



**ENOVIA SmarTeam**

**Customizing Using  
Client-Side Hooks for  
Client-Based Applications  
Programmer's Guide**



© Dassault Systèmes, 1997, 2010. All rights reserved.

CATIA, ENOVIA, SmarTeam and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries in the US and/or other countries.

**PROPRIETARY RIGHTS NOTICE:** This documentation is the property of Dassault Systèmes. This documentation shall be treated as confidential information and may only be used by employees or contractors of the Customer in accordance with the terms of the End-User License Agreement accepted by Customer.

Any use of the Licensed Program contained in this media or accompanying it, is subject to the terms of the End User License Agreement accepted by Customer. The Licensed Program is protected by international copyright laws and international treaties. Unauthorized use, reproduction and/or distribution of any of the Licensed Program, or any part thereof, may result in severe civil and/or criminal penalties, and will be prosecuted to the maximum extent possible under the law. Company names and product names mentioned herein are the property of their respective owners and certain portions of the Licensed Program contain elements subject to copyright owned by these entities. See the Documentation CD provided with the Licensed Program for details and/or additional terms and conditions relating to these entities.

Part Number: API-A1-200210

---

## Table of Contents

1.	Introduction .....	1
	Adding Functionality to SmarTeam .....	1
	Understanding Script Hooks .....	3
	Using Script Hooks in Applications .....	4
2.	Writing a Script .....	7
	Script File Location .....	7
	Generic Script Function Arguments .....	7
	Error Codes .....	8
	Using COM API Functions in a Script .....	9
	Converting Procedural Parameters .....	9
	Getting the Current Session .....	9
	Programming Tips .....	10
	Handling Unlimited Size Data Types .....	10
	Using Commit Operations in a Script .....	11
	Verifying Input Attributes .....	11
3.	Passing Object Information to Script Functions .....	13
	Passing Information by Record List Parameters .....	13
	Representing SmarTeam Object Attributes .....	15
	Object Attributes Input to a Script .....	17
	Outputting Object Attributes from a Script .....	18
	Passing Link Information to Script Functions .....	20
	Representing SmarTeam Links .....	20
	Link Object Attributes Input to a Script .....	21
	Getting Class Attributes in the Data Model .....	23
4.	Scripts for Profile Card Events .....	24
	Profile Card Script Hooks .....	25
	Attaching a Script to a Profile Card Script Hook .....	25
	Script Execution Timing .....	28
	Screen Startup .....	29
	Script Hook Parameters .....	30
	Screen Exit .....	31
	Script Hook Parameters .....	31
	OnEnter .....	32
	Script Hook Parameters .....	33
	OnExit .....	33
	Script Hook Parameters .....	34
	OnClick .....	34

---

Script Hook Parameters .....	35
CALL_SCRIPT .....	35
Script Hook Parameters .....	37
Script Execution Timing .....	37
Examples .....	38
5. Scripts for SmarTeam Operations .....	41
Script Hooks Available for Operations .....	41
Attaching a Script to a Script Hook .....	43
Scripts for Object Database Operations .....	45
Add, AddAsCopy .....	45
Script Execution Timing .....	45
Script Hook Parameters .....	47
Examples .....	49
Update .....	53
Script Execution Timing .....	53
Script Hook Parameters .....	54
Examples .....	56
Delete .....	56
Script Execution Timing .....	57
Script Hook Parameters .....	57
Examples .....	59
Scripts for Lifecycle Operations .....	62
Overview of Lifecycle Script Hooks .....	62
Lifecycle Operation Sequence .....	64
Timing of Life-Cycle Script Hook Events .....	65
Individual Lifecycle Task Attributes .....	68
Applicable Hooks .....	68
NM_OBJECT_ID .....	69
NM_CLASS_ID .....	69
NM_OPER_ID .....	70
NM_EFFECTIVE_FROM .....	70
NM_EFFECTIVE_UNTIL .....	71
NM_FILE_NAME .....	71
NM_DIRECTORY .....	72
NM_VAULT_OBJ_ID .....	72
NM_REVISION .....	73
NM_DSC_NOTES .....	73
NM_PHASE .....	74
NM_TSK_KEEP_LOCAL_COPY .....	75
NM_TSK_KEEP_CHECKEDOUT .....	75
NM_TSK_NOCREATE_LOCAL_COPY .....	76

NM_LFCYC_NEW_BRANCH.....	77
NM_LFCYC_CHECKIN_MODE.....	77
NM_LOGICAL_LINK_COPY.....	79
NM_LINKS_TO_SONS_COPY.....	80
NM_FILE_OVERWRITE.....	81
NM_NOT_CHECK_AUTH.....	82
Passing Lifecycle Task Information to Script Functions.....	82
Representing SmarTeam Tasks.....	83
Passing Default Task Attributes.....	85
Scripts for Individual GUI-Based Lifecycle Operations.....	90
Load Lifecycle Screen.....	90
Script Execution Timing.....	91
Script Hook Parameters.....	91
Click LifeCycleOperation.....	94
Script Execution Timing.....	94
Script Hook Parameters.....	95
Examples.....	96
Scripts for Individual Non-GUI Lifecycle Operations.....	98
Script Execution Timing.....	98
Script Hook Parameters.....	100
Examples.....	103
Scripts for Group Lifecycle Operations.....	104
Group Lifecycle Task Attributes.....	105
NM_REPLACE_TO_LATEST_AVLBL.....	105
NM_TDM_GET_LATEST_AVLBL_CHILD.....	106
NM_ASK_FILE_NAME_NOT_UNIQUE.....	107
NM_MULTIPLE_REVISION_TREAT.....	108
NM_NO_ASK_CHILD_OPER_INCONSISTENT.....	108
Tree Filter Parameters.....	110
NM_RevisionFilter.....	111
NM_FromDate.....	112
NM_UntilDate.....	113
NM_AllowOverLap.....	113
Passing Life-Cycle Association Information to Script Functions.....	114
Representing SmarTeam Associations.....	114
Lifecycle Stage 1, 2 Hooks.....	118
Script Execution Timing.....	118
Script Hook Parameters.....	119
Examples.....	122
Scripts for File Operations.....	126
Script Execution Timing.....	127

Script Hook Parameters .....	128
Examples .....	130
Scripts for Authorization Operations .....	132
OnLogin .....	132
Script Execution Timing .....	132
OnLogin Hook Attributes.....	132
NM_LOGIN .....	132
NM_PASSWORD.....	133
Script Hook Parameters.....	133
Examples .....	134
Single Sign-On .....	137
OnBrowse.....	137
Script Execution Timing .....	137
Script Hook Parameters.....	137
Examples .....	138
OnRetrieveObjects .....	139
Script Execution Timing .....	140
OnRetrieveObject Hook Attributes .....	140
NM_OPER_NAME .....	140
VIEW_COL_LEAD_CLASS_ID .....	141
VIEW_COL_LEAD_OBJ_ID .....	141
NM_CLASS_ID.....	141
NM_CHECK_AUTHORIZATION_MODE .....	142
NM_ AUTHORIZED_OBJ.....	142
Script Hook Parameters.....	144
Examples .....	145
Scripts for CAD Identification .....	147
CAD Integration Script Hooks.....	147
CAD Integration.....	147
Script Execution Timing .....	147
Script Hook Parameters.....	148
Scripts in Flow Processes .....	148
OnOpen Hook .....	148
Before Send Hook .....	149
6. Scripts for Import/Export Operations.....	151
Virtual Attributes.....	151
Import/Export Script Hooks .....	151
Attaching a Script to an Import/Export Script Hook.....	152
OnImport, OnExport .....	152
Script Execution Timing .....	152
Script Hook Parameters.....	153

Examples .....	154
7. Scripts for User-Defined Commands .....	155
Attaching a Script to User-Defined Operations .....	155
User-Defined Command Script Hook .....	155
Script Execution Timing .....	155
Script Hook Parameters .....	156
Examples .....	156
Appendix A Attributes Passed by SmarTeam – Editor .....	161





# 1. Introduction

This document describes how to use procedural API script interfaces to add functionality to SmarTeam.

This document describes:

- Script hooks, where you can attach a script to the SmarTeam software
- The arguments passed for each type of script hook
- The events that trigger the script hooks
- The timing sequence of the script hooks
- Recommendations on how to write a script

This document describes *generic* script hooks for the procedural API interface. These script hooks are called generic because they use the same parameter set for all hooks.

Starting with SmarTeam 3.0, a new COM API interface was implemented for script hooks in SmarTeam libraries such as Smart Flow and ERP. These script hooks are called *library-specific* hooks; their parameter set is not fixed, but depends on the specific hook. For more information about the COM API interface and the library-specific hooks, see the appropriate chapter in the COM API Programmer's Guide. A COM API interface was also implemented for working with SmarTeam hooks in Server Mode (see *Customizing Using Server-Side Hooks for Server-Based Applications*).

While generic script hooks were originally designed to be used with the procedural API, they can be used with the COM API as well. It is highly recommended that you use the COM API to write scripts.

## ***Adding Functionality to SmarTeam***

You can enhance the functionality of basic SmarTeam operations by attaching scripts to script hooks. You can determine the exact functionality to be added and when that functionality will execute.

You determine the functionality by using SmarTeam's API to write a script that performs the required actions.

SmarTeam provides two ways that you can determine when that script will execute:

Attaching a script to an existing SmarTeam operation

Attaching a script to a user-defined command

## **Scripts for Operations**

One way is to attach the script to a script hook that is associated with an existing SmarTeam operation. Then, when the operation is performed, the script executes.

Script hooks are provided for the following types of SmarTeam operations:

Profile Card events—when viewing Profile Cards and performing database actions on objects represented by Profile Cards, see *Scripts for Profile Card Events*.

Object database operations—operations on objects in the database such as Add, Update and Delete, see *Scripts for Object Database Operations*.

Lifecycle operations—operations such as Check In and Check Out, which pertain to the life cycle of an object, see *Scripts for Lifecycle Operations*.

File operations—operations on files associated with an object, such as Edit, View and Print, see *Scripts for File Operations*.

Authorization operations—imposing authorization requirements on user actions, such as logging in, browsing objects, and retrieving objects, see *Scripts for Authorization Operations*.

Import/Export operations—operations pertaining to exporting objects from the database and import objects into the database, see *Scripts for Import/Export Operations*.

## **Scripts for User-Defined Commands**

Alternatively, you can attach a script to a special menu dedicated for that purpose; you execute the script by selecting that menu item. In that case, the script is called a *user-defined operation*. See *Scripts for User-Defined Commands*.

## Understanding Script Hooks

Script hooks are “locations” in the SmarTeam software where the software can be set up to execute a user-supplied script and pass parameters to it. Script hooks are provided for most standard SmarTeam operations. Three different stages of a SmarTeam operation are defined for script hooks: before the operation executes, after the operation executes or instead of the operation’s execution. These three operation stages are represented by the software events: Before, After, and InsteadOf operation. Thus, you specify a script hook uniquely by the SmarTeam operation and the stage with which it is associated. Accordingly, in this guide, it is convenient to refer to a script hook by a name in the form *EventName OperationName*, for example, Before Update, After Check In or InsteadOf Add. Note that not all operations have script hooks defined for all three of the operation stages.

The purpose of the script is to allow you to intervene and modify the action of the standard SmarTeam operation. For example, if a script is attached to the Before Update script hook, the script is executed by SmarTeam before SmarTeam actually updates the attributes of the object in the database. Here, the script can change some of the attributes before they are written in the database. If a script is attached to the After Update script hook, it is executed after the attributes are already updated in the database. Here, the script might send notification of the successful update.

Figure 1 illustrates the Before Update hook for the Update Operation

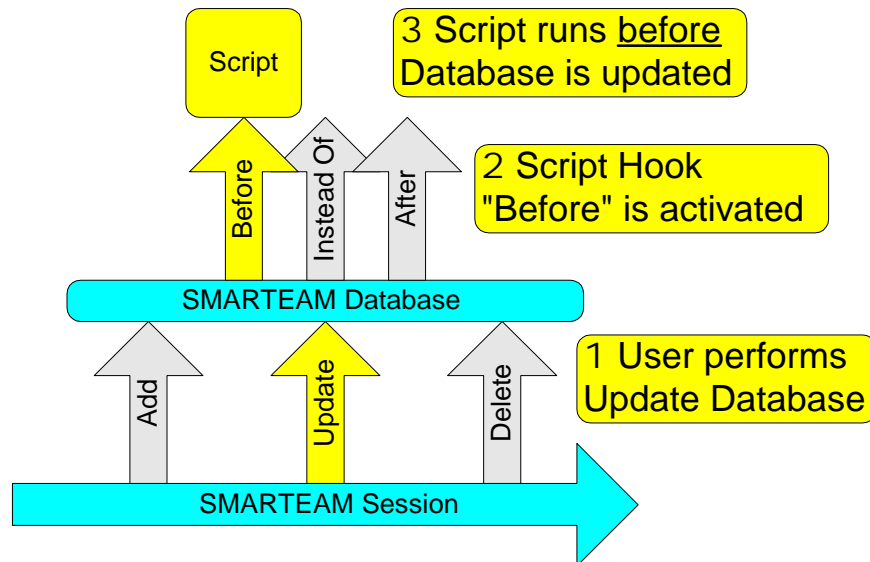


Figure 1 Before Update Script Hook Operation

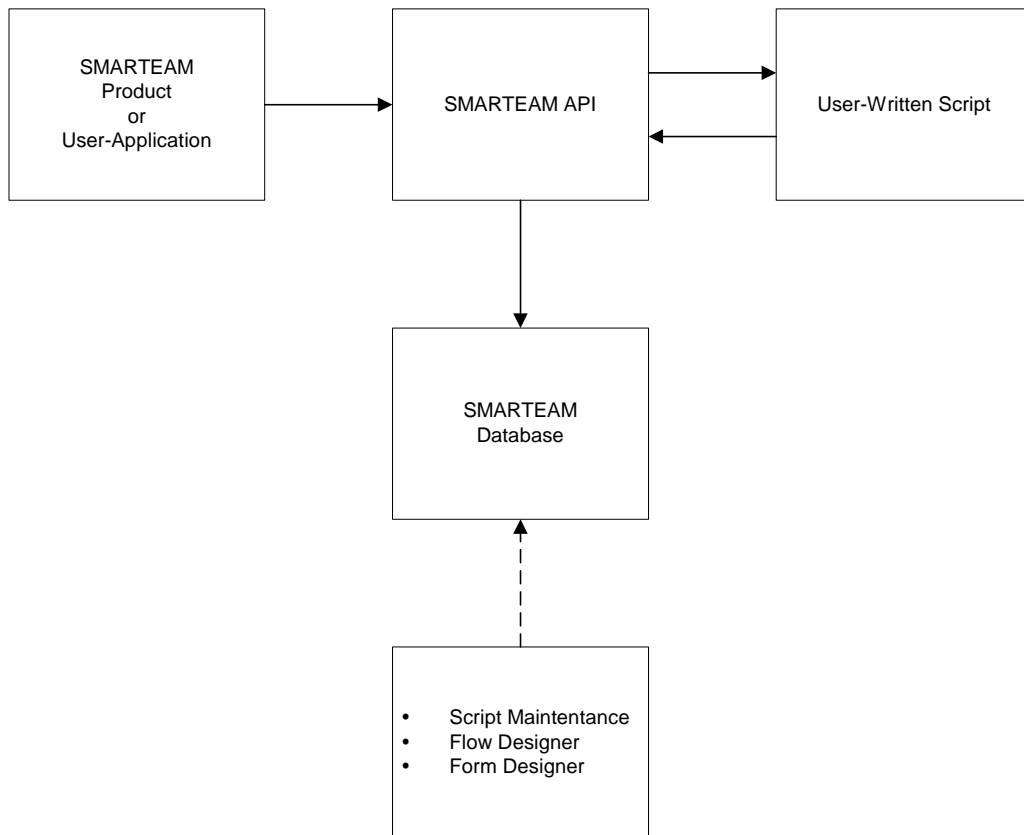
## ***Using Script Hooks in Applications***

Script hooks have wide applicability and can be used in user-written applications just as they are used in the SmarTeam family of products.

Due to the modular nature of SmarTeam, applications can be written based on the SmarTeam API. Script hooks can be used in these applications. You attach a script to a hook using the Script Maintenance facility or one of the library designer utilities. Then, when the appropriate API function is executed, the script attached to the hook executes. For example, if you attach a script to the Before Add hook in Script Maintenance, then when the corresponding API function (ISmObject.Insert) is executed by the application, the script is executed just before Insert inserts the object into the database.

You can disable the execution of an attached script by the SmBehavior object (see Chapter 5 of the Programmer's Guide.)

Figure 2 shows the relationship between an application and the user-written script. It shows that the mechanism of script hooks is connected to the SmarTeam API and is separate from the application itself. Thus it applies equally to any application.



*Figure 2 Script Hooks in SmarTeam Applications*



## 2. Writing a Script

This chapter discusses how to write a script and includes the following topics:

Script file location

Generic script function arguments

Using COM API Functions in a script

Programming tips

### ***Script File Location***

Script files should be located under the `script` directory (for example: `c:\SmarTeam\script`). The name of the script directory is defined in the System Configuration under the Key Name `Directory_Structure.ScriptDirectory`. The Configuration Set is `SmarTeam.std.legacypreferences.config`.

The script function name cannot be longer than 31 characters and the file name is limited to 63 characters.

The scripts should be written in Basic using the Smart BasicScript editor. However, it is important to note that the script can create and use COM objects written using any COM-enabled language or development tool.

### ***Generic Script Function Arguments***

In order to be a valid script function, a function must be defined with the following generic argument format:

```
Function Temp (ApplHndl As Long,  
Operation As String,  
FirstPar As Long,  
SecondPar As Long,  
ThirdPar As Long) As Integer
```

The arguments are described in general in Table 1.

The exact meaning of each argument depends on the particular script hook to which the script is attached. To verify the meaning of the arguments you need for a particular script, refer to the description of the script hook to which you are attaching your script.

*Table 1 Arguments in a Script Function*

Argument	Input/Output	Description
ApplHndl	Input	Converts to SmSession using the function SCREXT_ObjectForInterface
Operation	Input	A string parameter that contains the name of the standard SmarTeam operation of the script hook. The script must treat the operation parameter as case-insensitive.
FirstPar	Input	A pointer to a record list containing object information received from SmarTeam.
SecondPar	Input / Output	A pointer to a record list that contains object, task, or link information.
ThirdPar	Input/Output	A pointer to a record list containing object or task information to be returned to SmarTeam.
Return value	Output	One of the error-code constants in Table 2. You should always assign a return value.

## **Error Codes**

Table 2 describes the error codes you can use in a script.

*Table 2 Error Codes*

Error Code	Value	Description
Error_None	0	No error
Err_Gen	1	General software error
Err_Dup	2	Duplicate record
Err_NotFound	3	Record not found
Err_DeadLock	4	Database Deadlock
Err_Refuse	6	Refuse for the operation
Err_NotAuth	11	Not authorized access



## Using COM API Functions in a Script

Using COM API functions in a script allows you to use the new SmarTeam API. This section shows how to convert procedural record list arguments to their COM representation, and how to get the current Session object.

**Note:** It is recommended to use the newer SmarTeam API rather than the old procedural API; many features are available only through the COM API functions, and all new functionality is added only to the COM API.

### Converting Procedural Parameters

This section describes how to use COM API even if you have a procedural type of script function interface. In order to use COM API functions, you need to convert the procedural record list arguments to their COM representation using the conversion procedures provided. In addition you need to get the current Session object.

The general flow of the script should be:

Use the `CONV_RecListToComRecordList` procedure to convert the `FirstPar`, `SecondPar` and `ThirdPar` procedural record list parameters to COM representation.

Get the current Session

Perform the desired script operations using COM API functions on the COM representation of the record lists `SecondPar` and `ThirdPar`.

Before ending the script, convert the COM representation of `SecondPar` and `ThirdPar` back to procedural representation using function `ComRecListToRecordList`.

### Getting the Current Session

To get the current session use the following:

```
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
```

#### Example

```
*****
' Template for writing scripts
' ApplHndl - application handle
' FirstPar - record list containing attributes of the selected object(s)
' SecondPar - record list used for transfer of service data between
' script and application
' ThirdPar - record list to transfer object data back to the application
*****
```

```
Declare Sub CONV_RecListToComRecordList Lib "SmTdm32" (ByVal RecList As Long, ByRef COMRecList As ISmRecordList)
Declare Sub CONV_ComRecListToRecordList Lib "SmTdm32" (ByVal ComRecList As ISmRecordList, ByRef RecList As Long)
Function TemplateFunc (ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim COMFirstList As SmRecList.SmRecordList
    Dim COMSecondList As SmRecList.SmRecordList
    Dim COMThirdList As SmRecList.SmRecordList
    ' Get Session from Application handle
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert Record Lists to COM representation
    CONV_RecListToComRecordList FirstPar,COMFirstList
    CONV_RecListToComRecordList SecondPar,COMSecondList
    CONV_RecListToComRecordList ThirdPar,COMThirdList
    ...
    <Body of the script>
    ' Convert COM Record List to procedural record list
    ' to return to main application
    CONV_ComRecListToRecordList COMThirdList,ThirdPar
End Function
```

## ***Programming Tips***

This section contains some programming tips for the script writer. The following topics are included:

- Handling unlimited size data types
- Using Commit operations in a script
- Verifying input attributes
- Function Names in Scripts

### ***Handling Unlimited Size Data Types***

The Memo (character) and Blob (binary) data types have an unlimited data size. Thus when stored in the database they can occupy an unlimited space, subject to the resources available. However, when loading an object that has a Memo or Blob attribute into a record list -- which has a necessarily limited cell size -- the attribute contents can be truncated. For example, a Memo data item that has 400 characters in the database would appear in the record list with only the first 256 characters.

Because of this truncation, you should not save the record list back into the database (for example, using the `ObjectFromData` function to create an object from the record list and update the database). If you do so, you will likely lose part of the information.

### ***Using Commit Operations in a Script***

Avoid using the explicit “commit” database methods, in a procedural API script. This is because your script may be part of a larger database transaction in SmarTeam such that terminating the database operation at this point would make rollback impossible should the transaction fail.

### ***Verifying Input Attributes***

It is recommended to verify whether a specific attribute you need already exists in the input record list and not to automatically retrieve it from the database.

### ***Function Names in Scripts***

When the same function is used in different scripts which are attached by hooks, incorrect results occur. To ensure that scripts are launched correctly, provide different function names in the scripts.



## **3. Passing Object Information to Script Functions**

This chapter describes how to pass object information to and from the script by means of record list parameters. The information in this chapter serves as a basis for discussing the individual hooks in the following chapters.

The topics discussed in this chapter are:

- Passing object information
- Passing object link information

For more information about passing information in record lists, see:

- Passing Lifecycle Task Information to Script Functions
- Passing Life-Cycle Association Information to Script Functions

### ***Passing Information by Record List Parameters***

The main way in which a script communicates with SmarTeam is through record list parameters. A record list is a convenient way to pass the attributes of a SmarTeam object, both an ordinary object and a link object. In addition, a record list is also used to pass information for lifecycle task attributes.

Table 3 summarizes the use of the record list parameters FirstPar, SecondPar and ThirdPar in each type of script hook. It indicates whether the record list is input to the script or output from it and also shows the type of information passed in the record list.

