continue;
```

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

```
                    }
                    Sheet.Cells(Row,Col).Value = Source.Substring(Index);
                    break;
               }
               continue;
          }
          break;
     }
     Stream.Close();

     // Add a embedded charts
     var Graph1, Graph2, Graph3;
     Graph1 = Sheet.ChartObjects().Add(200,20,500,250);
     Graph2 = Sheet.ChartObjects().Add(200,290,500,250);
     Graph3 = Sheet.ChartObjects().Add(200,560,500,250);
     with (Graph1.Chart)
     {    SeriesCollection.Add(Sheet.Range("A2",Sheet.Range("B2").End(4)));
          SeriesCollection(1).Name = "V191";
          ChartArea.Interior.Color = 14606046;
          HasTitle = true;
          ChartTitle.Text = "RW1 Pulscount Time";
          ChartTitle.Font.Bold = true;
          ChartTitle.Font.Size = 12;
          Axes(1).HasTitle = true;
          Axes(1).AxisTitle.Caption = "TIME";
          Axes(1).AxisTitle.Font.Bold = false;
          Axes(1).TickMarkSpacing = Row/15;
          Axes(1).TickLabelSpacing = Row/15;
          Axes(1).TickLabels.Font.Size = 8;
          ChartType = 1;  // Area
     }
     with (Graph2.Chart)
     {    var DataRange2 : String;
          DataRange2 = "A2:A"+Row+",C2:C"+Row;
          SeriesCollection.Add(Sheet.Range(DataRange2));
          SeriesCollection(1).Name = "V193";
          SeriesCollection(1).Border.Weight = 3;
          SeriesCollection(1).Border.Color = 16711680;
          ChartArea.Interior.Color = 14606046;
          HasTitle = true;
          ChartTitle.Text = "RW2 Pulscount Time";
          ChartTitle.Font.Bold = true;
          ChartTitle.Font.Size = 12;
          Axes(1).HasTitle = true;
          Axes(1).AxisTitle.Caption = "TIME";
          Axes(1).AxisTitle.Font.Bold = false;
          Axes(1).TickMarkSpacing = Row/15;
          Axes(1).TickLabelSpacing = Row/15;
          Axes(1).TickLabels.Font.Size = 8;
```

```
          ChartType = 4;  // Line
     }
     with (Graph3.Chart)
     {    var DataRange3 : String;
          var DataRange4 : String;
          DataRange3 = "A2:A"+Row+",C2:C"+Row;
          DataRange4 = "B2:B"+Row;
          SeriesCollection.Add(Sheet.Range(DataRange3));
          SeriesCollection.Add(Sheet.Range(DataRange4));
          SeriesCollection(1).Name = "V193";
          SeriesCollection(2).Name = "V191";
          ChartArea.Interior.Color = 14606046;
          HasTitle = true;
          ChartTitle.Text = "RW1+2 Pulscount Times";
          ChartTitle.Font.Bold = true;
          ChartTitle.Font.Size = 12;
          Axes(1).HasTitle = true;
          Axes(1).AxisTitle.Caption = "TIME";
          Axes(1).AxisTitle.Font.Bold = false;
          Axes(1).TickMarkSpacing = Row/15;
          Axes(1).TickLabelSpacing = Row/15;
          Axes(1).TickLabels.Font.Size = 8;
          Axes(2).MinimumScale = 0.107;
          Axes(2).MaximumScale = 0.125;
          ChartType = 1;  // Area
     }

     // Save the work book
     var Ext : int;
     if ((Ext = InputFile.LastIndexOf(".")) ≥ 0)
     {    OutputFile.Remove(0,OutputFile.Length);
          OutputFile.Insert(0,InputFile.Substring(0,Ext)+".xls");  // Name of the
generated output file
          File.Delete(OutputFile.ToString());  // Prevent prompting for an
override
          Book.SaveAs(OutputFile.ToString());  // Save the generated file
     }

     // Close the current work book
     Book.Close(false);
     // Close all other work books
     Excel.Workbooks.Close();
     // Close the Excel object
     Excel.Application.Quit();
}
catch (exception)
{
     // Provide an error reason
     Error.Insert(0,exception.message);
```

```
    // Close the current work book
    Book.Close(false);
    // Close all other work books
    Excel.Workbooks.Close();
    // Close the Excel object
    Excel.Application.Quit();
}
```

This additional sample illustrates how to schedule a telemetry report:

```
// Declare the variables
var ID : uint;
var Time : Date = new Date();
var StartTime : DateTime = new
DateTime(Time.getFullYear(),Time.getMonth()+1,Time.getDate());
var StopTime : DateTime = new
DateTime(Time.getFullYear(),Time.getMonth()+1,Time.getDate());
var Interval : TimeSpan = new TimeSpan(1,0,0);

try
{
    // Executes the telemetry report for one day at an interval of one hour
    // The name 'Optical Report' must exist as a telemetry report profile
    ID = AddTelemetryReport("Optical
Report",StartTime.AddHours(1),Interval,StopTime.AddHours(24));
}
catch (exception)
{
    // Provide an error reason
    Error.Insert(0,exception.message);
}
```

The following sample shows how to use the satellite tracking, pass & interlink functions:

```
var f : StreamWriter = new StreamWriter(OutputFile, true);

var fLongitude : double;
var fLatitude : double;
var fAltitude : double;
var fVelocity : double;
var t : DateTime;
var tPassStartTime : DateTime;
var tPassStopTime : DateTime;
var tInterlinkStartTime : DateTime;
var tInterlinkStopTime : DateTime;

var cPosition : CSatellitePosition;
var cVelocity : CSatelliteVelocity;

try
{
```

```
    fLongitude = CalculateSpacecraftOrbitLongitude("ISS (ZAYRA)", 25544, (t =
DateTime.UtcNow).ToUniversalTime());
    fLatitude = CalculateSpacecraftOrbitLatitude("ISS (ZAYRA)", 25544,
t.ToUniversalTime());
    fAltitude = CalculateSpacecraftOrbitAltitude("ISS (ZAYRA)", 25544,
t.ToUniversalTime());
    fVelocity = CalculateSpacecraftOrbitVelocity("ISS (ZAYRA)", 25544,
t.ToUniversalTime());

    cPosition = CalculateSpacecraftPosition("ISS (ZAYRA)", 25544,
t.ToUniversalTime());
    cVelocity = CalculateSpacecraftVelocity("ISS (ZAYRA)", 25544,
t.ToUniversalTime());

    tPassStartTime = CalculateSpacecraftPassStartTime("ISS (ZAYRA)", 25544,
"Zurich", 8.5500025, 47.367347, 0.425, t.ToUniversalTime(),
TimeSpan.FromTicks(86400 * 10 * 1000000));
    tPassStopTime = CalculateSpacecraftPassStopTime("ISS (ZAYRA)", 25544,
"Zurich", 8.5500025, 47.367347, 0.425, t.ToUniversalTime(),
TimeSpan.FromTicks(86400 * 10 * 1000000));

    tInterlinkStartTime = CalculateSpacecraftInterlinkStartTime("ISS (ZAYRA)",
25544, "ARTEMIS", 26863,"Hubble", 20580, t.ToUniversalTime(),
TimeSpan.FromTicks(86400 * 10 * 1000000));
    tInterlinkStopTime = CalculateSpacecraftInterlinkStopTime("ISS (ZAYRA)",
25544, "ARTEMIS", 26863,"Hubble", 20580, t.ToUniversalTime(),
TimeSpan.FromTicks(86400 * 10 * 1000000));

    f.AutoFlush = true;
    f.WriteLine("ISS (ZAYRA) - Orbit Data at " + t.ToLongDateString() + " " +
t.ToLongTimeString());
    f.WriteLine("Longitude: " + fLongitude + " deg" + ", Latitude: " + fLatitude
+ " deg" + ", Altitude: " + fAltitude + " km" + ", Velocity: " + fVelocity + "
km/s" );
    f.WriteLine("Sun distance: " + cPosition.Distance() + " km, Sun velocity: "
+ cVelocity.Speed() + " km/s");
    f.WriteLine("Next pass over Zurich: " + tPassStartTime.ToLongDateString() +
" " + tPassStartTime.ToLongTimeString() + " to " +
tPassStopTime.ToLongDateString() + " " + tPassStopTime.ToLongTimeString());
    f.WriteLine("Next interlink between ISS-ARTEMIS-Hubble: " +
tInterlinkStartTime.ToLongDateString() + " " +
tInterlinkStartTime.ToLongTimeString() + " to " +
tInterlinkStopTime.ToLongDateString() + " " +
tInterlinkStopTime.ToLongTimeString());
}
catch (exception)
{
    // Provide an error reason
```

```
    Error.Insert(0,exception.message);
}
```

## 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