*Table 3 Record List Information Passed to and from Scripts*

<b>Script Hook Type</b>	<b>FirstPar</b>	<b>SecondPar</b>	<b>ThirdPar</b>
Profile Card Events	Input object attributes displayed on Profile Card tab	Input all object attributes	Output object attributes
Database Operations	Input object attributes	Not used	Output object attributes
Before LoadLCScreen	Input object attributes	Not used	Input/output default task attributes
After LoadLCScreen	Input object attributes	Create and output task attributes	Input/output default task attributes
Before Individual Life-Cycle Operation	Input object attributes	Create and output task attributes	Output object attributes
Instead, After Individual Life-Cycle Operation	Input object attributes and task attributes	Input task attributes	Output new object attributes for Instead hook
Group Life-Cycle Operation	Input objects' attributes	Input association attributes	Input/output default task attributes and output task attributes
Before File Operation	Input file object attributes	Input operation tool attributes	Output new file object attributes
Instead, After File Operation	Input file object attributes	Input operation tool attributes	Not used
User-Defined Operation	Input selected object attributes.	Input Class_ID of current object	Not used.
Import/Export Operation	Input object attributes.	Input virtual object attributes.	Output object attributes.
Before OnLogin	Not used	Not used	Output login and password
After OnLogin	Input login and password	Not used	Not used

OnBrowse	Input Class_ID and Object_ID of current project	Input Class_ID of the class selected for browsing	Not used
OnRetrieveObjects	Input retrieved objects. Used also for output	Input information about current query	Not used

## Representing SmarTeam Object Attributes

This section discusses in detail record list parameters that pass object attributes. It is assumed that the record list parameters have been converted to COM format (see Converting Procedural Parameters on page 9).

The attributes of a single SmarTeam Object are represented by a SmRecord object, which represents one record or row of a SmRecordList object.

A SmRecordList object is a matrix that can represent the attribute values of some or all of the objects of a class. The headers of the columns of the SmRecordList matrix represent the attributes of the class and the rows of the SmRecordList matrix represent the objects of the class. Each cell of a row contains the value of the object attribute denoted by the cell column's header. Figure 3 shows a SmRecordList object that represents the attributes of a class that has n attributes and m objects. The Object\_ID and the Class\_ID attributes, uniquely identify the object.

Figure 3 Object Attributes Represented by a SmRecordList

Header: Name Type Size	Object Attributes				
	Object_ID sdtObject Identifier 4	Class_ID sdtSmallInt 2	Attribute-3 Type-3 Size-3	...	Attribute-n Type-n Size-n
<b>Object-1</b>	ObjectID-1	ClassID	Value-3-1	...	Value-n-1
<b>Object-2</b>	ObjectID-2	ClassID	Value-3-2	...	Value-n-2
<b>Object-3</b>	ObjectID-3	ClassID	Value-3-3	...	Value-n-3
...	...	...	...	...	...
<b>Object-m</b>	ObjectID-m	ClassID	Value-3-m	...	Value-n-m

The SmRecordList header cells always contain the triple:

Attribute name

Attribute data type, represented by its number (see Table 4)

Size of the attribute value in bytes.

**Note:** For the Memo and Blob attributes data types, the attribute size parameter contains the number of bytes of the possibly truncated attribute in the Record List and not the number of bytes of the actual attribute stored in the Database Record, which can be greater.

### Attribute Data Types

Table 4 presents the values of the data types constants.

*Table 4 Attribute Data Types by Number*

Data Type	Header Number	Data Type	Header Number
sdtNoType	0	sdtObjectIdentifier	10
sdtChar	1	sdtEffectiveDateFrom	11
sdtSmallInt	2	sdtEffectiveDateUntil	12
sdtInteger	3	sdtTimeStamp	13
sdtDouble	4	sdtBoolean	15
sdtBlob	5	sdtRelTimeStamp	16
sdtMemo	6	sdtObject	17
sdtDate	7		
sdtTime	8		

### Passing Object Attributes as a Record List Parameter

The SmRecordList object is dynamic. By reducing the number of columns, it can represent any subset of the attributes of the class and by reducing the number of rows it can represent any subset of the objects of the class.

The dynamic property of the SmRecordList is utilized to pass object attribute information only for those objects and attributes that are involved in the SmarTeam operation being executed.



Accordingly, the required attributes and objects are packed into a SmRecordList and transferred to and from the script function as parameters. Figure 4 shows a record list that has four attributes and one object. This record list might be used to pass information to the script for a SmarTeam Update operation on Object1 where the user changed the object description on the Profile Card. The Object\_ID and Class\_ID attributes, which uniquely identify the object, are always present.

*Figure 4 Record List Parameter*

		Identifying Attributes		Operation-Related Attributes	Additional Attributes
<b>Header:</b>	Name	Object_ID	Class_ID	Description	State
	Type	10	2	1	10
	Size	4	2	71	4
<b>Object1</b>		3798	641	New description	0

### ***Object Attributes Input to a Script***

This section discusses the various object attributes that can be input to a script, usually through FirstPar.

The object attributes input record list passed to the script does not generally contain all possible attributes in the object's class, but rather it contains attributes of three types:

Attributes that identify the object uniquely

Attributes that are relevant to the operation being performed on the object by the user

Additional attributes passed by the application under which the script is running.

### **Attributes that Identify the Object**

The input record list always contains attributes that identify an object uniquely:

Class\_ID

Object\_ID (except for Before Add and InsteadOf Add)

The Class\_ID identifies the object's class and the Object\_ID identifies the particular object in the class.

## Operation-Related Attributes

The input record list can also contain object attributes whose values were changed by the user and are going to be modified by SmarTeam in the database. For example, if the user changed the description of an object in an UPDATE operation, then the CN\_DESCRIPTION attribute, including its new value, is passed to the script function.

## Attributes Passed by SmarTeam

Table 11 in Appendix A shows common, useful object attributes that are frequently passed by SmarTeam to a script through FirstPar. The particular attributes that are passed to the script depend on which operation is being performed.

If your script is to run under a different application than SmarTeam, you need to verify which attributes will be passed to the script.

## Handling Multiple Objects

The input record list can contain more than one object, for example, in a user-defined operation. You determine the number of objects present by the method `SmRecordList.RecordCount`. You manipulate the object values by the property `SmRecordList.ValueAs...(HeaderName, RecordIndex)` where you use `RecordIndex` to specify the object whose value you are reading or writing. For an extended example using multiple objects, see Examples for User-Defined Commands.

For example,

```
'number of objects in FirstPar
RecCount = FirstRec.RecordCount
'read from first object in FirstPar, write in first object in ThirdPar
ThirdRec.ValueAsString("FILE_NAME", 0) =
FirstRec.ValueAsString("FILE_NAME", 0) + ".txt"
'read from second object in FirstPar, write in second object in ThirdPar
ThirdRec.ValueAsString("FILE_NAME", 1) =
FirstRec.ValueAsString("FILE_NAME", 1) + ".log"
```

## Outputting Object Attributes from a Script

The output record list parameter, usually ThirdPar, contains object attributes that were changed or added by the script, which SmarTeam will use during the current operation.

In order to specify an attribute value for an object in an output record list parameter, you:

Use the method `SmRecordList.AddHeader` method to add a header for the new attribute to the `ThirdPar` record list.

Use the property `SmRecordList.ValueAs...(HeaderName, RecordIndex)` to add the appropriate attribute value to the `ThirdPar` record list where `RecordIndex` points to the object to which the attribute value is to apply.

**Note:** Make sure that the triplet (attribute name, size and type) of header parameters you specify in the `AddHeader` method match the attribute definition in the object class.

**Note:** Do not change the values of the identifying attributes `CLASS_ID` and `OBJECT_ID`.

## Changing an Attribute Value

You can use the script to change the value of an attribute that was passed to it in the input parameter, for example for the script hook `Before Update`.

### *Example*

The following is an example of how to change the value of the `FileName` of the object where the `FileName` attribute was input with `FirstPar`. In this example there is only one object, in the first record of the Record List.

*'Check if received attribute FILE\_NAME in FirstPar - which means that the file name was possibly changed by the user.*

*'Add an extension according to file type*

```
ThirdRec.AddHeader ("FILE_NAME", 50,1)
```

```
ThirdRec.ValueAsString("FILE_NAME", 0) =
```

```
FirstRec.ValueAsString("FILE_NAME", 0) + ".txt"
```

### *Example*

This example shows how to change the value of an attribute that was not received in `FirstPar`, based on the value of an attribute that was received in `FirstPar`. The script defines the `Description` of the object as the concatenation of some constant prefix plus the file name of the object.

*'[add code to example: Check if received attribute FILE\_NAME in FirstPar - which means that the file name was possibly changed by the user.*

```
ThirdRec.AddHeader ("CN_DESCRIPTION", 21, sdtChar)
```

```
ThirdRec.ValueAsString("CN_DESCRIPTION", 0) = "prefix for file" +
```

```
FirstRec.ValueAsString("FILE_NAME", 0) "
```

***Example***

The following example shows how to add an attribute header and value for an attribute that is not based on any information passed in FirstPar.

```
ThirdRec.AddHeader ("CN_COST", 21, sdtChar)
ThirdRec.ValueAsString ("CN_COST",0) = 1000000
```

An example of such a ThirdPar Record List for one object is shown in Figure 5. The attribute CN\_COST has been added. Its value will be added to the object in the database.

*Figure 5 ThirdPar Record List*

	Attributes
<b>Header:</b> Name Type Size	COST 1 21
<b>Object1</b>	1000000

***Example***

This example has more than one object in the record list. The object records in the ThirdPar Record List are indexed in the same way as the object records in the FirstPar Record List. Thus, the following code reads the file name from the second object in the FirstPar, which corresponds to the index 1, and writes the file name in the same object in the ThirdPar, which also corresponds to index 1.

```
ThirdRec.ValueAsString("FILE_NAME", 1) =  
FirstRec.ValueAsString("FILE_NAME", 1) + ".txt"
```

***Passing Link Information to Script Functions***

When a database operation such as Add or Update is performed on a link object, the link information is passed to the script function by means of the input record list parameter. This section discusses these parameters in detail. In this section, it is assumed that the record list parameters have been converted to COM format (see Converting Procedural Parameters on page 9).

***Representing SmarTeam Links***

The link attributes of a SmarTeam link such as hierarchical link are represented by a SmRecord object, which represents one record or row of a SmRecordList object.

An SmRecordList object can represent link attributes for a set of links. The headers of the columns of the SmRecordList matrix represent the link attributes and the rows of the SmRecordList matrix represent the link objects. Each cell of the row contains the value of the link attribute denoted by the cell column's header.

### ***Link Object Attributes Input to a Script***

There are three types of link objects:

- Hierarchic links
- One-level links
- Complex links

#### **Hierarchic Links**

Figure 6 shows a SmRecordList object that represents n hierarchical link attributes of m links. The attributes shown identify the linked parent and son objects by their Class\_ID and Object\_ID. The Class\_ID and Object\_ID of the link object itself are not shown in the figure.

*Figure 6 Hierarchical Link Attributes Represented by a SmRecordList*

	Link Attributes				
<b>Header:</b> Name	Class_Id1	ObjectId1	Class_Id2	ObjectId2	...
Type	2	10	2	10	
Size	2	4	2	4	
<b>Link-1</b>	Class_Id1-1	ObjectId1-1	Class_Id2-1	ObjectId2-1	...
<b>Link-2</b>	Class_Id1-2	ObjectId1-2	Class_Id2-2	ObjectId2-2	...
<b>Link-3</b>	Class_Id1-3	ObjectId1-3	Class_Id2-3	ObjectId2-3	...
...	...	...	...	...	...
<b>Link-m</b>	Class_Id1-m	ObjectId1-m	Class_Id2-m	ObjectId2-m	...

Table 5 describes the attributes for hierarchic links.

*Table 5 Attributes for Hierarchical Links*

Attribute Name	Type (preface with sdt)	Description
CLASS_ID	SmallInt	CLASS_ID of link.
OBJECT_ID	ObjectIdentifier	OBJECT_ID of link.
CLASS_ID1	SmallInt	CLASS_ID of parent object in hierarchical link
OBJECT_ID1	ObjectIdentifier	OBJECT_ID of parent object in hierarchical link
CLASS_ID2	SmallInt	CLASS_ID of son object in hierarchical link
OBJECT_ID2	ObjectIdentifier	OBJECT_ID of son object in hierarchical link

### One-Level and Complex Links

Figure 7 shows a SmRecordList object that represents n one-level link attributes of m links. The attributes shown identify the linked objects Object1 and Object2 by their Class\_ID and Object\_ID.

*Figure 7 One-Level Link Attributes Represented by a SmRecordList*

	Link Attributes				
	Class_Id1	ObjectId1	Class_Id2	ObjectId2	...
<b>Header:</b> Name Type Size	Class_Id1 2 2	ObjectId1 10 4	Class_Id2 2 2	ObjectId2 10 4	...
<b>Link-1</b>	Class_Id1-1	ObjectId1-1	Class_Id2-1	ObjectId2-1	...
<b>Link-2</b>	Class_Id1-2	ObjectId1-2	Class_Id2-2	ObjectId2-2	...
<b>Link-3</b>	Class_Id1-3	ObjectId1-3	Class_Id2-3	ObjectId2-3	...
...	...	...	...	...	...
<b>Link-m</b>	Class_Id1-m	ObjectId1-m	Class_Id2-m	ObjectId2-m	...

Table 6 describes the attributes for one-level or complex links, which occur in individual lifecycle operations

*Table 6 Attributes for One-Level or Complex Link*

Attribute Name	Type (preface with sdt)	Description
CLASS_ID	SmallInt	CLASS_ID of link.
OBJECT_ID	ObjectIdentifier	OBJECT_ID of link.
CLASS_ID1	SmallInt	CLASS_ID of first object in one-level link
OBJECT_ID1	ObjectIdentifier	OBJECT_ID of first object in one-level link
CLASS_ID2	SmallInt	CLASS_ID of second object in one-level link
OBJECT_ID2	ObjectIdentifier	OBJECT_ID of second object in one-level link

### ***Getting Class Attributes in the Data Model***

It is important to know which class attributes are defined in the SmarTeam data model for two reasons:

In order to verify if a specific attribute is passed to a script in a record list parameter, you use the function

`IsmRecordListHeaders.HeaderExists ("HeaderName")`. In order to use that function you need to know the value of the attribute's HeaderName.

To know which new record list headers can be defined for an object belonging to a certain class.

Please consult the SmarTeam administrator for a complete list of attributes available to a script at your installation. Another way to obtain the class attributes in the data model is to use the SmartWizard. Click the Report button on the opening screen.

## 4. Scripts for Profile Card Events

This section discusses script hooks related to SmarTeam – Editor Profile Cards. In SmarTeam – Editor, a Profile Card represents an object such as a Document or a Part and displays some of the object’s attributes. These script hooks are related to viewing Profile Cards and to database actions performed on objects represented by the Profile Cards such as adding or updating an object in the database.

### Purpose

The purpose of the Profile Card script hooks is to enable you to intervene in the display of a Profile Card to:

Change the display of default object attributes on the Profile Card

Establish control over user input, for example, reject out-of-range input

Display information in a Profile Card field based on user input in other fields.

### When Used

You can use Profile Card script hooks:

When displaying a Profile Card in the SmarTeam View. This includes navigating through the Profile cards in the “Edit/Find Object” command.

During a SmarTeam Profile Card action such as Add or Update when:

- Starting or terminating the Profile Card action.
- When entering or exiting from a specific Profile Card field.

When clicking on a specific button on the Profile Card.

In the Find Object by Attribute operation, using a Profile Card to specify the query conditions.

Saving data to SmarTeam from an external application that has been integrated with SmarTeam, using a Profile Card to specify information for SmarTeam.

### Applicable to Class

You set up a script hook for the class represented by a Profile Card, for example, Document or Folder. This means that the script is activated for any object in the class. Thus, if you attach a script to a Document Profile Card, it is activated every time you open a Profile Card for any Document.



## ***Profile Card Script Hooks***

The following Profile Card script hooks are available:

<b>Script Hook</b>	<b>Applicable to</b>	<b>Occurs</b>
Screen StartUp	View, Add, Update, Save to SmarTeam	Prior to first displaying Profile Card
Screen Exit	Add, Update, Save to SmarTeam	Prior to exiting Profile Card
OnEnter	Add, Update, Save to SmarTeam	When establishing input focus on Profile Card field
OnExit	Add, Update, Save to SmarTeam	When removing input focus from Profile Card field
OnClick	Add, Update, Save to SmarTeam, Find Object by Attribute	When clicking on button or URL.
CALL_SCRIPT	Save to SmarTeam	Prior to first displaying Profile Card, prior to Screen StartUp hook.

## ***Attaching a Script to a Profile Card Script Hook***

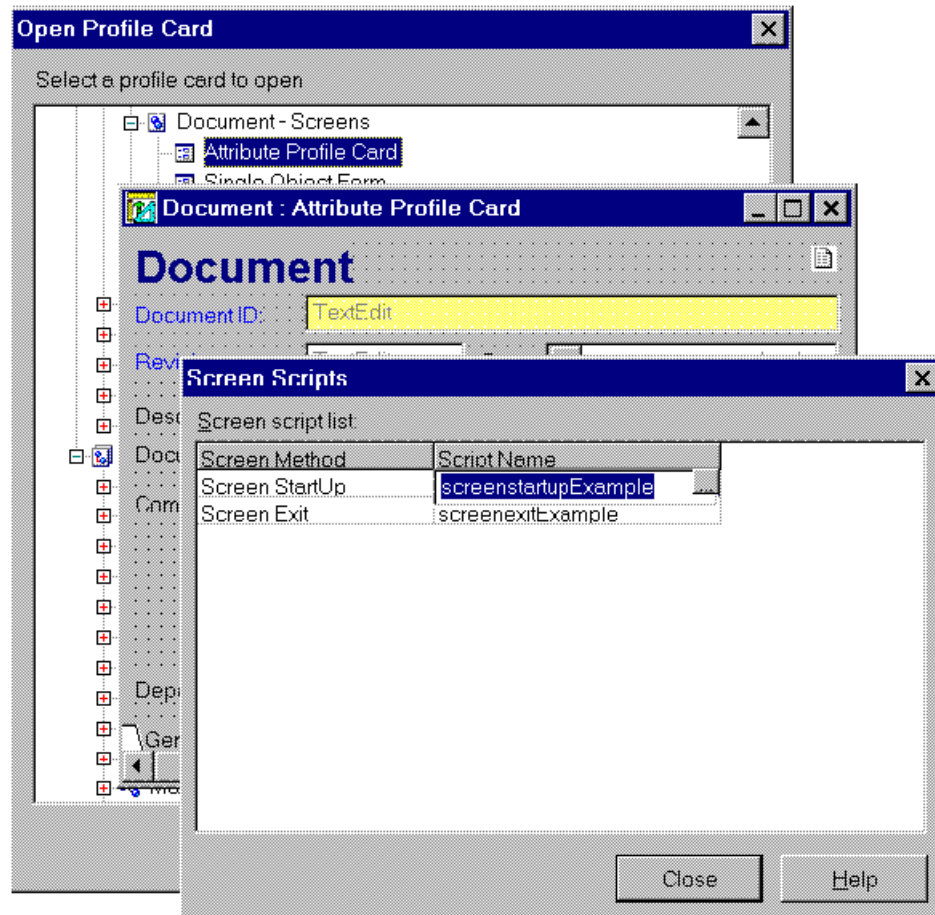
Once the script has been written, you use the Form Designer utility to attach it to the desired script hook (the CALL\_SCRIPT script hook is an exception, see its section below). The Form Designer utility can also be invoked directly from SmarTeam – Editor using the menu item Tools/Form Designer. This section presents a summary of the procedure; see the SmarTeam – Editor Administrator's Guide for more information.

## **Scripts in Search Editor**

Other than scripts hooked to buttons, it is not possible to hook a script for execution in a Query by Attribute (QBA) search window. Some scripts, such as the "Screen Startup/Exit" and "On Enter/Exit" scripts, contain parameters that include information about a specific object, and errors may occur when trying to run such scripts in the query, as no "current object" exists when running a QBA.

## Screen Startup Hook

The following steps summarize the procedure for attaching a script to a Profile Card through the Screen Startup or Screen Exit hook. The successive screens are illustrated in Figure 8.



*Figure 8 Attaching a Script to the ScreenStartup Hook*

Start the Form Designer utility, or perform File/Open if you are already in the Form Designer utility, to get the Open Profile Card screen.

Select the Profile Card for the desired class on the Open Profile Card screen and Click Ok to get the selected Profile Card.

Perform Tools/Scripts on the Form Designer utility to get the Screen Scripts window. The two script hooks that are associated with the screen are displayed: Screen StartUp and Screen Exit.

On the Screen Scripts window, click on the browse button in the desired script hook name field to get the Script Browser.

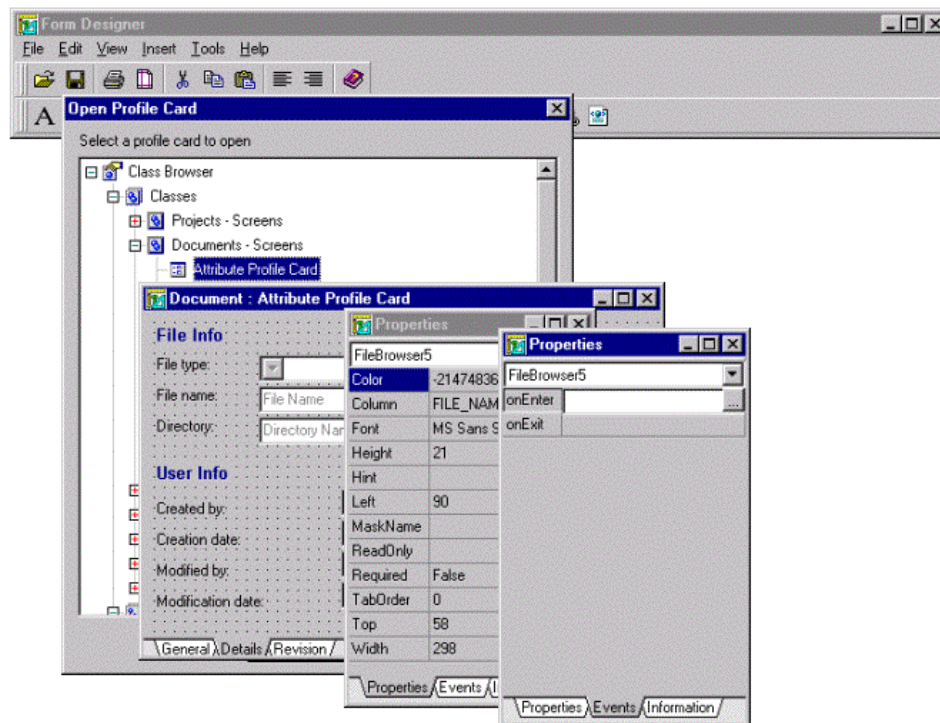
In the Script Browser, choose a file and a script to attach to the selected script hook and click **Ok**.

To remove a script, select it on the Screen Scripts window and delete it.

For more information about assigning scripts to Profile Card script hooks, see Chapter 3 of the SmarTeam – Editor Administrator’s Guide.

## **OnEnter, OnExit Hook**

The following steps summarize the procedure for attaching a script to a specific field of a Profile Card through the OnEnter and OnExit hooks. The successive screens are illustrated in Figure 9.



*Figure 9 Attaching a Script to the OnEnter Hook*

Start the Form Designer, or perform File/Open if you are already in the Form Designer utility, to get the Open Profile Card screen.

Select the Profile Card for the desired class on the Open Profile Card screen and Click **Ok** to get the selected Profile Card.

Click on the Profile Card field to which you want to attach the OnEnter or OnExit script (for example, File Name as shown in the figure)

Press F4 to get the Properties window for that field.

**Note:** The object attribute name of the field appears in the Column row of the Properties tab (FILE\_NAME in the figure); this is the name you use in the script to refer to that field -- prefaced by NM\_.

Click on the Events tab on the Properties window to get the Events window; this window displays the OnEnter and OnExit script hooks that are associated with the field.

Click on in the desired script hook name field in the Events window to get the Script Browser.

In the Script Browser, choose a file and a script to attach to the selected script hook and click Ok.

To remove a script, select it on the Events window and delete it.

For more information about assigning scripts to Profile Card script hooks, see Chapter 3 of the SmarTeam – Editor Administrator’s Guide.

## ***Script Execution Timing***

This section describes the hook timing for object database operations such as Add or Update. There are two sets of script hooks that are relevant to the object database operations: the Profile Card script hooks described in this section and the database operation script hooks described in the section Scripts for Object Database Operations.

The Profile Card script hooks all occur before the Database Operations script hooks. For example, the script hooks for an Update operation execute with the following timing:

User invokes Update on an object on the SmarTeam view

### **Screen Startup hook occurs**

The user performs data entry on the Profile Card – the **OnEnter** and **OnExit** hooks occur

The user clicks on a button or URL on the Profile Card – the **OnClick** hook occurs

User clicks **OK** on the Update screen

### **OnExit hook occurs**

### **Screen Exit hook occurs**

### **Before Update hook occurs**

The input data is updated in the database – by SmarTeam – Editor or by the **InsteadOf Update** hook operation

### **After Update hook occurs**

The Update screen is removed and changes caused by the hook scripts are displayed on the SmarTeam View window.

## **Screen Startup**

The Screen Startup hook is useful for:

- Entering default text into Profile Card fields at run time
- Performing calculations based on information in Profile Card fields at run time
- Enforcing standards on a Profile Card field.

### **When Used**

The Screen Startup hook can occur in a number of different situations in which a Profile Card for an object or object action is displayed. In each case, the Screen Startup hook occurs immediately prior to displaying the Profile Card.

The Screen Startup hook can be set to occur:

When viewing an object in the SmarTeam View, prior to displaying the object's Profile Card

When invoking the Profile Card actions:

*Update*, *Add as Copy*, or *Add* prior to displaying the Profile Card for the object action.

When saving data to SmarTeam from an external application, which has been integrated with SmarTeam, where a Profile Card is used to specify information for SmarTeam: prior to displaying the Profile Card.

## Script Hook Parameters

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	<p>Describes the current operation for which the script is being invoked.</p> <p>ADD:</p> <p>When adding a new Profile Card of this class.</p> <p>VIEW:</p> <p>When displaying the Profile Card on a SmarTeam View.</p> <p>When displaying a Profile Card after performing SmarTeam Edit/Find Object query search.</p> <p>UPDATE:</p> <p>when updating the Profile Card, before displaying the Update Profile Card screen.</p> <p>AddAsCopy:</p> <p>When performing Add as Copy on the Profile Card, before displaying the AddAsCopy Profile Card screens.</p>
FirstPar	Input object attributes. Only those object attributes corresponding to fields on the Profile Card tab about to be displayed.
SecondPar	Input object attributes. All object attributes in the database are input.
ThirdPar	<p>Output object attributes.</p> <p>You can change attributes values by filling the record list with values (the first value of each element is considered by the system). See the section Outputting Object Attributes from a Script.</p> <p>SmarTeam – Editor carries out the changes only if the Profile Card is enabled for the current user.</p>
Return Value	If the script returns Err_Gen or Err_Refuse, the screen is not shown.

## **Screen Exit**

The Screen Exit hook can be used to check user input on Update or Add operations. For example, you can reject out-of-range input or you can search the database to see if the user input is unique.

It is relevant for the operations: Add, Update, Save to SmarTeam.

You use this hook to replace field values that were input by the user. Output the replacement values in ThirdPar and use a return code of Err\_None. These replacement values are entered into the database instead of the ones that the user entered and they appear on the screen after the database operation. If you use a return code of Err\_Gen, the replacement values will not be used.

The Screen Exit hook is called when exiting a Profile Card. It can be called in the same operations that the Screen Startup hook can be called, except for the VIEW operation.

## **Script Hook Parameters**

### **Arguments**

<b>Argument</b>	<b>Description</b>
ApplHndl	Input. See Table 1.
SelectOp	Describes the current operation for which the script is being invoked.  ADD:  When adding a new Profile Card of this class.  VIEW:  There is no Screen Exit script hook for VIEW operations.  UPDATE:  When updating the Profile Card, before displaying the Update Profile Card screen.  AddAsCopy:  When performing Add as Copy on the Profile Card, before displaying the AddAsCopy Profile Card screens.
FirstPar	Input containing all object attributes in the database (Same as Screen Startup hook, SecondPar)
SecondPar	Not used.

---

ThirdPar	Output. You can change attribute values by filling the record list with values (the first value of each element is considered by the system). See the section Outputting Object Attributes from a Script.
Return Value	If the script returns Err_None, the ThirdPar output is used. If the script returns Err_Gen or Err_Refuse, the exit continues but the ThirdPar output is not used.

---

## ***OnEnter***

The OnEnter hook can be used for the same purposes as the Screen Startup hook, normally to enter default values.

It is relevant for the operations: Add, Update, Save to SmarTeam.

The OnEnter hook occurs when clicking on the Profile Card field for which it is set.

If you use a return code of Err\_Gen, the field is entered but the replacement values will not be used.

The OnEnter hook works for the following screen objects:

TextEdit box

Memo box

Combo box

Multicombo box

Check box

Radio group

Viewer

Date edit

Date/time edit

Relative time

File browser

Vault browser

HTML browser



## **Script Hook Parameters**

### **Arguments**

<b>Argument</b>	<b>Description</b>
ApplHndl	Input. See Table 1.
SelectOp	Object attribute name associated with the current Profile Card field, for example, "CN_ID".
FirstPar	Input object attributes -- only those object attributes corresponding to fields on the Profile Card tab currently displayed. The values correspond to what currently appears in the tab fields, including changes made by the user.
SecondPar	Input containing all object attributes in the database. These values do not necessarily correspond to what appears in the Profile Card tab fields -- if the user has made changes.
ThirdPar	Output. You can change or add attribute values (the first value of each element is considered by the system.) See section Outputting Object Attributes from a Script.
Return Value	If the script returns Err_None, the ThirdPar output is used. If the script returns Err_Gen or Err_Refuse, the action continues but the ThirdPar output is not used.

---

### **OnExit**

The OnExit hook can be used for the same purposes as the Exit Screen hook.

It is relevant for the operations: Add, Update, Save to SmarTeam.

The OnExit hook occurs when exiting the Profile Card field for which it is set, for example, when clicking on another field or when clicking Ok on the Profile Card.

You use this hook to replace field values that were input by the user. Output the replacement values in ThirdPar and use a return code of Err\_None. These replacement values are entered into the field instead of the ones that the user entered. If you use a return code of Err\_Gen, the exit continues but the replacement values are not used.

The OnExit hook works for the same screen objects as the OnEnter hook.

## **Script Hook Parameters**

### **Arguments**

<b>Argument</b>	<b>Description</b>
ApplHndl	Input. See Table 1.
SelectOp	Object attribute name associated with the current Profile Card field, for example, "CN_ID"
FirstPar	Input object attributes -- only those object attributes corresponding to fields on the Profile Card tab currently displayed. The values correspond to what currently appears in the tab fields, including changes made by the user.
SecondPar	Input containing all object attributes in the database. These values do not necessarily correspond to what appears in the Profile Card tab fields -- if the user has made changes.
ThirdPar	Output. You can change or add attribute values by filling the record list with elements and values. See section <i>Outputting Object Attributes from a Script</i> .
Return Value	If the script returns Err_None, the ThirdPar output is used. If the script returns Err_Gen or Err_Refuse, the exit continues but the ThirdPar output is not used.

### **OnClick**

The OnClick script hook can be used to attach a special function to a button or URL on a Profile Card.

It is relevant to the operations: Add, Update, Save to SmarTeam, Find Object by Attribute.

The OnClick hook can be set to occur for:

a button

a hyperlink

## Script Hook Parameters

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	“None”
FirstPar	Input object attributes -- only those object attributes corresponding to fields on the Profile Card tab currently displayed. The values correspond to what currently appears in the tab fields, including changes made by the user.
SecondPar	Input containing all object attributes in the database. These values do not necessarily correspond to what appears in the Profile Card tab fields -- if the user has made changes.
ThirdPar	Output. You can change or add attribute values by filling the record list with elements and values (the first value of each element is considered by the system). See section Outputting Object Attributes from a Script.
Return Value	If the script returns Err_None, the ThirdPar output is used. If the script returns Err_Gen or Err_Refuse, the exit continues but the ThirdPar output is not used.

---

## CALL\_SCRIPT

The CALL\_SCRIPT hook is useful for:

Entering value into object attributes at run time prior to displaying Profile Card

Performing calculations based on information in object attributes at run time

### When Used

The CALL\_SCRIPT hook occurs prior to displaying the Profile Card when using the methods ISmCADInterface.Save and ISmCADInterface.OdmaSave to save data to SmarTeam from an integration. The Profile Card is used to specify information about the saved object before it is updated in SmarTeam. The CALL\_SCRIPT hook occurs prior to the Screen Startup hook.

### **Difference between CALL\_SCRIPT and Screen Startup Hooks**

The CALL\_SCRIPT hook, although it occurs at the same point in time as the Screen Startup hook, has a different purpose. As mentioned in the next section, the CALL\_SCRIPT hook is set by the Administrator in the System Configuration for each integration separately. Accordingly, it is used for global replacement of object information for all users of an integration. An example might be the default file name of an object. The script would convert a SmarTeam default file name into a standard default file name of the organization.

### **Attaching a Script to CALL\_SCRIPT Script Hook**

As opposed to the other Profile Card script hooks, you do not use the Form Designer to attach a script to the CALL\_SCRIPT script hook. A script is attached to the CALL\_SCRIPT script hook by the System Configuration utility, using the System Configuration Editor. The System Configuration contains an entry for attaching a CALL\_SCRIPT script hook for each integration tool.

The key path for each tool is as follows:

<Integration Name>.CALL\_SCRIPT=<Script Name>

## ***Script Hook Parameters***

### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Describes the current operation for which the script is being invoked.  This value is received from the integration; it does not need to be entered.
FirstPar	Input object attributes. Only those object attributes corresponding to fields on the Profile Card tab about to be displayed.
SecondPar	Input object attributes. All object attributes in the database are input.
ThirdPar	Output object attributes.  You can change attributes values by filling the record list with values (the first value of each element is considered by the system). See the section Outputting Object Attributes from a Script.  SmarTeam – Editor carries out the changes only if the Profile Card is enabled for the current user.
Return Value	If the script returns Err_Gen or Err_Refuse, the screen is not shown.

---

## ***Script Execution Timing***

This section describes the hook timing for the operations `ISmCADInterface.Save` and `ISmCADInterface.OdmaSave`. There are two sets of script hooks that are relevant to the object database operations: the Profile Card script hooks described in this section and the database operation script hooks described in the section *Scripts for Object Database Operations*.

The Profile Card script hooks all occur before the Database Operations script hooks. The script hooks for the Save operations execute with the following timing:

User invokes `ISmCADInterface.Save` or `ISmCADInterface.OdmaSave` on an object on the SmarTeam integration

**CALL\_SCRIPT hook occurs**

**Screen Startup hook occurs**

The user performs data entry on the Profile Card – the **OnEnter** and **OnExit hooks occur**

The user clicks on a button or URL on the Profile Card – the **OnClick hook occurs**

User clicks **OK** on the Save screen

**OnExit hook occurs**

**Screen Exit hook occurs**

**Before Update hook occurs**

The input data is updated in the database – by SmarTeam – Editor or by the **InsteadOf Update hook** operation

**After Update hook occurs**

The Save screen is removed and changes caused by the hook scripts are displayed on the SmarTeam View window.

## ***Examples***

### **ScreenStartup**

This script is to be attached to the ScreenStartup script hook. It sets the description and phase defaults in the Profile Card by ThirdPar.

```
Function ScreenStartupExample(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
```

```
    Dim SmSession As SmApplic.SmSession
```

```
    Dim FirstRec As Object
```

```
    Dim SecondRec As Object
```

```
    Dim ThirdRec As Object
```

```
    Dim PhaseClassId As Integer
```

```
    Dim Phase As ISmLookupObject
```

```
    ' Convert pointer to COM object SmSession
```

```
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
```

```
    ' Convert input parameter to COM object
```

```
    CONV_RecListToComRecordList FirstPar,FirstRec
```

```
    CONV_RecListToComRecordList ThirdPar,ThirdRec
```

```
    ' Add to description
```

```
    ThirdRec.AddHeader "CN_DESCRIPTION", 71, 1
```

```
ThirdRec.ValueAsString("CN_DESCRIPTION", 0) = "Design Description of  
Part " + FirstRec.ValueAsString("CN_DESCRIPTION", 0)  
' Get phase lookup table class id  
PhaseClassId = SmSession.Metainfo.SmClassByName("Phase").ClassId  
' Get lookup object by unique name  
Set Phase =  
SmSession.ObjectStore.GetSmLookUpByUniqueName(PhaseClassId,"Design")  
' Add attribute to result record list  
ThirdRec.AddHeader NM_PHASE,SIZE_OBJ_ID,TDMT_OBJ_ID  
' Set phase stage on lifecycle screen as Design  
ThirdRec.ValueAsInteger(NM_PHASE,0) = Phase.Id  
CONV_ComRecListToRecordList ThirdRec, ThirdPar  
ScreenStartupExample = Err_None  
End Function
```

## OnExit

This script is attached to the On Exit hook for the Project class. It checks the user input in the Budget field (in the SmDemo database.) If the project budget is outside the range 1000 to 10000, the user entry is replaced by the maximum or the minimum allowed.

```
Declare Sub CONV_RecListToCOMREcordList Lib "Smtdm32" (ByVal RecList As  
Long, ByRef COMRecList As SmREcList.SmRecordList)  
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As  
SmREcList.SmRecordList, ByRef RecList As Long)  
Function OnExit(ApplHndl As Long, Sstr As String, FirstPar As Long,  
SecondPar As Long, ThirdPar As Long ) As Integer  
Dim FirstRec As Object  
Dim SecondRec As Object  
Dim ThirdRec As Object  
  
On Error GoTo HandleError  
  
' Convert input parameter to COM object  
CONV_RecListToComRecordList FirstPar, FirstRec  
CONV_RecListToComRecordList ThirdPar, ThirdRec  
If FirstRec.ValueAsInteger("CN_TOTAL_BUDGET",0) > 10000 Then  
ThirdRec.ValueAsInteger("CN_TOTAL_BUDGET",0) = 10000  
End If  
If FirstRec.ValueAsInteger("CN_TOTAL_BUDGET",0) < 1000 Then  
ThirdRec.ValueAsInteger("CN_TOTAL_BUDGET",0) = 1000  
End If  
CONV_COMRecListToREcordList ThirdRec, ThirdPar
```

```
Exit Function
HandleError:
    MsgBox Err.Description
End Function
```

## OnClick

This script is attached to the OnClick hook for all classes. It displays the E-mail address of the user who was the creator of the object corresponding to the Profile Card on which the clicked button is located.

```
Declare Sub CONV_RecListToCOMREcordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmREcList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmREcList.SmRecordList, ByRef RecList As Long)
Function OnClick(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim User As SmApplic.ISmObject
    Dim WorkObject As SmApplic.ISmObject
    Dim UserClassId As Integer
    Dim UserId As Long
    Dim FirstRec As Object
    Dim SmSession As SmApplic.SmSession
    On Error GoTo HandleError
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    CONV_RecListToCOMREcordList SecondPar,FirstRec
    Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)
    UserClassId =
WorkObject.SmClass.Attributes.ItemByName("USER_OBJECT_ID").ReferencedClassId
    UserId = WorkObject.Data.ValueAsInteger("USER_OBJECT_ID")
    Set User = SmSession.ObjectStore.RetrieveObject(UserClassId , UserId)
    MsgBox "The object creator email: " &
User.Data.ValueAsString("USER_EMAIL")
    Exit Function
HandleError:
    MsgBox Err.Description
End Function
```



## 5. Scripts for SmarTeam Operations

Script hooks are available for many different types of SmarTeam operations including:

Scripts for Object Database Operations

Scripts for Individual GUI-Based Lifecycle Operations

Scripts for Individual Non-GUI Lifecycle Operations

Scripts for Group Lifecycle Operations

Scripts for File Operations

Scripts for Authorization Operations

This chapter discusses these hooks in detail.

### ***Script Hooks Available for Operations***

The script hooks available for a SmarTeam operation depends on the class of the object on which the operation is performed. For each SmarTeam class, a certain subset of SmarTeam operations has script hooks. For each class, you can view the set of operations that have a script hook in the System/Operation column in the Script Maintenance utility. Figure 10 shows the Document class operations that have a script hook.

In addition, some SmarTeam operations have script hooks for only two of the three stages. Table 7 lists the script hooks available for each SmarTeam operation and stage.

Table 7 Script Hooks for SmarTeam Operations

Operation in Script Maintenance	Stage		
	Before	After	InsteadOf
<b>Database Operations on Objects</b>			
Add	X	X	X
Add As Copy	X	X	X
Update	X	X	X
Delete	X	X	X

<b>Script Hooks for Individual Life-Cycle Operations (GUI dependent)</b>			
Load Life-Cycle Screen	X	X	
On Life-Cycle Click CheckOut	X		
On Life-Cycle Click CheckIn	X		
On Life-Cycle Click Release	X		
On Life-Cycle Click New Release	X		
On Life-Cycle Click Obsolete	X		
On Life-Cycle Click Copy File	X		

<b>Script Hooks for Individual Life-Cycle Operations</b>			
Check Out	X	X	X
Undo Check Out	X	X	X
Check In	X	X	X
Release	X	X	X
NewRelease	X	X	X
Obsolete	X	X	X
Copy File	X	X	X

Script Hooks for Group Life-Cycle Operations			
Life-Cycle Stage 1	X	X	X
Life-Cycle Stage 2	X	X	

File Operations			
Edit	X	X	X
View	X	X	X
RedLine	X	X	X
Print	X	X	X
View Object			
On Viewer	X		
Copy File	X	X	X

Authorization Operations			
OnLogin	X	X	
OnBrowse	X		
OnRetrieveObjects		X	

CAD Operations			
Object identification for CAD	X		X

### ***Attaching a Script to a Script Hook***

Once the script has been written, you use the Script Maintenance utility to attach it to the desired script hook. Figure 10 shows various scripts that have been assigned to the script hooks. The Script Maintenance utility can also be invoked directly from SmarTeam – Editor using the menu item Tools>Script Maintenance. See the Script Maintenance document for more information.

In the Script Maintenance utility you first select a file to attach and then select the desired script function in the file.

## Attaching Scripts to Multiple Classes

You control the activation of the script on a link object by the level of the class tree at which you assign the script in Script Maintenance. If you assign a script to the “Class Browser” node of the class tree, that script will be activated for all object classes, including regular and link objects. For example, in the Add operation, if you add a new object in MainClassTree SmarTeam View, the script will be activated twice, once when the new object is added and once when the link between the new object and its linked object is added. If you want the script to be activated only when the new object is added (and not when the link is added), assign the script to the actual class or else assign it at the level of the “Classes” node in the class tree.

**Note:** In a Vault Server environment, on saving a Redline hook directly to a class, the script hook will not be activated because the class id is not passed to the function that calls the hook.

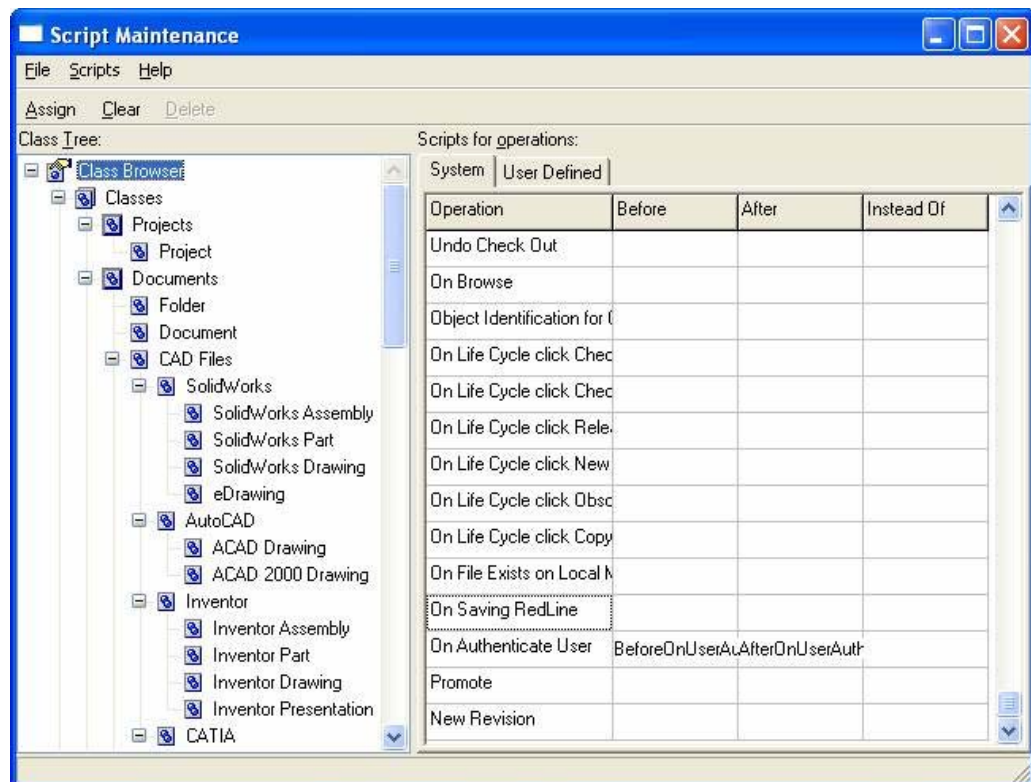


Figure 10 Script Maintenance showing Script Hooks for the Document Class

---

## Scripts for Object Database Operations

Scripts can be attached to script hooks related to SmarTeam database operations on objects such as Add and Update. For example, you can create a script to send an E-mail each time you add an object to the database. See examples at the end of each section.

Scripts for Profile Card Events also work in database operations. The difference is that the Profile Card script hooks work on the SmarTeam – Editor Profile Card of the Add or Update operation to control user input, whereas the script hooks of this section work after the user input is complete (see the section *Script Execution Timing*). The script hooks in this section also work for database operations in a non-GUI application.

Scripts can be attached to the following SmarTeam database operations:

Add  
AddAsCopy  
Update  
Delete

### ***Add, AddAsCopy***

This script is performed when adding a new object to the database by the SmarTeam Add or AddAsCopy operation.

The script can act on link objects as well as on regular objects. For example, you can attach a script to the operation of adding a tree link object in class “Project Trees”.

### ***Script Execution Timing***

The following table shows the timing of script hook execution relative to the physical addition of the object to the database.

Stage	Timing
<b>Before</b>	The script is executed before the physical addition of the object to the database.
<b>After</b>	The script is executed after the physical addition of the object to the database.
<b>InsteadOf</b>	The script is executed instead of the system’s add operation. The system expects the script to execute its own add operation.

## **Hook Timing in the SmarTeam Add Operation**

This section describes the hook timing relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the Add script hooks.

The Add script hooks can execute for the SmarTeam Add operation on the following user actions:

Adding an object by the Add operation

Adding a link by dragging one object and dropping it over another.

### **Add Operation:**

User invokes the Add operation on an object on the SmarTeam view.

Profile Card hooks occur, if set. See section Script Execution Timing .

User clicks OK on the Add screen.

If the user makes no changes on the Add screen, the Add hooks still occur.

### **Before Add hook occurs.**

The input data is added in the database –by SmarTeam – Editor or by the **InsteadOf Add hook** operation.

### **After Add hook occurs.**

The Add screen is closed and changes caused by the hook scripts are displayed on the SmarTeam view window.

### **Adding a Link:**

User drags one object over another on the SmarTeam view.

User clicks OK on the dialog box: “Link: ...”.

### **Before Add hook occurs.**

The input data is added in the database – by SmarTeam – Editor or by the **InsteadOf Add hook** operation.

The script attached to hook **Before Add** operation will run after the confirmation message regarding the creation of the link to the parent object

### **After Add hook occurs.**

The added link is displayed on the SmarTeam view window.

**Aborting:**

If the operation is aborted by returning Err\_Gen in the Before Add hook, the Add screen is left as is and no message is displayed by the system.

**Note:** Lifecycle operations such as Check Out do not trigger this hook.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before Add operation script hook.

#### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	'ADD'
FirstPar	Input.  The attributes passed depend on the kind of object being added. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.  The Object_ID is not available for the Before and InsteadOf script hooks because at that stage the object is not yet added to the database.
SecondPar	Not used
ThirdPar	Output new or changed object attribute values.  See the section Outputting Object Attributes from a Script.
Return value	See Table 1.  You can cause the Add operation to be aborted by assigning Err_Gen as the return value.

### **After Hook**

The following table describes the arguments passed in the After Add operation script hook.

## Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	‘ADD’
FirstPar	Input.  The attributes passed depend on the kind of object being added. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Does not update database.  See the section Outputting Object Attributes from a Script.
Return value	See Table 1.

---

## InsteadOf Hook

The following table describes the arguments passed in the InsteadOf Add operation script hook.



## Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	'ADD'
FirstPar	Input.  The attributes passed depend on the kind of object being added. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.  The Object_ID is not available for the Before and InsteadOf script hooks because at that stage the object is not yet added to the database.
SecondPar	Not used
ThirdPar	Output new object attribute values. Must contain Class_ID and Object_ID for the new object added.  See the section Outputting Object Attributes from a Script.
Return value	See Table 1.

---

## Examples

### Before Add

This script is designed for the Before Add script hook for the class Users (SmDemo Database). It is applicable to adding a new employee to the database. It checks if the E-mail record exists and is filled in for the new employee before adding the record.

```
Declare Sub CONV_RecListToCOMREcordList Lib "Smtm32" (ByVal RecList As Long, ByRef COMRecList As SmREcList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtm32" (ByVal COMRecList As SmREcList.SmRecordList, ByRef RecList As Long)
Function BeforeAddUser(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim FirstRec As Object
    On Error GoTo HandleError
```

```

        CONV_RecListToCOMREcordList    FirstPar,FirstRec
' If the E-mail header does not exist - notify user
    If Not FirstRec.Headers.HeaderExists("USER_EMAIL") Then
        BeforeAddUser = Err_Gen
        MsgBox "The employee E-mail heading does not exist"
    Else
' If the E-mail address is missing - notify user
        If Len(Trim(FirstRec.ValueAsString("USER_EMAIL",0)))>0 Then
            BeforeAddUser = Err_None
        Else
            BeforeAddUser = Err_Gen
            MsgBox "The employee E-mail address is missing"
        End If
    End If
Exit Function
HandleError:
    MsgBox Err.Description
End Function

```

## InsteadOf Add

This script is designed for the InsteadOf Add script hook for the class Projects (SmDemo Database). It checks if a manager has been assigned to the project. If not, it notifies the user, otherwise it adds the object by itself.

```

Declare Sub CONV_RecListToCOMREcordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRecList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmRecList.SmRecordList, ByRef RecList As Long)
Function InstdAdd(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim FirstRec As Object
    Dim ThirdRec As Object
    Dim SmSession As SmApplic.SmSession
    Dim Behavior As SmApplic.ISmBehavior
    Dim NewWorkObject As SmApplic.ISmObject
    Dim ProjectManager As String
    Dim ClassId As Integer
    Dim NewObjectId As Long

    CONV_RecListToCOMREcordList    FirstPar,FirstRec
    CONV_RecListToCOMREcordList    ThirdPar,ThirdRec
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    Set Behavior = SmSession.ObjectStore.DefaultBehavior.Clone

```

```
RetCode = Err_NotFound
On Error GoTo ErrorTreat
'Check if CN_MANAGER header exists
If FirstRec.Headers.HeaderExists("CN_MANAGER") Then
    RetCode = Err_None
    'Get project manager's name
    ProjectManager = FirstRec.ValueAsString("CN_MANAGER",0)
    If (StrComp(Trim$(ProjectManager),"") = 0) Then
        MsgBox "You have to assign a manager to the project"
        RetCode = Err_Gen
    Else 'A project manager has been assigned.
        'Retrieving the Class id
        ClassId = FirstRec.ValueAsInteger(NM_CLASS_ID,0)
        ' Creating a new object
        Set NewWorkObject = SmSession.ObjectStore.NewObject(ClassId)
        ' Assign object attributes record to the new object
        NewWorkObject.AddAllAttributes
        ' Copy data from first record list to new object data
        NewWorkObject.Data.Copy FirstRec.GetRecord(0)
        ' Don't activate scripts to avoid infinite loop
        Behavior.InvokeScripts = False
        ' Insert object to DB
        NewWorkObject.InsertEx Behavior
        ' Pass Object_ID for new object to SmarTeam - Editor
        ThirdRec.AddHeader NM_OBJECT_ID,SIZE_OBJ_ID, sdtInteger
        ThirdRec.ValueAsInteger(NM_OBJECT_ID,0) = NewWorkObject.ObjectId
        ThirdRec.AddHeader "CN_DESCRIPTION", 71, 1
        ThirdRec.ValueAsString("CN_DESCRIPTION", 0) =
FirstRec.ValueAsString("CN_DESCRIPTION", 0)+ " additional text"
        CONV_COMRecListToREcordList ThirdRec, ThirdPar
    End If
End If
InstdAdd = RetCode
Exit Function
ErrorTreat:
    InstdAdd = Err.Number
    MsgBox "Error : " + Err.Description
    On Error GoTo 0
End Function
```

## After Add

This script is designed for the After Add script hook for the class Users (SmDemo Database). It is applicable to adding a new employee to the database. It sends a SmartMessage notifying user about the new user.

```
Declare Sub CONV_RecListToCOMREcordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRecList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmRecList.SmRecordList, ByRef RecList As Long)
Function AfterAdd(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim User As SmApplic.ISmObject
    Dim NewUser As SmApplic.ISmObject
    Dim MessageStore As SmartMessages.SmMessageStore
    Dim Message As SmartMessages.SmMessage
    Dim UserClassId As Integer
    Dim FirstRec As Object
    Dim SmSession As SmApplic.SmSession
    Dim SimpleQuery As SmApplic.ISmSimpleQuery

    On Error GoTo HandleError

    Set SmSession=SCREXT_ObjectForInterface(ApplHndl)
    CONV_RecListToCOMREcordList      FirstPar,FirstRec
    UserClassId = FirstRec.ValueAsSmallInt("CLASS_ID",0)
    Set SimpleQuery = SmSession.ObjectStore.NewSimpleQuery
    SimpleQuery.SelectStatement = "select CLASS_ID,OBJECT_ID from USERS
where LOGIN = " + chr$(39) + "joe" + chr$(39)
    SimpleQuery.Run
    Set User =
SmSession.ObjectStore.ObjectFromData(SimpleQuery.QueryResult.GetRecord(0),
true)

    Set NewUser =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0), true)
    Set MessageStore =
SmSession.GetService("SmartMessages.SmMessageStore")
    Set Message = MessageStore.NewSmartMessage
    Message.AddRecipient User.ObjectId, mrTo
    Message.AddRecipient SmSession.UserMetaInfo.UserId, mrFrom
    Message.Subject = "New user added"
    Message.Body = "The user " + NewUser.Data.ValueAsString("LOGIN") + "
was added to SmProject. Please assign him to the QA group."
    Message.Send
    Exit Function
```

```
HandleError:
    MsgBox Err.Description
End Function
```

## **Update**

This script is performed when updating an object in the database by the SmarTeam Update operation.

The script can be performed on link objects as well as on regular objects. For example, you can attach a script to the operation of updating a tree link object in class “Project Trees”. When you perform an Update operation in the SmarTeam view, two objects are updated automatically: the regular and the corresponding Link object. The script is activated twice: once for each update.

### **Script Execution Timing**

The following table shows the timing of script hook execution relative to the modification of the object in the database.

Stage	Timing
<b>Before</b>	The script is executed before the physical object’s modification in the database.
<b>After</b>	The script is executed after the physical object’s modification in the database.
<b>InsteadOf</b>	The script is executed instead of the system’s update operation. The system expects the script to execute the update operation.

### **Hook Timing in the SmarTeam Update Operation**

This section describes the hook timing relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the Update script hooks.

The Update script hooks execute with the following timing:

- User invokes Update on an object on the SmarTeam view

- Profile Card hooks occur, if set. See section Script Execution Timing .

- User clicks OK on the Update screen

  - If the user makes no changes on the screen, none of the update hooks occur

- Before Update hook** occurs

The input data is updated in the database – by SmarTeam – Editor or by the **InsteadOf Update hook** operation

**After Update hook** occurs

The Update screen is removed and changes caused by the hook scripts are displayed on the SmarTeam view window.

**Aborting:**

If the operation is aborted by returning Err\_Gen in the Before Update hook, the Update screen is left as is and no message is displayed by the system.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before Update operation script hook.

#### **Arguments**

<b>Argument</b>	<b>Description</b>
ApplHndl	Input. See Table 1.
Operation	‘UPDATE’
FirstPar	Input.  The attributes passed depend on the kind of object being updated. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Output new or changed object attribute values. See the section Outputting Object Attributes from a Script.
Return value	See Table 1. You can cause the Update operation to be aborted by assigning Err_Gen as the return value.

---

### **After Hook**

The following table describes the arguments passed in the After Update operation script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	‘Update’
FirstPar	Input.  The attributes passed depend on the kind of object being updated. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Output new or changed object attribute values. See the section Outputting Object Attributes from a Script.
Return value	See Table 1.

---

**InsteadOf Hook**

The following table describes the arguments passed in the InsteadOf Update operation script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1..
SelectOp	‘Update’
FirstPar	Input.  The attributes passed depend on the kind of object being updated. See the sections Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Output new or changed object attribute values. See the section Outputting Object Attributes from a Script.
Return value	See Table 1.

---

## ***Examples***

### **Before Update**

This script outputs in ThirdPar a description of the object as the component name input in FirstPar.

```
'Converts old API pointer on record list to COM object ISmRecordList
Declare Sub CONV_RecListToComRecordList Lib "Smtdm32" (ByVal RecList As
Long, ByRef COMRecList As ISmRecordList)

' Converts COM Object ISmRecordList to pointer on record list
Declare Sub CONV_ComRecListToRecordList Lib "Smtdm32" (ByVal COMRecList As
ISmRecordList, ByRef REcList As Long)

Function AddDescription(ApplHndl As Long, Sstr As String, FirstPar As
Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim ThirdRec As Object

    ' Convert pointer to COM object SmSession
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert input parameter to COM object
    CONV_RecListToComRecordList FirstPar,FirstRec
    CONV_RecListToComRecordList ThirdPar,ThirdRec

    FirstRec.PrintToFile "Example input record list" , "C:\Example.txt"
    ' ThirdRec.AddHeader "CN_DESCRIPTION", 71, 1
    ' ThirdRec.ValueAsString("CN_DESCRIPTION", 0) =
    FirstRec.ValueAsString("CN_COMPONENT_NAME", 0)
    CONV_ComRecListToRecordList ThirdRec, ThirdPar
    AddDescription = Err_None
End Function
```

### ***Delete***

This script is performed when deleting an object in the database by the SmarTeam Delete operation.

The script Delete is performed on link objects as well as on regular objects. For example, you can attach a script to the operation of deleting a tree link object in class “Project Trees”.



## ***Script Execution Timing***

Stage	Timing
<b>Before</b>	The script is executed before the physical object's deletion from the database
<b>After</b>	The script is executed after the physical object's deletion from the database.
<b>InsteadOf</b>	The script is executed instead of the system's delete operation. The system expects the script to execute the delete operation of the object.

## **Hook Timing in the SmarTeam Delete Operation**

This section describes the hook timing relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the Delete script hooks.

User invokes Delete on an object on the SmarTeam view

The confirm Delete dialog box is displayed

### **Before Delete hook occurs**

Get dialog box: Found reference to the Object "..." in Class "...". Would you like to proceed with Delete operation anyway?

The input data is deleted in the database – by SmarTeam – Editor or by the **InsteadOf Delete hook** operation

### **After Delete hook occurs**

Steps 2-6 are repeated for each object to be deleted.

The object is removed from the SmarTeam view window.

### **Aborting:**

If the operation is aborted by returning Err\_Gen in the Before Delete hook, the SmarTeam View is left as is and no message is displayed by the system.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before Delete operation script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	'Delete'
FirstPar	Input.  The attributes passed depend on the kind of object being deleted. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Not used
Return value	See Table 1.  You can cause the Delete operation to be aborted by assigning Err_Gen as the return value.

**After Hook**

The following table describes the arguments passed in the After Delete operation script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	'Delete'
FirstPar	Input.  The attributes passed depend on the kind of object being deleted. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Not used
Return value	See Table 1.

**InsteadOf Hook**

The following table describes the arguments passed in the InsteadOf Delete operation script hook.

## Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	'Delete'
FirstPar	Input.  The attributes passed depend on the kind of object being deleted. See the section Object Attributes Input to a Script and Link Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Not used
Return value	See Table 1.  You can abort the deletion operation by assigning Err_Gen, Err_Refuse as the return value.

---

## Examples

### Before Delete

This script is designed for the Before Delete script hook for the class SolidWorks Assembly (SmDemo). It is applicable to deleting an Assembly from the database. It checks if an existing SolidWorks Drawing is linked to the Assembly to be deleted. If so, it aborts the Delete operation.

```
Declare Sub CONV_RecListToCOMRecordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRecList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmRecList.SmRecordList, ByRef RecList As Long)
Function BeforeDelete(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim Assembly As SmApplic.ISmObject
    Dim Drawings As SmApplic.ISmObjects
    Dim QueryDef As SmApplic.ISmQueryDefinition
    Dim LinkClassId As Integer
    Dim DrawingClassId As Integer
    Dim FirstRec As Object
    Dim SmSession As SmApplic.SmSession

    On Error GoTo HandleError
```

```
Set SmSession=SCREXT_ObjectForInterface(ApplHndl)
CONV_RecListToCOMRecordList      FirstPar,FirstRec
Set Assembly =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),True)
Set QueryDef = SmSession.ObjectStore.NewQueryDefinition()
LinkClassId = 0
DrawingClassId = SmSession.MetaInfo.SmClassByName("SolidWorks
Drawing")
QueryDef.Roles.Add DrawingClassId, "S"
QueryDef.Roles.Add LinkClassId, "L"

Set Drawings = Assembly.RetrieveRelations(QueryDef)
If Drawings Is Nothing
    BeforeDelete = Err_None
    Exit Function
End If
If Drawings.Count = 0 Nothing
    BeforeDelete = Err_None
    Exit Function
End If
BeforeDelete = Err_Gen
Exit Function
HandleError:
MsgBox Err.Description
End Function
```

## **Before Delete**

This script is designed for the Before Delete script hook for the class SolidWorks Assembly (SmDemo). It cancels the delete operation if the object is connected by a hierarchical link to the parent object.

```
Function BefDel(ApplHndl As Long, Sstr As String, FirstPar As Long,
SecondPar As Long, ThirdPar As Long ) As Integer

    Dim WorkObject As SmApplic.ISmObject
    Dim Parents As SmApplic.ISmObjects
    Dim FirstRec As Object
    Dim SmSession As SmApplic.SmSession
    Dim QueryDef As ISmQueryDefinition

    CONV_RecListToCOMREcordList FirstPar,FirstRec
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Get Sm object to delete
    Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),false)
    If WorkObject Is Nothing Then
        BefDel = Err_NotFound
        Exit Function
    End If
    ' Retrieve Sm object parents
    Set Parents = WorkObject.RetrieveParents(QueryDef)
    ' Check if parents exist for given Sm Object
    If Parents Is Nothing Or Parents.Count = 0 Then
        BefDel = Err_Gen
    Else
        BefDel = Err_None
    End If
End Function
```

---

## Scripts for Lifecycle Operations

### Overview of Lifecycle Script Hooks

Script hooks are provided for lifecycle operations such as Check In or Release. To give you greater control, separate script hooks are provided for each operational step of the lifecycle operation such as initiating the lifecycle operation, selecting objects on the lifecycle screen and clicking OK on the lifecycle screen. As in the database operation hooks, each lifecycle hook can have the three stages *Before*, *After* and *InsteadOf* operation.

In general, SmarTeam lifecycle operations are performed on a group of objects at the same time. This clearly happens when you invoke a lifecycle operation on a group of objects you selected on the SmarTeam view, but it can also happen even if you select only one object on the SmarTeam view. Then, depending on the settings, SmarTeam may retrieve and add objects that are related to the one you selected on the SmarTeam view and perform the lifecycle operation on the group.

Accordingly, lifecycle hooks are divided into three categories:

- **Individual lifecycle hooks (GUI dependent)** –These hooks relate to the lifecycle operation on an individual object even if it is part of a group and includes setting lifecycle task attributes by user input screens.
- **Individual lifecycle hooks** –These hooks relate to the lifecycle operation on an individual object even if it is part of a group and includes setting lifecycle task attributes independently of user screens.
- **Group lifecycle hooks** – These hooks give the user access to internal information used by SmarTeam – Editor to execute the lifecycle operation on a group of objects. They are independent of user input screens.

**Note:** After performing a lifecycle operation, the status of file properties and of File Catalog is updated on completion of a group lifecycle operation. Therefore, this change can be verified only in the *After LifeCycle Stage 2* hook or *After LifeCycle Stage 1* hook. **Figure 11** shows the lifecycle script hooks.

*Figure 11 Script Hooks for Life-Cycle Operations*

Operation in Script Maintenance	Before	After	InsteadOf
<b>Script Hooks for Individual Life-Cycle Operations (GUI dependent)</b>			
Load Life-Cycle Screen	X	X	
On Life-Cycle click CheckOut	X		

---

**Scripts for SmarTeam Operations**

On Life-Cycle click CheckIn	X		
On Life-Cycle click Release	X		
On Life-Cycle click New Release	X		
On Life-Cycle click Obsolete	X		
On Life-Cycle click Copy File	X		

Script Hooks for Individual Life-Cycle Operations			
Check Out	X	X	X
Undo Check Out	X	X	X
Check In	X	X	X
Release	X	X	X
NewRelease	X	X	X
Obsolete	X	X	X
Copy File	X	X	X

Script Hooks for Group Life-Cycle Operations			
Life-Cycle Stage 1	X	X	X
Life-Cycle Stage 2	X	X	X

### ***Lifecycle Operation Sequence***

In order to understand the location of the lifecycle script hooks in the software, you need to understand the way SmarTeam – Editor performs a lifecycle operation such as Check In or Check Out. The following table shows how a lifecycle operation is broken down into the SmarTeam system responses to individual user actions.

User Action	SmarTeam – Editor Action
Selects objects on SmarTeam View for LC operation	Shows Profile Card for last selected object
Performs LC Operation	Displays the LC screen for the LC operation. The LC screen shows:  For the advanced LC screen--the objects selected and may also display objects related to the ones selected.  For light LC screen – no objects are displayed.
Selects objects on LC screen	Displays Profile Card for selected object
Changes appropriate fields on LC screen	
Clicks OK	Builds trees connected to objects  Searches for links and CFO  Checks consistency  Performs LC operation on individual objects in a loop.

---



**Note:** The above refers to a high-level, GUI-based process carried out by SmarTeam – Editor. You can use an API function from within a script to perform group lifecycle operations (see `SmSessionUtil.ExecuteOperationOnTrees`, `ExecuteOperationOnObjectTree`). For an example using `ExecuteOperationOnTrees`, see Examples for User-Defined commands.

## ***Timing of Life-Cycle Script Hook Events***

It is important to understand the timing of the lifecycle script hook events, that is, at which stages of the lifecycle operation they occur.

Table 8 shows the timing of script hook events for a typical lifecycle operation. The user/system actions and resulting events are shown in the sequence that they occur, together with the type of input and output information to the script hooks for each event. The input and output information is presented in more detail in later sections.

*Table 8 Timing of Life-Cycle Script Hooks*

User/system action	Script Hook	FirstPar	SecondPar	ThirdPar
User: Performs LC Operation on objects selected in View (LC screen not set to Light)	Before LoadLCScreen	Input selected objects	None	Input/output default tasks
System: After related objects are retrieved but before LC screen is displayed	After LoadLCScreen	Input selected and retrieved objects	Output object tasks	Input/output default tasks
User: Selects object on LC Screen and before object information is displayed on LC screen	Click [LC Operation]	Input object task attributes	Input/output tasks for one object	None
User: Clicks OK System: Before all checking is started on objects	Before LifeCycleStage1	Input objects on LC screen	Input links between objects in FirstPar (not for Light LC screen)	Input/output default (for objects subsequently retrieved during checking) and object tasks

System: Builds trees  
connected to objects

Searches for links  
and CFO

Checks consistency

System: After checks  
are complete and  
before the physical  
operations on the  
database

Before  
LifeCycleStage2

Input objects  
on LC screen

Input: Links  
between objects  
in FirstPar

Input/ output,  
default tasks  
and object  
tasks

System: Start loop on  
objects

System: Begins LC  
operation on an  
individual object.

Before [LC Operation]

Input object

Input/output  
tasks for one  
object

Output object  
attributes

System: Makes  
change in database  
for an individual  
object.

Instead\_[LC Operation]

Input object

Input/output  
tasks for one  
object

Output  
attributes for  
new object

System: Finishes LC  
operation on an  
individual object.

After [LC Operation]

Input object

Input/output  
tasks for one  
object

Output object  
attributes

System: End of loop

System: After all  
operations on the  
database are done

After LifeCycle2

Input objects

Input: Links  
between objects  
in FirstPar

Input/ output,  
default tasks  
and object  
tasks

After LifeCycle1

Input objects

Input: Links  
between objects  
in FirstPar

Input/ output,  
default tasks  
and object  
tasks

**Warning:** It is recommended to use the group lifecycle hooks only when all possible effects of the script are taken into account. Scripts attached to these hooks are executed during sensitive database transaction processing. Inappropriate changes such as adding an invalid object to the list of objects on which the operation is to be executed, can cause severe inconsistency problems.

### **Timing of Check In/Keep Checked Out**

For the Check In operation with the Keep Checked Out option set, SmarTeam – Editor performs the Check In operation but does not actually perform a Check Out operation. Instead, the Check Out operation is simulated by creating a new version of the object in the state it would be if it were actually checked out after being checked in.

Accordingly, the Check In and Check Out hooks occur in the following sequence:

Before Check In

Before Check Out

InsteadOf Check In

After Check In

After Check Out

Note that at step 2 the object being checked in still has the status of Checked Out.

Note that the default settings are applied to the new checked out version, however you can use the Before Check Out hook at stage 2 to override the default setting. For example, if the default is to copy links to a checked-out object, you can use the script to prevent copying the links.

### **Script Hook Timing for Light Life-Cycle Screen**

When the light lifecycle screen is in effect, the LoadLifeCycle screen script hooks work as follows:

The effect of the Before and After LoadLifeCycleScreen hooks are the same.

It is sufficient to attach one of them; if both are attached, only the Before LoadLifeCycleScreen will operate.

You cannot abort the LoadLifeCycleScreen operation with this hook.

No parameters are passed to these hooks

## ***Individual Lifecycle Task Attributes***

This section describes the various task attributes, which you can use in the lifecycle hooks to influence the individual lifecycle process. For information on how to pass these attributes to a script, see the section *Passing Lifecycle Task Information to Script Functions*. For information about task attributes for group lifecycle operations, see the section *Group Lifecycle Task Attributes*.

For each task attribute, the lifecycle operations for which it is relevant is shown.

### ***Applicable Hooks***

A certain task attribute can be relevant for only a certain part of the lifecycle. Consequently, only script hooks in that part of the lifecycle will accept the task attribute. Other lifecycle script hooks will ignore it. Thus, for each task attribute, it is important to know for which script hooks in the life cycle it is relevant.

In the sections below, the script hooks for which the task attribute is relevant is shown in the “Applicable Hooks” entry. For each entry a range of applicable hooks is denoted. This range is part of the sequence of all script hooks that can occur (if set) during a typical lifecycle operation, as shown below (see also Table 8).

In general, the individual task attributes are relevant to hooks in the range: Before LoadLCScreen – Before [LC Operation], whereas the group task attributes are relevant to the range Before LifeCycleStage1 – Before LifeCycleStage2.

Before LoadLCScreen

After LoadLCScreen

Click\_[LC Operation]

Before LifeCycleStage1

Before LifeCycleStage2

Before [LC Operation]

Instead\_[LC Operation]

After [LC Operation]

After LifeCycle2

After LifeCycle1

Note: If you set a task attribute in a script hook which is not one of the hooks denoted in the “Applicable Hooks” entry for that attribute, the value will be ignored by SmarTeam – Editor.

### ***NM\_OBJECT\_ID***

<b>Attribute Name</b>	“OBJECT_ID”
<b>Data Type</b>	sdtObjectIdentifier
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The Object\_ID of the object under the current operation. Internal reference.

### ***NM\_CLASS\_ID***

<b>Attribute Name</b>	“CLASS_ID”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The Class\_ID of the object under the current operation. Internal reference.

### ***NM\_OPER\_ID***

<b>Attribute Name</b>	“OPERATION_ID”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The Operation ID of the current operation. Internal reference.

### ***NM\_EFFECTIVE\_FROM***

<b>Attribute Name</b>	“EFFECTIVE_FROM”
<b>Data Type</b>	sdtEffectiveDateFrom
<b>Lifecycle Operations</b>	All except Check Out and New Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The Effective From date of the revision of the object after the current operation is complete.

Appears on SmarTeam – Editor in: Lifecycle screen/Effectivity tab/Effective from.

### ***NM\_EFFECTIVE\_UNTIL***

<b>Attribute Name</b>	“EFFECTIVE_UNTIL”
<b>Data Type</b>	sdtEffectiveDateUntil
<b>Lifecycle Operations</b>	All except Check Out and New Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The Effective Until date of the revision of the object after the current operation is completed.

Appears on SmarTeam – Editor in: Lifecycle screen/Effectivity tab/Effective Until.

### ***NM\_FILE\_NAME***

<b>Attribute Name</b>	“FILE_NAME”
<b>Data Type</b>	sdtChar/MAX_FILE_NAME_LEN
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

#### **Description**

The destination file name associated with the object under the current operation.

Appears on SmarTeam – Editor in: Lifecycle screen/ General tab/File name.

## ***NM\_DIRECTORY***

<b>Attribute Name</b>	“DIRECTORY”
<b>Data Type</b>	sdtChar/MAX_DIR_NAME_LEN
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

The name of the directory in which NM\_FILE\_NAME is to be located.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Destination Directory.

## ***NM\_VAULT\_OBJ\_ID***

<b>Attribute Name</b>	“VAULT_OBJECT_ID”
<b>Data Type</b>	sdtObjectIdentifier
<b>Lifecycle Operations</b>	Check In, Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Internal reference to the destination vault object.



## ***NM\_ REVISION***

<b>Attribute Name</b>	“REVISION”
<b>Data Type</b>	sdtChar
<b>Lifecycle Operations</b>	All, except for Copy File
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

The destination revision of the object under the current operation.

Appears on SmarTeam – Editor, according to lifecycle operation, in:

Check In: SmarTeam View Profile Card/Revision tab /Revision (after CheckIn is complete)

Check Out, New Release, Obsolete: Lifecycle screen/General tab/Next revision

Release: Lifecycle screen/General tab/Revision

when User-defined revision is clicked and if Tools/Administrator Options/Lifecycle Options/ General/ “Enable user-defined revisions” is set

## ***NM\_ DSC\_NOTES***

<b>Attribute Name</b>	“DSC_NOTES”
<b>Data Type</b>	sdtChar/MAX_DIR_NAME_LEN
<b>Lifecycle Operations</b>	All, except for Copy File
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Descriptive notes concerning the operation.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Comment.

## ***NM\_PHASE***

<b>Attribute Name</b>	"PHASE"
<b>Data Type</b>	sdtObjectIdentifier
<b>Lifecycle Operations</b>	All, except for Copy File
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

The name of the phase for the revision of the object under the current operation.

The possible phases are:

Default

Preliminary Design

Design

Prototype

Production

Retrofit

The user can change, add or delete phase values.

Appears on SmarTeam – Editor in: Lifecycle screen/Effectivity tab/Phase.

The following code fragment shows how to get the identifier for a given phase and enter a desired value of the phase into a record list.

```
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
' Get phase lookup table class id
PhaseClassId = SmSession.MetaInfo.SmClassByName("Phase").ClassId
' Get lookup object by unique name
Set Phase =
SmSession.ObjectStore.GetSmLookUpByUniqueName(PhaseClassId,"Design")
' Add attribute to result record list
SecondRec.AddHeader NM_PHASE, SIZE_OBJ_ID, TDMT_OBJ_ID
' Set phase stage on lifecycle screen as Design
SecondRec.ValueAsInteger(NM_PHASE,0) = Phase.Id
```

## ***NM\_ TSK\_ KEEP\_ LOCAL\_ COPY***

<b>Attribute Name</b>	“TDM_KEEP_LOCAL_COPY ”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check In, Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Whether to keep a read-only copy of the file located in the working directory for the object currently being checked in or released.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Keep local file.

### **Values**

<b>Value</b>	<b>Description</b>
0	Don't keep local copy (default)
1	Keep local copy

## ***NM\_ TSK\_ KEEP\_ CHECKEDOUT***

<b>Attribute Name</b>	“TDM_KEEP_CHECKEDOUT”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check In, Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Whether to keep the file checked out for modifications after performing the Check In operation.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Keep checked out.

**Values**

Value	Description
0	Don't keep file checked out (default)
1	Keep file checked out

**NM\_ TSK\_NOCREATE\_LOCAL\_COPY**

<b>Attribute Name</b>	"TDM_NOCREATE_LOCAL_COPY "
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check Out, NewRelease
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

**Description**

Whether to perform the Check Out or NewRelease operation without copying the file from the vault to the working directory.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Do not get the file from the vault.

**Values**

Value	Description
0	Copy the file to the working directory on Check Out or New Release (default)
1	Don't copy the file to the working directory on Check Out or New Release

## ***NM\_LFCYC\_NEW\_BRANCH***

<b>Attribute Name</b>	“NEW_BRANCH”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check Out, New Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Whether to create parallel branches of a revision based on the same file.

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/ Create new branch.

### **Values**

<b>Value</b>	<b>Description</b>
0	Don't create a new branch on Check Out or New Release (default)
1	Create a new branch on Check Out or New Release

## ***NM\_LFCYC\_CHECKIN\_MODE***

<b>Attribute Name</b>	“CHECKININ_MODE”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check In, Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Specifies the revision of an object to use for the Check In or Release operation.

Possibilities are:

The operation is performed with current revision of the object

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/ Current Revision

The operation is performed with the previous (parent) revision of the object

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/Replace Previous Revision

The operation is performed with the user defined revision of the object

Appears on SmarTeam – Editor in: Lifecycle screen/General tab/User defined revision

The user-defined revision box is shown only if the Tools/Administrator Options/Lifecycle Options/General/ “Enable user-defined revisions” is set

### Values

Value	Description
LFCYC_WorkRev (= 1)	The checked out object is checked in, keeping the same revision. No previous object is deleted.
LFCYC_PrevRev (= 2)	The checked out object is copied to the immediately previous checked in revision. The checked in revision number is used and the checked out object is deleted.
LFCYC_UserRev (= 3)	The checked out object replaces the checked in revision selected by the user. The checked in revision number is used. The replaced version is considered to be the latest.

## ***NM\_ LOGICAL\_LINK\_COPY***

<b>Attribute Name</b>	"LOGICAL_LINK_COPY"
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check In, Release, Check Out, NewRelease
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Specifies whether the logical links of the current object under operation should be copied to the destination version. For example, when you check out a Document, which is linked to an Assembly – you would establish new link between the checked out Document's revision and the Assembly.

Possibilities are:

Do not copy the logical links

Copy the logical links

Appears on SmarTeam – Editor in: Lifecycle screen/Options/Copy general links

This attribute overrides the settings:

Tools/Administrator Options/Lifecycle Options/ Into Vault/ "Copy general links on Check In/Release"

Tools/Administrator Options/Lifecycle Options/ Out Of Vault/ "Copy general links on Check Out/New Release"

### **Values**

<b>Value</b>	<b>Description</b>
LCS_No	do not copy logical links
LCS_Yes	copy logical links (Default)

## ***NM\_LINKS\_TO\_SONS\_COPY***

<b>Attribute Name</b>	"LINKS_TO_SONS_COPY"
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check Out, New Release  Also applies to the checked out version in the operation Check In/Keep Checked Out
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Specifies whether the hierarchical links of the current object under operation should be copied to the destination version. For example, when you check out a Document that has children – you would establish new hierarchical links between the checked out Document's revision and its children.

Possibilities are:

Do not copy the hierarchical links

Copy the hierarchical links

Appears on SmarTeam – Editor in: Life-cycle screen/Options/ Copy tree links to children

This attribute overrides the settings:

Tools/Administrator Options/Lifecycle Options/ Out Of Vault/ "Copy tree links to children on Check Out "

### **Values**

<b>Value</b>	<b>Description</b>
LCS_No	Do not copy hierarchical links
LCS_Yes	Copy hierarchical links (Default)



## ***NM\_ FILE\_OVERWRITE***

<b>Attribute Name</b>	"FIIE_OVERWRITE"
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	Check Out, New Release
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Specifies the behavior of the Check Out or New Release operation when a file with the same name as the one in the vault exists in destination directory. The attribute works for destination files that are either in-work (read/write) files or copied (read-only) files.

The possibilities are listed in the Values section below.

Appears on SmarTeam – Editor in: Lifecycle screen/Options/Replace local files on Check Out [For in-work files; For copied files]

This property overrides the setting:

Tools/Administrator Options/Lifecycle Options/ Out Of Vault/ "Replace local files on Check Out/New Release" [Yes, when not in work; Yes; No; Ask]

### **Values**

<b>Value</b>	<b>Description</b>
fowYes	Overwrite the file
fowRO_Yes	Overwrite the file if it's read only
fowNO	Do not overwrite the file
fowAnswer	Display dialog box to query whether to overwrite the file
fowRO_No	Don't overwrite if the file is read-only
fowYesAll	Overwrite the file, and if the same situation occurs in the subsequent Lifecycle operations for other files, overwrite these files too
fowNoAll	Don't overwrite the file, and if the same situation occurs in subsequent Lifecycle operations don't overwrite the files

---

fowCancel	Cancel the lifecycle operation if a file with the same name exists in the destination directory
fowNO_RW _RONewer	Don't overwrite if the file in destination directory is read-only or if it is has a later version than the one coming from the vault

---

## ***NM\_NOT\_CHECK\_AUTH***

<b>Attribute Name</b>	“NOT_CHECK_AUTH”
<b>Data Type</b>	sdtSmallInt
<b>Lifecycle Operations</b>	All
<b>Applicable Hooks</b>	Before LoadLCScreen – Before [LC Operation]

### **Description**

Specifies if the lifecycle operation checks for user authorization. This flag should be used according to the security practice of the organization.

### **Values**

<b>Value</b>	<b>Description</b>
LCS_No	Check for user authorization (Default).
LCS_Yes	Don't check for user authorization.

---

## ***Passing Lifecycle Task Information to Script Functions***

For hooks related to lifecycle task operations, task information is passed to and from a script function by means of the record list parameters SecondPar and ThirdPar. This task information is related to the object information passed in FirstPar. See Table 3 Record List Information Passed to and from Scripts for more information. This section discusses these parameters in detail. In this section, it is assumed that the Record List parameters have been converted to COM format (see Converting Procedural Parameters on page 9.) See section Individual Lifecycle Task Attributes for detailed information on the individual task attributes.

## Representing SmarTeam Tasks

The task attributes of a SmarTeam lifecycle task such as Check In are represented by an SmRecord object, which represents one record or row of an SmRecordList object.

An SmRecordList object can represent task attributes for a set of tasks. The headers of the columns of the SmRecordList matrix represent the task attributes and the SmRecord rows of the SmRecordList matrix represent the tasks. Each cell of a row contains the value of the task attribute denoted by the cell column's header. Figure 12 shows a SmRecordList object that represents n task attributes of m tasks. The sections Individual Lifecycle Task Attributes and Group Lifecycle Task Attributes describe available task attributes.

Figure 12 Task Attributes represented by an SmRecordList

	Task Attributes				
<b>Header:</b> Name Type Size	Object_ID 10 4	Class_ID 2 2	Oper_ID Type-3 Size-3	...	Attribute-n Type-n Size-n
<b>Task-1</b>	ObjectID-1	ClassID-1	Oper_ID -1	...	Value-n-1
<b>Task-2</b>	ObjectID-2	ClassID-2	Oper_ID -2	...	Value-n-2
<b>Task-3</b>	ObjectID-3	ClassID-3	Oper_ID -3	...	Value-n-3
...	...	...	...	...	...
<b>Task-m</b>	ObjectID-m	ClassID-m	Oper_ID -m	...	Value-n-m

The SmRecordList header cells always contain the triple:

attribute name

attribute data type, represented by its number

size of the attribute value in bytes.

## Relation between Tasks and Objects

In general, the tasks passed in SecondPar correspond to objects passed in FirstPar. The task attributes Object\_ID and Class\_ID in the SecondPar, point to the object in FirstPar for which the task is to be executed. In addition, the object attribute Oper\_ID of an object in the FirstPar refers to the task in SecondPar that operates on it. Figure 13 shows the Record Lists for FirstPar and SecondPar in such a way that the correspondence is emphasized. In this example, the task specified in the first row of SecondPar, for example, Check In, operates on the object represented by the first row of FirstPar, for example, Folder\_222.

Figure 13 Relation between Tasks and Objects

	Object Attributes				Task Attributes	
<b>Header:</b> Name Type Size	Oper_ID 2 2	...		<b>Header:</b> Name Type Size	Object_ID 10 4	...
<b>Object-1</b>	Oper_ID -1	...		<b>Task-1</b>	ObjectID-1	...
<b>Object-2</b>	Oper_ID -2	...		<b>Task-2</b>	ObjectID-2	...
<b>Object-3</b>	Oper_ID -3	...		<b>Task-3</b>	ObjectID-3	...
...	...	...		...	...	...
<b>Object-m</b>	Oper_ID -m	...		<b>Task-m</b>	ObjectID-m	...

FirstPar

SecondPar

## Passing Task Attributes as a Record List Parameter

As above, the `SmRecordList` object is dynamic. It can represent any subset of the task attributes for any number of tasks.

The dynamic property of the SmRecordList is utilized to pass task attribute information to a script function. In general, it is required to pass task attribute information only for the tasks involved in the SmarTeam lifecycle operation being executed.

Accordingly, the required tasks attributes and tasks are packed into a reduced SmRecordList and transferred to and from the script function as parameters. Figure 14 shows a reduced record list that has four task attributes and two tasks. This record list might be used to pass information to the script for a SmarTeam Check In operation on two folders.

Figure 14 Example of Task Attributes Record List

Task Attributes				
<b>Header:</b> Name	FILE_NAME	DIRECTORY	VAULT_OBJ_ID	REVISION
Type	50	50	1	4
Size	4	4	16	10
<b>Task1</b>	FileName1	C:\ST	2333	3.1
<b>Task2</b>	Filename2	C:\ST	2505	2.2

## Passing Default Task Attributes

For some hooks, default task attributes are passed to the script along with the task attributes. These are attributes for the current lifecycle operation and for tasks that are related to it. The default task attributes are normally passed in the ThirdPar.

By changing an attribute of a default task in ThirdPar, you change the corresponding attribute values for that task when it operates on objects in FirstPar. For example, if you change the Effective\_Until date in the default task for Check In, the new date will apply to all objects being checked in.

In addition, the default attributes provide a reference to the script writer as to what the default values are in case they are missing or changed in a task record.

Depending on the specific hook, default task attributes may be the only contents of ThirdPar as in Figure 15, in the hook Before LoadLifeCycleScreen, or the default task attributes may be together with task attributes as in Figure 16, in the hook BeforeLifeCycle1.

Figure 15 Default Task Attributes in ThirdPar

Task Attributes					
<b>Header:</b> Name	Object_ID	Class_ID	Oper_ID	Attribute-4	...
Type	10	2	2	Type-4	
Size	4	2	2	Size-4	
<b>DefaultTask-1</b>			Oper_ID-d1	Value-4-d1	...
<b>DefaultTask-2</b>			Oper_ID-d2	Value-4-d2	...
<b>DefaultTask-3</b>			Oper_ID-d3	Value-4-d3	...

*Figure 16 Default and Object Task Attributes in ThirdPar*

	Task Attributes				
<b>Header:</b> Name Type Size	Object_ID 10 4	Class_ID 2 2	Oper_ID 2 2	Attribute-4 Type-4 Size-4	...
<b>DefaultTask-1</b>			Oper_ID-d1	Value-4-d1	...
<b>DefaultTask-2</b>			Oper_ID-d2	Value-4-d2	...
<b>DefaultTask-3</b>			Oper_ID-d3	Value-4-d3	...
<b>Task-1</b>	ObjectID-1	ClassID-1	Oper_ID-1	Value-4-1	...
<b>Task-2</b>	ObjectID-2	ClassID-2	Oper_ID-2	Value-4-2	...
<b>Task-3</b>	ObjectID-3	ClassID-3	Oper_ID-3	Value-4-3	...
...	...	...	...	...	...
<b>Task-m</b>	ObjectID-m	ClassID-m	Oper_ID-m	Value-4-m	...

Note that the object identifier attributes Object\_ID and Class\_ID are empty in the default tasks. The reason is that the default tasks do not refer to a specific object but rather provide default information for any object on which that task operates. You use the Oper\_ID attribute to identify the type of operation.

Usually, at least the default task for the current operation is passed. However, frequently more than one default task is passed. Multiple default tasks are passed, for example, when the user performs a Check Out operation on a group of objects. SmarTeam – Editor takes into account that it may not be appropriate to perform a Check Out operation on all of these objects. Other lifecycle operations like New Release or Copy File may need to be performed instead. Accordingly, the default tasks for these operations are included in the ThirdPar Record List.

### Identifying Oper\_ID

As explained above, the Oper\_ID attribute lets you identify the default task record. This section explains how to identify which lifecycle operation is associated with the integer Oper\_ID.

The list of possible lifecycle operations is:

Operation Name	Operation String	Description
NM_OPER_CHECKIN	'CHECKIN'	Check In
NM_OPER_CHECKOUT	'CHECKOUT'	Check Out
NM_OPER_APPROVE	'APPROVE'	Approve
NM_OPER_NEWRELEASE	'NEWRELEASE'	New Release
NM_OPER_FREEZE	'FREEZE'	Freeze
NM_OPER_COPY_FILE	'CopyFile'	Copy File
NM_OPER_NO_OPER	'NoOperation'	No operation
NM_OPER_NOT_ALLOWED	'OperNotAllowed'	Operation not allowed

In order to identify the operation associated with the Integer value Oper\_Id proceed as follows:

**For COM APIs:**

Use the following function to get the values of the integer OperId for the possible operations you are dealing with in the script, and then compare them with the value of OperId in the record list to establish its identity:

The property `ISmMetaInfo.SmOperationByName[OperationName]` returns an object of type `ISmOperation`, which has property 'Id'

**Example**

Dim CheckOutId, CheckInId as integer

```
CheckOutId = Session.MetaInfo.SmOperationByName(NM_OPER_CHECKOUT).Id
CheckInId = Session.MetaInfo.SmOperationByName(NM_OPER_CHECKIN).Id
'now compare CheckOutId and CheckOutId with the OperId value in the Record
List
```

## **Operation Code**

As a rule, in the lifecycle script hooks described above, the `Oper_ID` attribute is passed among both the object attributes in `FirstPar` and the task attributes in `SecondPar`.

There is one exception to this rule: In the Lifecycle Stage 2 script hooks, in the object attributes in the `FirstPar`, the *Operation Code* attribute is passed instead of `Oper_ID`. In the task attributes of the `SecondPar`, the `Oper_ID` is passed as before.

The association of the values of the Operation Code attribute with the lifecycle operations is shown in Table 9. Note that you do not need to use an intermediate function to determine the operation with which they are associated as you do with the `Oper_ID` parameter described above.



*Table 9 Operation Code Values*

Operation Name	Operation Code	Description
OPCHECKOUT	0	CHECKOUT
OPNEWREL	1	NEWREL
OPREGISTR		obsolete
OPCHECKIN	3	CHECKIN
OPAPPROVE	4	APPROVE
OPFREEZE	5	FREEZE
OPCOPYFILE	6	COPYFILE
OPNOOP	7	NOOP
OPNOTALLOWED	8	NOTALLOWED
OPDUMMY	9	DUMMY
OPSECONDARYCOPY	10	SECONDARYCOPY
OPLOCKCOPY	11	LOCKCOPY
OPUNLOCK	12	UNLOCK

---

## Scripts for Individual GUI-Based Lifecycle Operations

This section describes script hooks for GUI-based lifecycle operations including:

- Load Lifecycle Screen
- Click LifeCycleOperation

### ***Load Lifecycle Screen***

The Load Lifecycle Screen script hook occurs whenever a user invokes one of the lifecycle operations, such as Check In or Release, from a SmarTeam View window. There are two script hooks defined for this event corresponding to the event stages Before and After:

Before LoadLifeCycleScreen

After LoadLifeCycleScreen

Both of these hooks occur before the lifecycle screen is actually displayed. They enable you to set up task attribute values for lifecycle tasks operating on specific objects, or to set up default task attributes for all objects undergoing a lifecycle operation. As a result of the script action, the new task attribute values appear in the lifecycle screen when it is displayed.

This hook is useful for customizing the data fields in the lifecycle screens and providing default values in them.

See Table 8 for a comparison between the Before LoadLifeCycleScreen and After LoadLifeCycleScreen hooks -- when each occurs and which attributes are passed to each. See there also the timing of these hooks relative to other lifecycle hooks.

**Note:** These script hooks are only relevant when you work in the Advanced Lifecycle screen (not the Light Lifecycle screen).

## ***Script Execution Timing***

Stage	Timing
<b>Before</b>	Executed when user invokes a lifecycle command from a SmarTeam view but before the system retrieves objects related to the objects selected on the view.
<b>After</b>	Executed after the system retrieves the objects related to the objects selected on the view but before the lifecycle screen is displayed.
<b>InsteadOf</b>	Not relevant

## **Hook Timing in the SmarTeam Load Lifecycle Screen Operation**

This section describes the hook timing relative to the SmarTeam GUI operations.

User invokes a lifecycle operation on an object on the SmarTeam view

**Before LoadLifeCycleScreen hook** occurs

**After LoadLifeCycleScreen hook** occurs

Click\_[LC-Operation] script hooks for the specific lifecycle operation, if set, occur

The lifecycle screen for the lifecycle operation is displayed.

### **Aborting:**

You cannot abort the Load Lifecycle Screen operation using these hooks.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before LoadLifeCycleScreen script hook.

By changing the attribute of a default task in ThirdPar, you can change the corresponding attribute values for that task when it operates on objects in FirstPar. Specific task values set in SecondPar in the After hook override default values set in ThirdPar in both the Before and After hooks.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Name of current lifecycle operation: Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input containing list of objects that were selected on the SmarTeam View.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Not used
ThirdPar	Inputs and outputs default task attributes for each task relevant to the current lifecycle operation including the Operation_ID attribute of the task. See Figure 15.  For more information see Passing Default Task Attributes on page 85.  The following default task attributes are always passed: OBJECT_ID CLASS_ID OPERATION_ID
Return value	See Table 1.  This operation cannot be aborted by assigning Err_Gen, Err_Refuse to the script return value.

**After Hook**

The following table describes the arguments passed in the After LoadLifeCycleScreen script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Name of current lifecycle operation: Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input containing list of objects to be loaded to the screen, including objects selected on the SmarTeam view and related objects that were retrieved by SmarTeam – Editor.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Create and output task attributes for the lifecycle task being executed on objects in FirstPar. See Figure 12.  For more information see Representing SmarTeam Tasks on page 83.
ThirdPar	Inputs and outputs default task attributes for each task relevant to the current lifecycle operation, including the Operation_ID attribute of the task. See Figure 15.  The following default task attributes are passed: See Before Hook.  For more information see Passing Default Task Attributes on page 85.
Return value	See Table 1.  This operation cannot be aborted by assigning Err_Gen, Err_Refuse to the script return value.

---

## ***Click LifeCycleOperation***

This hook is executed while browsing in the LifeCycleOperation view before the current object task attributes values are displayed, enabling you to set attributes for the selected object. The Click LifeCycleOperation script hooks exist for the following lifecycle operations:

Check Out

Undo Check Out

Check In

Release

New Release

Obsolete

Copy File

Since the behavior of the script hook is similar for all of the individual lifecycle operations, one description is presented. The term “LifeCycleOperation” is used to represent any one of the operations listed above.

## ***Script Execution Timing***

Stage	Timing
Click	Executed while browsing in the LifeCycleOperation view before the current object task attributes values are displayed, enabling you to set attributes for the selected object.

## **Hook Timing in the SmarTeam LifeCycleOperation**

This section describes the hook timing relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the LifeCycleOperation script hooks.

User invokes a LifeCycleOperation on an object on the SmarTeam view

The Before LoadLifeCycleScreen and After LoadLifeCycleScreen hooks occur, if set.

The **Click LifeCycleOperation hook** occurs

The LifeCycleOperation screen is displayed.

User clicks OK on the LifeCycleOperation screen

The Before LifeCycleOperation hook occurs

The LifeCycleOperation occurs or the InsteadOf LifeCycleOperation hook occurs.

The After LifeCycleOperation hook occurs

The SmarTeam view shows the object in its new lifecycle state.

**Aborting:**

You cannot abort the Life-Cycle Operation using these hooks.

**Script Hook Parameters****Click Hook**

The following table describes the arguments passed in the Click LifeCycleOperation script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Check In -- 'CHECKIN' Check Out -- 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing selected object attributes.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Empty record list for creating and outputting task attributes. See Figure 12.
ThirdPar	Not used.
Return value	See Table 1.  This operation cannot be aborted by assigning Err_Gen, Err_Refuse to the script return value.

## Examples

### Before Load Life-Cycle Screen

This script is designed for the Before LoadLifeCycle Screen script hook for a class. It sets the default values for task attributes in ThirdPar for objects of the class undergoing this lifecycle operation.

```
Function BeforeLoadLCScreenExample(ApplHndl As Long, Sstr As String,
FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim SecondRec As Object
    Dim ThirdRec As Object
    Dim CheckOutId, CheckInId, OperId As Integer
    ' Convert pointer to COM object SmSession
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert input parameter to COM object
    CONV_RecListToComRecordList FirstPar,FirstRec
    CONV_RecListToComRecordList SecondPar,SecondRec
    CONV_RecListToComRecordList ThirdPar,ThirdRec
    'Set common defaults
    ThirdRec.ValueAsString(NM_DSC_NOTES, 0) = "DSC_NOTES"
    ThirdRec.ValueAsString(NM_DIRECTORY, 0) = "c:\\"
    ThirdRec.ValueAsString(NM_EFFECTIVE_UNTIL, 0) = "2/27/2001"
    ThirdRec.ValueAsString(NM_REVISION, 0) = "b"
    ThirdRec.ValueAsString(NM_PHASE, 0) = 6 'set production phase
    If (Sstr = "CHECKIN") Then
        'Set CheckIn defaults
        ThirdRec.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_TSK_KEEP_LOCAL_COPY, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_TSK_KEEP_CHECKEDOUT, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_LOGICAL_LINK_COPY, 0) = 1
        ThirdRec.AddHeader NM_REPLACE_TO_LATEST_AVLBL, 2, 2
        ThirdRec.ValueAsSmallInt(NM_REPLACE_TO_LATEST_AVLBL, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_CPY_LOG_LNKS_LIST, 0) = 0
        ThirdRec.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_WorkRev
    ElseIf (Sstr = "CHECKOUT") Then
        'Set CheckOut defaults
        ThirdRec.ValueAsSmallInt(NM_TSK_NOCREATE_LOCAL_COPY, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_LFCYC_NEW_BRANCH, 0) = 1
        ThirdRec.ValueAsSmallInt(NM_LOGICAL_LINK_COPY, 0) = 1
```



```
ThirdRec.ValueAsSmallInt(NM_LINKS_TO_SONS_COPY, 0) = 1
ThirdRec.ValueAsSmallInt(NM_FILE_OVERWRITE, 0) = fowYes
End If
CONV_ComRecListToRecordList      ThirdRec, ThirdPar
BeforeLoadLCScreenExample = Err_None
End Function
```

## After Load Life-Cycle Screen

This script is designed for the After Load Life-Cycle Screen script hook for a class. It sets the phase value of “Design” to the objects of the class undergoing this lifecycle operation.

```
Declare Sub CONV_RecListToCOMRecordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRecList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmRecList.SmRecordList, ByRef RecList As Long)

Function LFCycBrowseOper(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim PhaseClassId As Integer
    Dim Phase As ISmLookUpObject
    Dim SecondRec As Object
    Dim SmSession As SmApplic.SmSession

    On Error GoTo ErrorTreat
    CONV_RecListToCOMRecordList SecondPar,SecondRec
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Get phase lookup table class id
    PhaseClassId = SmSession.MetaInfo.SmClassByName("Phase").ClassId
    ' Get lookup object by unique name
    Set Phase =
SmSession.ObjectStore.GetSmLookUpByUniqueName(PhaseClassId,"Design")
    ' Add attribute to result record list
    SecondRec.AddHeader NM_PHASE,SIZE_OBJ_ID,TDMT_OBJ_ID
    ' Set phase stage on lifecycle screen as Design
    SecondRec.ValueAsInteger(NM_PHASE,0) = Phase.Id

    CONV_COMRecListToREcordList SecondRec, SecondPar
ErrorTreat:
    LFCycBrowseOper = Err.Number
End Function
```

---

## Scripts for Individual Non-GUI Lifecycle Operations

This section describes script hooks for the individual non GUI-based lifecycle operations:

Check Out

Undo Check Out

Check In

Release

New Release

Obsolete

Copy File

Where each operation has a script hook for each of the stages:

Before

After

InsteadOf

Since the behavior of the script hooks is similar for all of the individual lifecycle operations, one description is presented. The term “LifeCycleOperation” is used to represent any one of the operations listed above.

### ***Script Execution Timing***

Stage	Timing
<b>Before</b>	Executed before the system performs the LifeCycleOperation on the object.
<b>After</b>	Executed after the system has performed the LifeCycleOperation the object.
<b>InsteadOf</b>	Executed instead of the system performing the LifeCycleOperation on the object.

### **Hook Timing in the SmarTeam LifeCycleOperation**

This section describes the hook timing relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the LifeCycleOperation script hooks.

User invokes a LifeCycleOperation on an object on the SmarTeam view

The Before LoadLifeCycleScreen and After LoadLifeCycleScreen hooks occur, if set.

The Click LifeCycleOperation hook occurs

The LifeCycleOperation screen is displayed.

User clicks OK on the LifeCycleOperation screen

The **Before LifeCycleOperation** hook occurs

The LifeCycleOperation occurs or the **InsteadOf LifeCycleOperation** hook occurs.

The **After LifeCycleOperation** hook occurs

The SmarTeam view shows the object in its new lifecycle state.

See Table 8 for the timing of these hooks relative to the others.

**Aborting:**

You can abort the lifecycle operation by returning Err\_Gen in the Before and InsteadOf hooks.

## Script Hook Parameters

### Before Hook

The following table describes the arguments passed in the Before LifeCycleOperation script hook.

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing selected object attributes. The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Empty record list for creating and outputting task attributes. See Figure 12.
ThirdPar	Output object attributes.
Return value	See Table 1. This operation is aborted by assigning Err_Gen, Err_Refuse to the script return value.

### After Hook

The following table describes the arguments passed in the After LifeCycleOperation script hook.

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing selected individual object attributes and task attributes that were used in the operation.  The attributes passed depend on the kind of object selected and the task being executed. See the section Object Attributes Input to a Script and Representing SmarTeam Tasks.
SecondPar	Input record list containing individual and group task attributes values that were used in the operation.
ThirdPar	Output object attributes.
Return value	See Table 1.  This operation cannot be aborted by assigning Err_Gen, Err_Refuse to the script return value.

## InsteadOf Hook

The following table describes the arguments passed in the InsteadOf LifeCycleOperation script hook.

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing selected individual object attributes and task attributes that were used in the operation.  The attributes passed depend on the kind of object selected and the task being executed. See the section Object Attributes Input to a Script and Representing SmarTeam Tasks.
SecondPar	Input record list containing individual and group task attributes values that were used in the operation.
ThirdPar	Output object attributes.
Return value	See Table 1.  This operation is aborted by assigning Err_Gen, Err_Refuse to the script return value.

**Note:** The Execution script **InsteadOf** is designed to work on unlocked objects only. Vault objects are locked by the SmarTeam locking mechanism. When performing out of vault lifecycle operations the objects are unlocked. If you implement your own lifecycle operations, you must unlock the objects in the Before hook.

Then, the next time you promote, this fix will be promoted.

## **Examples**

### **After Release**

This script is attached to the After Release hook and maintains a log of released items.

```
Declare Sub CONV_RecListToComRecordList Lib "SmtDm32" (ByVal RecList As Long, ByRef COMRecList As ISmRecordList)
Declare Sub CONV_ComRecListToRecordList Lib "SmtDm32" (ByVal COMRecList As ISmRecordList, ByRef REcList As Long)

Function AfterApprove(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim ReleasedObject As SmApplic.ISmObject
    Dim FirstRec As Object

    If StrComp(Sstr, "APPROVE") <> 0 Then
        Exit Function
    End If

    CONV_RecListToComRecordList FirstPar,FirstRec
    Set SmSession=SCREXT_ObjectForInterface(ApplHndl)
    Set ReleasedObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0), True)
    Open SmSession.Config.HomeDirectory + "\Released.log" For Append As #1
    Print #1, "CLASSID = " & CStr(ReleasedObject.ClassId)
    Print #1, "OBJECTID = " & CStr(ReleasedObject.ObjectId)
    Close #1

    AfterApprove = Err_None
End Function
```

### **Before LifeCycleOperation**

For additional examples, see the examples for After Load Life-Cycle Screen. Attributes for individual lifecycle operations that can be set for After Load Life-Cycle Screen can also be set for each individual Before LifeCycleOperation hook, for example, Before CheckIn.

---

## Scripts for Group Lifecycle Operations

As mentioned above, in general SmarTeam – Editor performs lifecycle operations on a group of objects. Prior to performing the operation on each object, the system builds trees connected to objects, searches for links and CFO and checks consistency. The group lifecycle hooks permit you to intervene at various stages of this process.

The group lifecycle hooks have the following properties:

They operate at a stage at which no further user input is possible. Any changes caused by the script at this point are final.

They operate for all of the lifecycle tasks.

More extensive object and task information is provided than in the previous hooks.

The group lifecycle hooks are ideal for making general changes and modifications such as introducing a different file naming convention or revision numbering system than the one provided by SmarTeam – Editor.

**Note:** For Life-Cycle Stages 1 and 2, the scripts are hooked from the Parent (Common) Class.

The following hooks are provided for group lifecycle operations:

Operation in Script Maintenance	Before	After	InsteadOf
<b>Life-Cycle Stage 1</b>	<b>X</b>	<b>X</b>	<b>X</b>
Life-Cycle Stage 2	X	X	X

### LifeCycle Stage 1 Hook

The Before LifeCycle1 hook is preferred over LifeCycle Stage 2 for making general changes and modifications. The reason is that it occurs before the system assembles and checks all the input to the lifecycle operation. Therefore any changes you do are checked.

### LifeCycle Stage 2 Hook

The Before LifeCycle Stage 2 hook occurs after the system checks and just before the lifecycle task is executed. This is an ideal place to carry out additional checks of your own on the input. There is generally more information available at this hook because of the retrieval done during checking so that it is easier to carry out certain operations here. This is also the place to be careful; the system does not check anything you do here. You must do the checking.



See Table 8 for the timing of this hook with respect to the other lifecycle hooks.

## **Group Lifecycle Task Attributes**

This section describes the various task attributes, which you can use in the lifecycle hooks to influence the group lifecycle process. See the section *Passing Lifecycle Task Information to Script Functions* for information on how to pass these attributes to a script. For information about task attributes for individual lifecycle operations, see the section *Individual Lifecycle Task Attributes*.

For each task attribute, the lifecycle operations for which it is relevant is shown.

In addition, the script hooks for which the task attribute is relevant are shown in the *Applicable Hooks* entry.

Note: if you set a task attribute in a script hook that is outside of the range denoted, SmarTeam – Editor will ignore the value.

### ***NM\_REPLACE\_TO\_LATEST\_AVLBL***

<b>Attribute Name</b>	“REPLACE_TO_LATEST_AVLBL”
<b>Data Type</b>	sdtSmallInt
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

#### **Description**

Specifies whether a group lifecycle operation should replace objects with their latest available revision. For example, if you perform a lifecycle operation on an Assembly with several levels of children, including those with links, SmarTeam – Editor will replace the object itself and the children objects with their latest revision.

This property does not override the setting:

Tools/ Administrator Options/Life-Cycle Options/Out Of Vault/ Latest revision/ Always check out latest available revision

**Values**

Value	Description
0	Don't replace with latest available revision
1	Replace with latest available revision

***NM\_TDM\_GET\_LATEST\_AVLBL\_CHILD***

<b>Attribute Name</b>	"TDM_GET_LATEST_AVLBL_CHILD"
<b>Data Type</b>	sdtSmallInt
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

**Description**

Specifies whether a group lifecycle operation should replace an object's children with their latest available revision.

For example, if you perform a lifecycle operation on an Assembly with several levels of children, including those with links, SmarTeam – Editor will replace the child objects with their latest revision. The parents are not replaced with their latest revision.

This property does not override the setting:

Tools/ Administrator Options/Life-Cycle Options/Out Of Vault/ Latest revision/ Always check out latest available revision

**Values**

Value	Description
0	Don't replace child objects with latest available revision
1	Replace child objects with latest available revision

***NM\_ ASK\_FILE\_NAME\_NOT\_UNIQUE***

**Attribute Name** "ASK\_FILE\_NAME\_NOT\_UNIQUE"

**Data Type** sdtSmallInt

**Life-Cycle Operations** Check In

**Applicable Hooks** Before LifeCycleStage1

**Description**

During a Check In operation, if the name of the file that is being checked in is not unique in the vault, SmarTeam – Editor will try to replace the file name with one that is unique.

NM\_ ASK\_FILE\_NAME\_NOT\_UNIQUE specifies whether to ask the user to verify the name change.

You get a message similar to the following:

```
File 'new-tech.doc' (for Object "DOC-0075 b.1") is not  
unique in directory "c:\".
```

```
Are you sure you want to change the file name to "new-  
tech_3948.doc"?
```

**Values**

Value	Description
0	Don't ask if the file name is not unique
1	Ask if the file name is not unique

**NM\_ MULTIPLE\_REVISION\_TREAT**

<b>Attribute Name</b>	“MULTIPLE_REVISION_TREAT”
<b>Data Type</b>	sdtSmallInt
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, NewRelease
<b>Applicable Hooks</b>	Before LifeCycleStage1

**Description**

NM\_ MULTIPLE\_REVISION\_TREAT lets you specify the behavior of SmarTeam – Editor if multiple revisions of an object are found during a group lifecycle operation. The possibilities are listed in the Value section below.

**Values**

Value	Description
Attribute column not found	A dialog box is displayed to query the user which version to use.
0 or NULL_Int16	The lifecycle operation is aborted with error code Err_Refuse when multiple revisions are found.
1	The lifecycle operation proceeds with the selected version without asking

**NM\_ NO\_ASK\_CHILD\_OPER\_INCONSISTENT**

<b>Attribute Name</b>	“NO_ASK_CHILD_OPER_INCONSISTENT”
<b>Data Type</b>	sdtBoolean
<b>Life-Cycle Operations</b>	Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

**Description**

NM\_ NO\_ASK\_CHILD\_OPER\_INCONSISTENT lets you specify whether to ask the user when operation codes are inconsistent in the tree during a group lifecycle operation. The possibilities are listed in the Value section below.

One type of operation code inconsistency occurs when an object has two parents, where one parent needs to be checked out and the other parent requires the copy file operation.

A second type of inconsistency can occur with a CFO object. Some CFO objects can have Check Out/New Release operation ID, and the parents of some of them have CheckOut/New Release operation ID, while others have CopyFile or NoOp operation ID.

Note that you cannot set this field from the screen -- only from a script.

### Values

Value	Description
Attribute column not found	If an inconsistency is found a dialog box is displayed to ask whether to continue.
0 or NULL_Int16	The lifecycle operation is aborted with error code Err_Refuse when an inconsistency is found
1	The operation proceeds without asking when an inconsistency is found

### ***Tree Filter Parameters***

The following parameters:

Parameter	Tree Filter Type
NM_RevisionFilter	Revision Filter
NM_FromDate	Effectivity Date Filter
NM_UntilDate	Effectivity Date Filter
NM_AllowOverLap	Effectivity Date Filter

give you control over the respective tree filters when a tree is constructed during a group lifecycle operation.

**Note:** You should set parameters for one filter type only. Otherwise, the results are unpredictable.

The action of the Revision Filter is explained in the section Tree Filter Parameters.

The results of the Effectivity Date Filter are as follows.

Defining:

“Filter Interval” as the time interval [FromDate—UntilDate]

“Object Interval” as the time interval [EffectiveFrom—EffectiveUntil],

the following table shows the results of the filter depending on the relation of the two intervals.

Relation between Filter Interval and Object Interval	Is Object included?
Filter Interval included in Object Interval	Yes
Filter Interval not included and doesn't overlap Object Interval	No
Filter Interval overlaps but not included in Object Interval	Yes if NM_AllowOverLap=1; No if NM_AllowOverLap=0

## ***NM\_RevisionFilter***

<b>Attribute Name</b>	“RevisionFilter”
<b>Data Type</b>	sdtSmallInt
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

### **Description**

NM\_ REVISIONFILTER lets you specify the action of the Revision Filter. The Revision Filter is used when SmarTeam – Editor builds a tree of objects for a lifecycle operation and there are objects in a branch, which are different revisions of the same object. The possibilities are listed in the Value section below.

Appears on SmarTeam – Editor in: Tree Properties/Tree Filter

### **Values**

<b>Value</b>	<b>Description</b>
NotRevisionFilter	No revision filter is applied.
LastRevisionFilter	Only those objects are selected whose revision is the last public revision among all existing versions in the system. The last public revision is indicated by the database attribute REVISION_STG having a value of 1. If there are no such children, no objects are added to the list. This option corresponds to the “Last public revision” button on the SmarTeam – Editor Tree Properties/Tree Filter screen.
LatestRevisionFilter	Only those objects are selected that have the latest chronological revision among the set of objects input to the lifecycle operation. Those having older revisions are not added. This option includes objects that are not available (i.e., checked out). This option corresponds to the “Latest revision” button on the SmarTeam – Editor Tree

---

	Properties/Tree Filter screen.
LatestAvailableRevisionFilter	Same as LatestRevisionFilter except that revisions not available (checked out) are not included.

---

## ***NM\_ FromDate***

<b>Attribute Name</b>	“DateFrom”
<b>Data Type</b>	sdtEffectiveDateFrom
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

### **Description**

NM\_ FromDate lets you specify the action of the Effectivity Date Filter. The Effectivity Date Filter is used when SmarTeam – Editor builds a tree of objects for a lifecycle operation and there are objects in a branch. FromDate lets you include only those objects whose effectivity date, NM\_EFFECTIVE\_FROM, is before the value given for NM\_ FromDate (for more information on the action of this filter see Tree Filter Parameters).

Appears on SmarTeam – Editor in: Tree Properties/Tree Filter/Effectivity Date Filters/From

### **Values**

From Date.



## ***NM\_ UntilDate***

<b>Attribute Name</b>	“DateUntil”
<b>Data Type</b>	sdtEffectiveDateUntil
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

### **Description**

NM\_ UntilDate lets you specify the action of the Effectivity Date Filter. The Effectivity Date Filter is used when SmarTeam – Editor builds a tree of objects for a lifecycle operation and there are objects in a branch. UntilDate lets you include only those objects whose effectivity date, NM\_EFFECTIVE\_UNTIL, is later than the value given for NM\_ UntilDate (for more information on the action of this filter see Tree Filter Parameters.)

Appears on SmarTeam – Editor in: Tree Properties/Tree Filter/Effectivity Date Filters/To

### **Values**

Until Date

## ***NM\_ AllowOverLap***

<b>Attribute Name</b>	“AllowOverLapping”
<b>Data Type</b>	sdtSmallInt
<b>Life-Cycle Operations</b>	Check In, Release, Check Out, New Release
<b>Applicable Hooks</b>	Before LifeCycleStage1

### **Description**

NM\_ AllowOverLap lets you specify the action of the Effectivity Date Filter. The Effectivity Date Filter is used when SmarTeam – Editor builds a tree of objects for a lifecycle operation and there are objects in a branch. AllowOverLap lets you specify the action of the filter when the Filter Interval overlaps but is not included in Object Interval (for more information on the action of this filter see Tree Filter Parameters).

Appears on SmarTeam – Editor in: Tree Properties/Tree Filter/Effectivity Date Filters/Allow overlapping

### Values

Value	Description
0	If Filter Interval overlaps but is not included in Object Interval, don't include object
1	If Filter Interval overlaps but is not included in Object Interval, include object

## ***Passing Life-Cycle Association Information to Script Functions***

In the group lifecycle scripts, association information is passed to and from a script function by means of the Record List parameter SecondPar. This association information is related to the object information passed in FirstPar. This section discusses these parameters in detail for scripts attached to advanced lifecycle operations. In this section, it is assumed that the Record List parameters have been converted to COM format (see Converting Procedural Parameters on page 9).

### ***Representing SmarTeam Associations***

The attributes of a SmarTeam association are represented by an SmRecord object, which represents one record or row of an SmRecordList object.

An SmRecordList object can represent association attributes for a set of associations. The headers of the columns of the SmRecordList matrix represent the association attributes and the rows of the SmRecordList matrix represent the association objects. Each cell of the row contains the value of the association attribute denoted by the cell column's header.

Figure 17 shows a SmRecordList object that represents n association attributes of m associations. The attributes shown identify the associated objects Object1 and Object2 by their Class\_ID and Object\_ID. Note that in both these figures the Class\_ID and the Object\_ID of the Associations themselves are not shown, although they are also passed to the script.

Table 10 describes the association attributes.

The user should not add associations.

*Figure 17 Association Attributes Represented by an SmRecordList*

	Association Attributes				
<b>Header:</b> Name Type Size	Class_Id1 2 2	Object_Id1 10 4	Class_Id2 2 2	Object_Id2 10 4	...
<b>Association-1</b>	Class_Id1-1	ObjectId1-1	Class_Id2-1	ObjectId2-1	...
<b>Association-2</b>	Class_Id1-2	ObjectId1-2	Class_Id2-2	ObjectId2-2	...
<b>Association-3</b>	Class_Id1-3	ObjectId1-3	Class_Id2-3	ObjectId2-3	...
...	...	...	...	...	...
<b>Association-m</b>	Class_Id1-m	ObjectId1-m	Class_Id2-m	ObjectId2-m	...

### Association Attributes

Table 10 describes the attributes for associations, which occur in group lifecycle operations

*Table 10 Association Attributes*

Attribute Name	Type (preface with sdt)	Description
CLASS_ID	SmallInt	CLASS_ID of association.  If the record represents a common file object association (no actual association between the objects) then CLASS_ID = NULL_INT16
OBJECT_ID	ObjectIdentifier	OBJECT_ID of association.  If the record represents a common file object association (no actual association between the objects) then OBJECT_ID = NULL_OBJ_ID
CLASS_ID1	SmallInt	CLASS_ID of first object. Can be NULL_INT16
OBJECT_ID1	ObjectIdentifier	OBJECT_ID of first object
CLASS_ID2	SmallInt	CLASS_ID of second object. Can be NULL_INT16
OBJECT_ID2	ObjectIdentifier	OBJECT_ID of second object

### **Relation between Associations and Objects**

In general, the associations passed in SecondPar correspond to the objects passed in FirstPar.

Figure 18 shows the Record Lists for FirstPar and SecondPar in such a way that the correspondence is emphasized. In this example, objects 1,2 and objects 3,4 in FirstPar have a parent-son relation. Accordingly, two association objects are passed in SecondPar.

*Figure 18 Relation between Association and Object Attributes*

Object Attributes		Association Attributes				
Object_ID	..	Class_Id1	Object_Id1	Class_Id2	Object_Id2	...
ObjectID-1	..	Class_Id1	Object_Id1	Class_Id2	Object_Id2	...
ObjectID-2	..	Class_Id3	Object_Id3	Class_Id4	Object_Id4	...
ObjectID-3	..					...
ObjectID-4	..					...

FirstPar

SecondPar

Note: Association information is passed for the advanced lifecycle script hooks only when the advanced LC screen is used. For the Light LC screen, no Association information is passed.

## ***Lifecycle Stage 1, 2 Hooks***

### ***Script Execution Timing***

Stage	Timing
<b>Before</b> LifeCycle1	Executed before the system builds trees connected to objects, searches for links and CFO and checks consistency.
<b>Before</b> LifeCycle2	Before system performs lifecycle operation on objects
<b>InsteadOf</b> LifeCycle2	Script performed instead of system group lifecycle operation
<b>InsteadOf</b> LifeCycle1	Script performed instead of system group lifecycle operation
<b>After</b> LifeCycle2	After the system has performed the lifecycle operation on all objects.
<b>After</b> LifeCycle1	After the system has performed the lifecycle operation on all objects.

See Table 8 for the timing of these hooks relative to the others.

**Aborting:**

You can abort the lifecycle operation by returning Err\_Gen in the Before and InsteadOf hooks.

As mentioned before, you should use them only when you are aware of all possible effects of the script.

**Script Hook Parameters**

The following script hooks parameters are applicable to both the Lifecycle Stage 1 and Lifecycle Stage 2 hooks.

**Note:** For Lifecycle Stages 1 and 2, the scripts are hooked from the Parent (Common) Class

**Before Hook**

The following table describes the arguments passed in the Before Lifecycle Stage script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Name of current lifecycle operation: Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing object attributes for all objects that appear on the lifecycle screen.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Object association input.  For the attributes that can be passed, see the section Passing Life-Cycle Association Information to Script

---

	Functions.
ThirdPar	Holds:  Tasks defaults input, See the section Passing Default Task Attributes on page 85 and Figure 16  Task default output (for subsequent objects)  Object task output, see Passing Lifecycle Task Information to Script Functions)
Return value	See Table 1.  This operation is aborted by assigning Err_Gen, Err_Refuse to the script return value.

---

### After Hook

The following table describes the arguments passed in the After Lifecycle Stage1 script hook.

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Name of current lifecycle operation: Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing object attributes for all objects that appear on the lifecycle screen.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Object association input. See Before hook.
ThirdPar	Holds default tasks. See Before hook.
Return value	See Table 1. This operation cannot be aborted by assigning Err_Gen, Err_Refuse to the script return

---



---

value.

---

### **InsteadOf Hook**

The following table describes the arguments passed in the InsteadOf Life-Cycle Stage1 script hook.

### **Arguments**

<b>Argument</b>	<b>Description</b>
ApplHndl	Input. See Table 1.
SelectOp	Name of current lifecycle operation: Check In -- 'CHECKIN' Check Out 'CHECKOUT' Undo Check Out -- 'UNDOCHECKOUT' Release -- 'APPROVE' New Release -- 'NEWRELEASE' Obsolete -- 'FREEZE' Copy File -- 'COPY FILE'
FirstPar	Input record list containing object attributes for all objects that appear on the lifecycle screen.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Object association input. See Before hook.
ThirdPar	Holds default tasks. See Before hook.
Return value	See Table 1.  This operation is aborted by assigning Err_Gen, Err_Refuse to the script return value.

---

## Examples

### Before Life-Cycle Stage 1

This script sets default values for the group lifecycle operation.

```
Function BLCStage1Example(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer

    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim SecondRec As Object
    Dim ThirdRec As Object

    ' Convert pointer to COM object SmSession
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert input parameter to COM object
    CONV_RecListToComRecordList FirstPar,FirstRec
    CONV_RecListToComRecordList SecondPar,SecondRec
    CONV_RecListToComRecordList ThirdPar,ThirdRec

    ThirdRec.AddHeader "RevisionFilter", 2, 2
    ThirdRec.ValueAsSmallInt("RevisionFilter", 0) = LastRevisionFilter
    'ThirdRec.ValueAsString("DateFrom", 0) = "1/1/2001"
    'ThirdRec.ValueAsString("DateUntil", 0) = "2/1/2001"
    'ThirdRec.ValueAsSmallInt("AllowOverLapping", 0) = 1
    ThirdRec.AddHeader "MULTIPLE_REVISION_TREAT", 2, 2
    ThirdRec.ValueAsSmallInt("MULTIPLE_REVISION_TREAT", 0) = 1
    ThirdRec.AddHeader NM_REPLACE_TO_LATEST_AVLBL, 2, 2
    ThirdRec.ValueAsSmallInt(NM_REPLACE_TO_LATEST_AVLBL, 0) = 1
    If (Sstr = "CHECKIN") Then
        'Set CheckIn defaults
        ThirdRec.AddHeader "ASK_FILE_NAME_NOT_UNIQUE", 2, 2
        ThirdRec.ValueAsSmallInt("ASK_FILE_NAME_NOT_UNIQUE", 0) = 0
    ElseIf (Sstr = "CHECKOUT") Then
        'Set CheckOut defaults
        ThirdRec.AddHeader "NO_ASK_CHILD_OPER_INCONSISTENT", 2, 2
        ThirdRec.ValueAsSmallInt("NO_ASK_CHILD_OPER_INCONSISTENT", 0) = 1
    End If
    CONV_ComRecListToRecordList ThirdRec, ThirdPar
    BLCStage1Example = Err_None
End Function
```

## After Life-Cycle Stage 1,2

'This script performs auto check out operation after the group lifecycle operation is complete.

```
Function AfterLifeCycle(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim SmSession      As SmApplic.SmSession
    Dim ObjStore        As SmApplic.ISmObjectStore
    Dim WorkObject      As ISmObject
    Dim SClassId        As String
    Dim SObjectId       As String
    Dim CNT             As Long
    Dim RetCode         As Integer

    On Error GoTo ErrorTreat
    Set SmSession=SCREXT_ObjectForInterface(ApplHndl)
    RetCode = 100
    ' If auto check out operation enabled - global variable set to false
    If SmSession.GlobalData.Value("LifeCycle") = False Then
        RetCode = 0
        ' Disable auto check out operation - loop not needed
        SmSession.GlobalData.Value("LifeCycle") = True
        RetCode = 1
        Set ObjStore = SmSession.ObjectStore
        Set WorkObject = Nothing
        ' Convert old record list into com object
        CONV_RecListToCOMRecordList    FirstPar,FirstRec
        ' Get first object from record list
        Set WorkObject =
ObjStore.ObjectFromData(FirstRec.GetRecord(0),false)
        ' Try to perform auto check out operation
        CheckOut SmSession,WorkObject
        ' Enable auto check out operation for next time
        SmSession.GlobalData.Value("LifeCycle") = False
    End If
    AfterLifeCycle = Err_None
    Exit Function

Declare Sub CONV_RecListToCOMRecordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRECList.SmRecordList)

Sub CheckOut(SmSession As SmApplic.SmSession,WorkObject As SmApplic.ISmObject)
```

```
Dim Operation As SmApplic.ISmOperation      'performed operation object
Dim NewWorkObject As SmApplic.ISmObject 'new object
Dim Metainfo As SmApplic.SmMetaInfo 'metainfo object for smClasses
Dim SessionUtil As SmUtil.SmSessionUtil 'main SmarTeam service object
'Dim TaskRecord As SmRecList.SmRecord 'tasks record for operation -
for VB
    Dim TaskRecord As Object 'tasks record for operation - for SmartScript
    Dim OperName As String 'operation name
    Dim InvokeScripts As Boolean 'invoke scripts on operation
    Dim Result As Long 'result of operation - new object id for refresh
    Dim WorkDir As String 'work directory

    ' Get service object
    Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")
    ' Get work directory
    WorkDir = SmSession.Config.HomeDirectory & "\\Work"
    ' Invoke scripts on operation execute (before, after, instead)
    InvokeScripts = True
    ' Set operation name according to default constant
    OperName = NM_OPER_CHECKOUT
    ' Create task record
    Set TaskRecord=CreateObject("SmRecList.SmRecord")
    ' Add task to tasks record
    TaskRecord.AddHeader NM_FILE_NAME, 255, sdtChar
    ' Add task to task record
    TaskRecord.AddHeader NM_DIRECTORY, 255, sdtChar
    ' Set destination file name for task record
    TaskRecord.ValueAsString(NM_FILE_NAME) =
WorkObject.Data.ValueAsString(NM_CAD_REF_FILE_NAME)
    ' Set destination work directory
    TaskRecord.ValueAsString(NM_DIRECTORY) = WorkDir
    Set Metainfo = SmSession.Metainfo
    If Not (WorkObject Is Nothing) Then
        ' Get operation object depend on specific SmClass and operation
name - only for test
        Set Operation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)
        ' Check if operation allowed for object
        If SessionUtil.OperationAllowedOnObject(WorkObject, Operation,
False) Then
            ' Perform operation on object using SmSessionUtil method
```

```
        Set NewWorkObject = SessionUtil.CheckOut(WorkObject, TaskRecord,  
InvokeScripts)  
    End If  
End If  
End Sub
```

---

## Scripts for File Operations

Operations of this family can be executed on objects from the File-Managed class.

The setup of file operations is defined by the system administrator in the Application Tool (in SmarTeam – Editor, use the menu Tools/Applications Setup). For example, the operation ‘Edit’ can be defined to open a doc file in an MS Word application in the SmDemo database.

This section describes script hooks for the file operations:

**Edit** – called in SmarTeam – Editor when you invoke the Edit operation

**View** – called in SmarTeam – Editor when you invoke the View operation

**RedLine** – called in SmarTeam – Editor when you invoke the Redline operation

**Print** – called in SmarTeam – Editor when you invoke the Print operation

**OnViewer** – called in SmarTeam – Editor when you choose the “Viewer” tab on the Profile Card and activate the viewer control.

Where each operation has a script hook for each of the stages:

- Before
- After
- InsteadOf

except for OnViewer, which only has a Before hook.

For information about the script hooks of the Copy File operation, see the section Scripts for Individual Non-GUI Lifecycle Operations. Note that SmarTeam – Editor uses Copy file in some of the file operations described below.

Since the behavior of the script hooks is similar for all of the file operations, one description is presented. The term “FileOperation” is used to represent any one of the operations listed above.

## ***Script Execution Timing***

Stage	Timing
Before	Executed before the system invokes FileOperation.
After	Executed immediately after the system has invoked FileOperation.
InsteadOf	Executed instead of FileOperation defined by the system.

### **Hook Timing in the SmarTeam Edit Operation**

This section describes the hook timing of the Edit file operation relative to the SmarTeam GUI operations. Note that this description does not necessarily apply to a user-written application using the Edit script hooks.

User invokes a Edit on an object on the SmarTeam view

The **Before Edit** hook occurs.

If set, the **InsteadOf Edit** hook occurs, else the system-defined Editor is launched

The **After Edit** hook occurs.

#### **Aborting:**

You cannot abort the file operation using these hooks.

### **Hook Timing in the Remaining SmarTeam File Operations**

This section describes the hook timing of the remaining file operations (other than Edit) relative to the SmarTeam GUI operations. These operations use the Copy File operation so that any hooks you have set for Copy File also occur. Note that this description does not necessarily apply to a user-written application using the FileOperation script hooks.

User invokes a file operation on a file on the SmarTeam view

The Before LifeCycleStage1, Before LifeCycleStage2, BeforeCopyFile, After LifeCycleStage1 and After LifeCycleStage2 hooks occur, if set.

The **Before FileOperation** hook occurs.

The **InsteadOf FileOperation** hook occurs, if set, else the file application is launched on the file.

The **After FileOperation** hook occurs.

If the **InsteadOf FileOperation** hook is not set, the **After FileOperation** hook occurs immediately after the file application is launched on the file. The file application remains open until the user closes it. Meanwhile SmarTeam operations can continue.

**Aborting:**

You cannot abort the file operation using these hooks.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before FileOperation script hook.

#### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Edit -- 'Edit' View -- 'View' Red Line -- 'RedLine' Print -- 'Print' On Viewer -- 'OnViewer'
FirstPar	Input record list containing object attributes for all objects.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Input containing attributes of the application tool that is invoked by SmarTeam – Editor. These attributes appear on the SmarTeam – Editor screen Tools/Applications Setup/ Tools/Add or Modify. (They are stored as an internal SmarTeam object and cannot be changed).
ThirdPar	Output new object attributes.  For example, the file can be changed by outputting the modified file name and directory (for example: if an object contains several attached files, you can decide which file to edit at run time).



### **After Hook**

The following table describes the arguments passed in the After FileOperation script hook.

#### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Edit -- 'Edit' View -- 'View' Red Line -- 'RedLine' Print -- 'Print'
FirstPar	Input record list containing object attributes for all objects.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Input containing attributes of the application tool that is invoked by SmarTeam – Editor. These attributes appear on the SmarTeam – Editor screen Tools/Applications Setup/ Tools/Add or Modify. (They are stored as an internal SmarTeam object and cannot be changed).
ThirdPar	Not used.

### **InsteadOf Hook**

The following table describes the arguments passed in the InsteadOf FileOperation script hook.

## Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Edit -- 'Edit' View -- 'View' Red Line -- 'RedLine' Print -- 'Print'
FirstPar	Input record list containing object attributes for all objects.  The attributes passed depend on the kind of object selected. See the section Object Attributes Input to a Script.
SecondPar	Input containing attributes of the application tool that is invoked by SmarTeam – Editor. These attributes appear on the SmarTeam – Editor screen Tools/Applications Setup/ Tools/Add or Modify. (They are stored as an internal SmarTeam object and cannot be changed).
ThirdPar	Not used.

## Examples

### Before On Viewer

This script is attached to the Before On Viewer script hook. In case the file to be viewed is a text file, the script causes the file to be opened and viewed in Word.

```
Declare Sub CONV_RecListToCOMRecordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmRecList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmRecList.SmRecordList, ByRef RecList As Long)
Function BeforeOnViewer(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim FirstRec As Object
    Dim WorkObject As SmApplic.ISmObject
    Dim SmSession As SmApplic.SmSession
    Dim TextFileType As Integer
    Dim LookUpClassId As Integer
    Dim Id
```

```
Dim CommandLine As String
Dim Cnt As Long
On Error GoTo HandleError

Set SmSession=SCREXT_ObjectForInterface(ApplHndl)
CONV_RecListToCOMRecordList FirstPar,FirstRec
Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)
LookUpClassId = SmSession.MetaInfo.SmClassByName("FILE_TYPE").ClassId
TextFileType =
SmSession.ObjectStore.GetSmLookUpByUniqueName(LookUpClassId,"Text")
If WorkObject.RetrieveFileType = TextFileType Then
    WorkObject.CopyFileFromVault "C:\Temp",
WorkObject.Data.ValueAsString("FILE_NAME")
    Id = Shell("word C:\Temp\"+
WorkObject.Data.ValueAsString("FILE_NAME"),vbHide)
    AppActivate Id
End If
Exit Function
HandleError:
MsgBox Err.Description
End Function
```

---

## Scripts for Authorization Operations

These hooks enable you to perform additional authorization analysis on objects in the system. For example, you can use the OnBrowse script hook to prevent opening the documents view when a user, not authorized to access a specific project, tries to browse the project.

### ***OnLogin***

This script hook is used to establish authorization procedures on login.

You can cause the login dialog to be skipped by passing the user name and password in the Before OnLogin script hook.

#### ***Script Execution Timing***

Stage	Timing
Before	Executed before the login dialog is opened.
After	Executed after the user clicks 'Ok' in the login dialog or after a correct password was entered in the Before script.
InsteadOf	Not applicable.

#### ***OnLogin Hook Attributes***

##### ***NM\_LOGIN***

**Attribute Name** "LOGIN"

**Data Type** sdtChar

##### **Description**

User name for access to application.

**Values**

Value	Description
User name	User name for entering the application

***NM\_PASSWORD***

**Attribute Name** "USER\_PASSWORD"

**Data Type** sdtChar

**Description**

Password.

**Values**

Value	Description
Password	Password for entering the application

***Script Hook Parameters*****Before Hook**

The following table describes the arguments passed in the Before OnLogin script hook.

**Arguments**

Argument	Description
ApplHndl	Input See Table 1.
SelectOp	Application name, for example, ‘SmarTeam – Editor’ or “Import/Export”
FirstPar	Not used
SecondPar	Not used
ThirdPar	Output login information, including user name and password.
Return Value	If script return value is unequal to Err_None or the user name and password were incorrect, then the login dialog is shown.

**After Hook**

The following table describes the arguments passed in the After OnLogin script hook.

**Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	Application name, for example, “SmarTeam – Editor” or “Import/Export”
FirstPar	Input containing the user name and password attributes
SecondPar	Not used
ThirdPar	Not used.
Return Value	In case the return value from the script is not equal to Err_None, the system start-up is aborted.

**Examples****Before OnLogin**

This script is designed for the Before OnLogin script hook. It bypasses the opening of the Login dialog and, if automatic login is set, automatically logs in the user according to user name and password.

```
Declare Sub CONV_RecListToComRecordList Lib "SmTdm32" (ByVal RecList As Long, ByRef COMRecList As ISmRecordList)
Declare Sub CONV_ComRecListToRecordList Lib "SmTdm32" (ByVal ComRecList As ISmRecordList, ByRef RecList As Long)
Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, Nsize As Long) As Long
Function OnLogin(ApplHndl As Long, Sstr As String, RecLst1 As Long, RecLst2 As Long, RecLst3 As Long ) As Integer
Dim Ret As Integer
Dim Lbuffer As String *255
Dim RetCode As Long
Dim SizeOf As Long
Dim AutoLogin As String
Dim hSubKey As Long
Dim dwType As Long, SZ As Long
Dim V As String
Dim Index As Integer
Dim Pos As Integer
Dim SmSession As Object
Dim COMFirstList As SmRecList.SmRecordList
Dim COMSecondList As SmRecList.SmRecordList
Dim COMThirdList As SmRecList.SmRecordList

'Getting the session object from the application handle
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
' Convert Record Lists to COM representation
CONV_RecListToComRecordList RecLst1, COMFirstList
CONV_RecListToComRecordList RecLst2, COMSecondList
CONV_RecListToComRecordList RecLst3, COMThirdList

AutoLogin = SmSession.Config.Value("$Local\Aurora\AutoLogin")
'Calling the DLL for SmarTools
'Call InitializeSmarTools(SmSession)
If AutoLogin = "FALSE" Then Exit Function
SizeOf = 255
RetCode = GetUserName(Lbuffer, SizeOf)
COMThirdList.ValueAsString("LOGIN", 0) = Lcase(Left(Lbuffer, SizeOf-1))
COMThirdList.ValueAsString(NM_PASSWORD, 0) = "mypassword"
' convert back
CONV_ComRecListToRecordList COMThirdList, RecLst3
OnLogin=Err_None
End Function
```

## After OnLogin

This script is designed for the After OnLogin script hook. It checks that the user that is trying to log into the application is the same one that logged in on Windows. If not, the login is aborted.

```
Declare Sub CONV_RecListToComRecordList Lib "SmTdm32" (ByVal RecList As Long, ByRef COMRecList As ISmRecordList)
Declare Sub CONV_ComRecListToRecordList Lib "SmTdm32" (ByVal ComRecList As ISmRecordList, ByRef RecList As Long)
Const OS_POWER_USER = "guy"
Const SmarTeam_POWER_USER = "joe"
Function AfterLogin(ApplHndl As Long, Sstr As String, RecLst1 As Long, RecLst2 As Long, RecLst3 As Long) As Integer
Dim Ret As Integer
Dim Lbuffer As String *255
Dim RetCode As Long
Dim SizeOf As Long
Dim EnteredLogin As String
Dim AutoLogin As String
Dim COMFirstList As SmRecList.SmRecordList
SizeOf = 255
' Convert Record Lists to COM representation
CONV_RecListToComRecordList RecLst1, COMFirstList
RetCode = GetUserName(Lbuffer, SizeOf)
EnteredLogin = COMFirstList.ValueAsString ("LOGIN", 0)
If (Lcase(Left(Lbuffer, SizeOf-1)) <> Trim$(EnteredLogin)) And
(Trim$(EnteredLogin) <> SmarTeam_POWER_USER) And
(Lcase(Left(Lbuffer, SizeOf-1)) <> OS_POWER_USER) Then
MsgBox "Access denied", vbInformation, "UserName/Password"
AfterLogin=ERR_GEN
End If
End Function
```



## ***Single Sign-On***

The SmarTeam Session Management Service centralizes authentication within the organization and provides a “single sign-on” mechanism. When a user logs in to a SmarTeam application, the OnLogin event is called and user authentication is performed. As a result of the single sign-on mechanism, the OnAuthenticate event is called instead of OnLogin when the user launches additional SmarTeam applications within the same work session. As a result of the introduction of the Single sign-on mechanism, the user is authenticated automatically and does not need to log in for each SmarTeam application that is launched.

## ***OnBrowse***

Executed when choosing ‘OnBrowse’ in a project view in SmarTeam – Editor and in the “Save as” dialog in integrations, for example: Documents view for specific project. This hook is used to prevent the user from viewing a class that is unauthorized to him. You prevent unauthorized access to a selected class by returning the appropriate error code.

## ***Script Execution Timing***

Stage	Timing
Before	Executed when invoking ‘OnBrowse’ for a specific class from a project view in SmarTeam – Editor and in the “Save as” dialog in the integrations.
After	Not applicable.
InsteadOf	Not applicable.

## ***Script Hook Parameters***

### **Before Hook**

The following table describes the arguments passed in the Before OnBrowse script hook.

### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.

SelectOp	'OnBrowse'
FirstPar	Input containing Class_ID and Object_ID of current project.
SecondPar	Input containing Class_ID of the class selected for browsing.
ThirdPar	Not used.
Return Value	If script return value is not equal to Err_None then the class that was selected is not opened for browsing.

---

## ***Examples***

### **Before OnBrowse**

This script is added to the Before OnBrowse script hook in Script Maintenance for the Class Browser to prevent browsing of a specific class (in this case the Document class) by users that are not administrators. This script works only for the SmDemo Database.

```
Function BeforeBrowseExample(ApplHndl As Long,Sstr As String,FirstPar As Long,SecondPar As Long,ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim SecondRec As Object
    Dim ThirdRec As Object
    Dim UserAuthorizedForDocClass As Boolean

    ' Convert pointer to COM object SmSession
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert input parameter to COM object
    CONV_RecListToComRecordList SecondPar,SecondRec

    'True if user is administrator
    UserAuthorizedForDocClass = SmSession.UserMetaInfo.UserAdmin
    DocumentClassId =
SmSession.MetaInfo.SmClassByName("Documents").ClassId
    If (SecondRec.ValueAsSmallint(NM_CLASS_ID, 0) = DocumentClassId) Then
        If UserAuthorizedForDocClass Then
            ' return no error to continue browse operation
            BeforeBrowseExample = Err_None
        Else
            ' return error to cancel browse operation
```

```
BeforeBrowseExample = Err_Gen
MsgBox "User not authorized to browse Documents class"
End If
Else
BeforeBrowseExample = Err_None
End If
End Function
```

## ***OnRetrieveObjects***

The OnRetrieveObjects script hook is executed after any mass retrieval from the database, for example:

Retrieving relations of the specific object

Retrieving children of the specific object

Executing user-defined queries

Other queries

This hook only has an After stage, which occurs after the retrieval operation is complete but before the objects are displayed in a SmarTeam view. The script receives, as input, the list of objects that have been retrieved and are about to be displayed. On the basis of authorization criteria, the script can prevent specific objects from being displayed in SmarTeam views.

Non-authorized objects are removed from the display by the script in one of two ways, according to the value of the input attribute NM\_CHECK\_AUTHORIZATION\_MODE:

They can be deleted from the view record list FirstPar

The script can direct the system not to show specific objects, without deleting them from FirstPar.

For details, see the section NM\_CHECK\_AUTHORIZATION\_MODE below.

### ***Script Execution Timing***

Stage	Timing
Before	Not applicable.
After	Executed after each retrieval from the database.
InsteadOf	Not applicable.

### ***OnRetrieveObject Hook Attributes***

The following attributes can be input to the OnRetrieveObject script.

#### ***NM\_OPER\_NAME***

**Attribute Name**                      “OPERATION\_NAME”

**Data Type**                              sdtChar

#### **Description**

Name of the operation doing the retrieval.

One of:

GetTopLevelObjects

GetChildren

GetGeneralLinks

GetOneLevelLinkRelations

GetDependencies

GetReverseDependencies

GetMultiFileReferencies (CFO)

GetRelatives

GetTopmostObjects

Query By Attribute

Query By Example

Note: Simple Query is not included, so that you can use it in your script without it triggering the OnRetrieve hook again.

### ***VIEW\_COL\_LEAD\_CLASS\_ID***

**Attribute Name** "LEAD\_CLASS\_ID"

**Data Type** sdtSmallInt

#### **Description**

Class Id of the object from which query was started (For instance: When retrieving links of a specific object, the Class\_ID of that object is passed).

### ***VIEW\_COL\_LEAD\_OBJ\_ID***

**Attribute Name** "LEAD\_OBJECT\_ID"

**Data Type** sdtObjectIdentifier

#### **Description**

Object Id of object from which query was started (For instance: When retrieving links of a specific object, the Object\_ID of that object is passed).

### ***NM\_CLASS\_ID***

**Attribute Name** "CLASS\_ID"

**Data Type** sdtSmallInt

#### **Description**

Super class of all classes that are included in the query results.

## ***NM\_CHECK\_AUTHORIZATION\_MODE***

**Attribute Name** "CHECK\_AUTHORIZATION\_MODE"

**Data Type** sdtSmallInt

### **Description**

Indicates the authorization mode for this query. It specifies if and how you can cause an individual object not to be displayed on a tree display.

When the value is voaCheckAndDeleteFromList, you delete the object from FirstPar.

When the value is voaCheckAndSignInList, you get an additional Boolean attribute, NM\_ AUTHORIZED\_OBJ, in FirstPar. Set it to True or False to cause the object to be displayed or not. You cannot delete the object from FirstPar in this case. This value is usually passed to the OnRetrieveObjects hook in lifecycle retrieve operations, where deleting the object is not desirable.

### **Values**

<b>Value</b>	<b>Description</b>
voaNotToCheck = 0	You cannot cause an object not to be displayed.
voaCheckAndDeleteFromList = 1	You can delete objects from FirstPar list to prevent them from being displayed.
voaCheckAndSignInList = 2	You cannot delete an object from FirstPar, but you are permitted to set 'TDM_OBJ_AUTHORIZATION' value in FirstPar list to False, so that the object is not displayed in the opened view.

## ***NM\_ AUTHORIZED\_OBJ***

**Attribute Name** "TDM\_OBJ\_AUTHORIZATION"

**Data Type** sdtBoolean

### **Description**

Authorizes current object for display.

**Note:** The FirstPar record list contains the 'TDM\_OBJ\_AUTHORIZATION' attribute only when the value of 'CHECK\_AUTHORIZATION\_MODE' is set to "voaCheckAndSignInList"

**Values**

Value	Description
False	Not authorized for display
True	Authorized for display

***Script Hook Parameters*****After Hook**

The following table describes the arguments passed in the After OnRetrieveObjects script hook.

**Arguments**

Argument	Description
ApplHndl	Input See Table 1.
SelectOp	'RetrieveObjects'
FirstPar	Input record list that contains objects that were found as a result of retrieval. Used also for output  Note that the attribute header name can be prefixed with the class ID of the object's superclass, such as:  1.CN_DESCRIPTION = Suppliers Documentation  1.STATE = 0  1.USER_OBJECT_ID = 1
SecondPar	Input record list that contains information about current query and authorization, See OnRetrieveObject Hook Attributes.
ThirdPar	Not used.



## **Examples**

### **After OnRetrieveObjects**

This script is added to OnRetrieveObjects script hook for Project classes to prevent viewing or retrieving specific project information by users that are not administrators. Works only for SmDemo Database.

```
Declare Sub CONV_RecListToCOMREcordList Lib "Smtdm32" (ByVal RecList As Long, ByRef COMRecList As SmREcList.SmRecordList)
Declare Sub CONV_COMRecListToREcordList Lib "Smtdm32" (ByVal COMRecList As SmREcList.SmRecordList, ByRef RecList As Long)

'Look in Db explorer or retrieve object Id for the project which you want to secure.
Const SecuredProjectObjectId = 3410
Function OnRetrieveExample(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim SecondRec As Object
    Dim Shift As Integer
    Dim iR As Integer
    Dim AuthorizationMode As Integer

    ' Convert pointer to COM object SmSession
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
    ' Convert input parameter to COM object
    CONV_RecListToComRecordList FirstPar,FirstRec
    CONV_RecListToComRecordList SecondPar,SecondRec
    AuthorizationMode =
SecondRec.ValueAsSmallInt("CHECK_AUTHORIZATION_MODE",0)
    'If user is administrator, exit scripts without securing class
    If SmSession.UserMetaInfo.UserAdmin Then
        Exit Function
    End If
    Shift = 0
    'find records to hide, according to its object Id
    'Object_ID can be prefaced by superclass or not
    'First look for superclass preface: Project superclass = 1
    If FirstRec.Headers.HeaderExists("1.OBJECT_ID") Then
        'find records to hide, according to its object Id
        For iR = 0 To FirstRec.RecordCount - 1
```

```

        If FirstRec.ValueAsInteger("1.OBJECT_ID",iR - Shift) =
SecuredProjectObjectId Then
            If AuthorizationMode = VoaCheckAndDeleteFromList Then
                'delete it from record list and take into account record shift
                FirstRec.DeleteRecord iR - Shift
                Shift = Shift + 1
            Else
                If AuthorizationMode = VoaCheckAndSignInList And
FirstRec.Headers.HeaderExists("1.TDM_OBJ_AUTHORIZATION") Then
                    'false, hide it
                    FirstRec.ValueAsBoolean("1.TDM_OBJ_AUTHORIZATION",iR -
Shift) = False
                End If
            End If
        End If
    Next
    'Now look without superclass preface
    Shift = 0
    If FirstRec.Headers.HeaderExists("OBJECT_ID") Then
        'find records to hide, according to its object Id
        For iR = 0 To FirstRec.RecordCount - 1
            If FirstRec.ValueAsInteger("OBJECT_ID",iR - Shift) =
SecuredProjectObjectId Then
                If AuthorizationMode = VoaCheckAndDeleteFromList Then
                    FirstRec.DeleteRecord iR - Shift    'delete it from rec list
                    Shift = Shift + 1
                Else
                    If AuthorizationMode = VoaCheckAndSignInList And
FirstRec.Headers.HeaderExists("1.TDM_OBJ_AUTHORIZATION") Then
                        'hide it
                        FirstRec.ValueAsBoolean("TDM_OBJ_AUTHORIZATION",iR - Shift)
= False
                    End If
                End If
            End If
        Next
    End If
    CONV_ComRecListToRecordList FirstRec, FirstPar
    CONV_ComRecListToRecordList SecondRec, SecondPar
    OnRetrieveExample = Err_None
End Function

```

---

## Scripts for CAD Identification

The SmarTeam CAD integration software provides a script hook for retrieving object information from the SmarTeam database about an object currently displayed in the CAD tool. This script hook is only necessary in special installations and is normally not required.

This script should be written in coordination with a SmarTeam support person.

### ***CAD Integration Script Hooks***

The following CAD Integration script hooks are available:

Script Hook	Applicable to	Occurs
CAD Identification	CAD operations within CAD tool	When CAD operation is invoked

### ***CAD Integration***

#### ***Script Execution Timing***

Stage	Timing
Before	Not applicable.
After	Not applicable.
InsteadOf	The script is invoked instead of the CAD integration software that retrieves object information from the SmarTeam database.

## Script Hook Parameters

### Arguments

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	“
FirstPar	Input containing the basic CAD object attributes such as CAD file name and directory.
SecondPar	Not used.
ThirdPar	Output.  SmarTeam attributes corresponding to the CAD object including Object_ID.
Return Value	If the script returns Err_Refuse, the operation is aborted.

---

## Scripts in Flow Processes

In order to resolve possible inconsistency problems for flow process for events **OnOpen** and **BeforeSend**, please use the following examples.

### OnOpen Hook

In a Flow Process script (**onOpen** hook), the following parameters are passed:

```
FlowSession As Object, FlowProcess As Object, Node As Object
```

To resolve inconsistency problems, add the following code into your customization code for the hook **OnOpenFlowProcess**:

**Note:** This code is only necessary when calling the method GetFollowingNodes.

```
Dim UpdatedQueueItem as ISmFlowQueueItem
```

```
FlowSession.InboxProcesses.Refresh  
Set UpdatedQueueItem = FlowSession.InboxProcesses.ItemByNode(Node)  
If UpdatedQueueItem Is Nothing then  
    Exit Sub  
End if
```

```
Set UpdatedFlowProcess = UpdatedQueueItem.FlowProcess  
Set UpdatedNode = UpdatedQueueItem.Node
```

Use the updated parameters **UpdateFlowProcess** and **UpdatedNode** instead of received previous parameters (**FlowProcess** and **Node**) to solve the inconsistency problems.

### ***Before Send Hook***

In a Flow Process script (**BeforeSend** hook), the following parameters are passed:

ActiveProcess As Object, Response As Object

Add the following code into your customization code:

```
Set FlowSession = ActiveProcess.FlowSession  
Set Node = ActiveProcess.QueueItem.Node  
  
Dim UpdatedQueueItem as ISmFlowQueueItem  
  
FlowSession.InboxProcesses.Refresh  
Set UpdatedQueueItem = FlowSession.InboxProcesses.ItemByNode(Node)  
If UpdatedQueueItem Is Nothing then  
    Exit Sub  
End if  
  
Set UpdatedActiveProcess = UpdatedQueueItem.ActiveProcess
```



## 6. Scripts for Import/Export Operations

By attaching a script to a specific import or export operation, you can modify the information that is being imported or exported during run time.

For example:

You can abort the import or export of a specific object

You can change or add values to specific imported or exported object attributes, e.g., change upper case letters to lower case.

You can limit the number of imported objects for a class by using a filter criterion. For example, filter out all objects whose ID value is above Doc-500.

### ***Virtual Attributes***

In addition to ordinary object attributes defined in the SmartWizard setup, the Import/Export facility recognizes *virtual* attributes.

A virtual attribute is a special attribute created in the Import/Export facility to be used in a filter criterion in a script attached to the Import or Export process. You might create a virtual attribute in case the attribute value in the external device has a different format than it has in the SmarTeam database. Thus you would create a virtual attribute vSTATE, corresponding to the SmarTeam attribute STATE, where vSTATE would hold the attribute value in its external format. See the example below.

You cannot change the value of a virtual attribute by a script.

An import of a particular object can be rejected based on any value of any attribute, real or virtual.

### ***Import/Export Script Hooks***

The following Import/Export script hooks are available:

Script Hook	Applicable to	Occurs
OnImport	Import operation	During import of each record
OnExport	Export operation	During export of each record

## ***Attaching a Script to an Import/Export Script Hook***

Once the script has been written, you use the Import or Export utility to attach it to the desired script hook.

### **Import**

To attach a script to an import operation:

Open the Import Utility

In the Import window, select a Class and click Tasks button to display the Import to Class window

In the Import from Class window, click Attach Script to display the Script Browser window

Select a file and a script and click Ok.

### **Export**

To attach a script to an export operation:

Open Export utility.

In the Export window, select an Export entry.

Click on the "Options" button under the Add Query button to open the Export Query attributes window.

In the Export Query attributes window click Attach Script to display the Script Browser window

Select a file and a script and click Ok.

**Note:** When defining a script within the Export tool, it appears in Script Maintenance as a User Defined Tool named after the export created. You may delete the Script Maintenance user-defined tool by deleting the specific export from the Export tool.

See the SmarTeam – Editor Administrator's Guide for more information.

## ***OnImport, OnExport***

The timing and parameters are the same for both the OnImport and OnExport script hooks.

### ***Script Execution Timing***

The export script hook is executed prior to writing each object record to the export file.



The import script hook is executed prior to writing each object record to the database.

## ***Script Hook Parameters***

### **Representing a Link Object**

In case the object is a link object, the record list parameter also contains the Class Id and primary identifiers of each linked object. For example, when importing a general link between two projects in SmDemo database the record list would contain the following elements.

```
1;CN_PROJECT_ID
1;CLASS_ID
2;CN_PROJECT_ID
2;CLASS_ID
STATE
CLASS_ID
...
```

where 2;CN\_PROJECT\_ID, 2;CLASS\_ID, 1;CN\_PROJECT\_ID and 1;CLASS\_ID are identifiers of the related projects, and STATE, CLASS\_ID, ... are attributes of the link between these two projects.

### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	“
FirstPar	Input containing Imported/Exported object attributes.
SecondPar	Virtual object attributes.
ThirdPar	Output.  You can change or add the value of an imported or exported attribute.
Return Value	If the script returns Err_Refuse, import or export of the specific object is not carried out.

## Examples

### OnImport

The following script is for the OnImport script hook. This script assumes that the imported class has an attribute called "CN\_MANAGER", which is referenced to the class "USERS".

In the import utility, a virtual attribute called "vManager" is defined to be the login name of the manager. The script checks this value, gets its Object\_ID from the SmarTeam database by a query, and returns the Object\_ID value in the third record list of the script.

```
Function GetManagerObjectID(ApplHndl As Long, Sstr As String, FirstPar As Long, SecondPar As Long, ThirdPar As Long ) As Integer
    Dim Session As ISmSession
    Dim FirstRL As Object
    Dim SecondRL As Object
    Dim ThirdRL As Object
    Dim QueryResult As Object
    Dim tmpStr As String
    Dim SmQuery As ISmSimpleQuery
    Dim UsersClassID As Integer
    Set Session = SCREXT_ObjectForInterface(ApplHndl)
    CONV_RecListToComRecordList FirstPar, FirstRL
    CONV_RecListToComRecordList SecondPar, SecondRL
    CONV_RecListToComRecordList ThirdPar, ThirdRL
    UsersClassID = Session.MetaInfo.SmClassByName("Users").ClassId
    tmpStr = SecondRL.Value("vManager", 0)
    Set SmQuery = Session.ObjectStore.NewSimpleQuery
    SmQuery.SelectStatement = "SELECT OBJECT_ID FROM " & Str(UsersClassID) &
    " WHERE LOGIN = '" & tmpStr & "'"
    SmQuery.Run
    Set QueryResult = SmQuery.QueryResult
    If QueryResult.RecordCount = 0 Then
        GetManagerObjectID = ERR_REFUSE
    Else
        ThirdRL.AddHeader "CN_MANAGER", 4, sdtInteger
        ThirdRL.Value("CN_MANAGER", ThirdRL.AddRecord) =
        QueryResult.Value("OBJECT_ID", 0)
        GetManagerObjectID = ERR_NONE
    End If
    CONV_COMRecListToREcordList ThirdRL, ThirdPar
End Function
```

## **7. Scripts for User-Defined Commands**

You can define script-based “user-defined” commands. These can be made to apply to specific SmarTeam applications, for example, to the SmarTeam main application or to the SolidWorks integration. They can also be defined to apply to a specific class, for example, to all objects of type Document.

### ***Attaching a Script to User-Defined Operations***

The commands are defined in the Script Maintenance utility in the “User Defined” tab. You define an operation name and attach a script that contains the command’s functionality. You also select the classes for which the new command is applicable.

The way in which the user-defined operation, which you defined in Script Maintenance, is activated depends on the menu preference: hard-coded menus or profile-based menus. (This setting is selected in the Administrator options menu.)

#### **Hard-Coded Menus**

In case the user is working with hard-coded menus these commands will appear under the “User Defined Tools” menu item in the popup menu and in the Tools/User Defined Tools menu item of the main menu of the SmarTeam application.

#### **Profile-Based Menus**

In case you are working with profile-based menus, you can have these commands appear in the menus and toolbars defined in the configurable Menu Profiles. Use the Menu Editor utility to define the corresponding menus.

### ***User-Defined Command Script Hook***

#### ***Script Execution Timing***

The script is executed when the user-defined command is invoked.

## ***Script Hook Parameters***

### **Arguments**

Argument	Description
ApplHndl	Input. See Table 1.
SelectOp	User-defined command operation name as defined in Script Maintenance
FirstPar	Input containing selected object attributes, including at least Class_ID, Object_ID and STATE of the selected objects.
SecondPar	Input record list containing the Class_ID of the current object, (if only one is selected) or class common to all selected objects.
ThirdPar	Not used.
Return Value	If the script returns an error code other than Err_None, the operation is not aborted.

### ***Examples***

#### **User-Defined Command**

The following script is for a user-defined command. Using ScriptMaintenance and Menu Editor, it can be added to a SmarTeam tool button. The command displays a specific predefined Project in a SmarTeam View--without the need to perform a search.

```
Function UserDefined(ApplHndl As Long,Sstr As String,FirstPar As Long,  
SecondPar As Long, ThirdPar As Long ) As Integer  
    Dim SmSession As SmApplic.SmSession  
    Dim SingleSmObject As SmApplic.ISmObject  
    Dim GUI As SmGUISrv.SmCommonGUI  
    Dim View As SmGUISrv.ISmView  
    'Get the Class ID and Object ID for Project-0013 Oil separator  
    Const CLASS_ID = 459  
    Const OBJECT_ID = 3410  
  
    On Error GoTo Handle  
    ' Convert pointer to COM object SmSession  
    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
```

```
Set GUI = SmSession.GetService("SmGUISrv.SmCommonGUI")
Set SingleSmObject = SmSession.RetrieveObject(CLASS_ID,OBJECT_ID)
Set View = GUI.Views.NewViewByType(vwtSingleObject)
Set View.DisplayObjects.SingleObject = SingleSmObject
View.SmViewWindow.ShowModal
UserDefined = Err_None
Exit Function

Handle:
    UserDefined = Err_Gen
End Function
```

## ExecuteOperationOnTrees

The following script is for a user-defined command. Using ScriptMaintenance and Menu Editor, it can be added to a SmarTeam tool button. The command checks in a set of Folder and Document object trees where the leading object of each tree has been specified, for example by selecting them on a tree View. Folders are checked in with CHECKINMODE = LFCYC\_PrevRev and the Documents are checked in with the default CHECKINMODE = LFCYC\_WorkRev. Thus, the Document revisions advance while the Folder revision stays the same.

The example uses the function ExecuteOperationOnTrees .The function has three record list parameters: LeadingObjects, TaskRL, DefaultTask.

The LeadingObjects record list generally contains multiple objects.

See the API document for more information on this function.

```
Function CheckInTrees(ApplHndl As Long,Sstr As String,FirstPar As Long,SecondPar As Long,ThirdPar As Long ) As Integer
    Dim SmSession As SmApplic.SmSession
    Dim FirstRec As Object
    Dim SecondRec As Object
    Dim ThirdRec As Object
    Dim Operation As SmApplic.ISmOperation    'performed operation object
    Dim MetaInfo As SmApplic.SmMetaInfo 'metainfo object for smClasses
    Dim SessionUtil As SmUtil.SmSessionUtil    'SmarTeam service object
    Dim DefaultTask As Object 'default task record list for operation
    Dim TaskRL As Object 'task record list for operation per object
    Dim LeadingObjects As SmApplic.ISmObjects
    Dim OperName As String 'operation name
    Dim LeadingObject As SmApplic.ISmObject
    Dim QueryDefinition As SmApplic.ISmQueryDefinition
```

```

Dim Children As SmApplic.ISmObjects ' collection of objects linked
Dim Child As Object
Dim FolderClassId As Integer
Dim LOIndex As Integer
Dim TaskRLIndex As Integer
Dim RecordCount As Integer
Dim ChildrenCount As Integer
Dim Result As Long
Dim k As Integer
On Error GoTo ErrorHandler
' Convert pointer to COM object SmSession
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)
Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")
Set Metainfo = SmSession.Metainfo
FolderClassId = Metainfo.SmClassByName("Folder").ClassId
' Set up LeadingObjects
Set LeadingObjects = SmSession.ObjectStore.NewObjects
'Put NM_OPER_ID in object list - applies to all objects
LeadingObjects.Data.AddHeader NM_OPER_ID, 2, sdtSmallInt
'Put NM_PROPAGATED in object list - applies to all objects
LeadingObjects.Data.AddHeader NM_PROPAGATED, 2, sdtSmallInt
'Set up TaskRL Record List
Set TaskRL=CreateObject("SmRecList.SmRecordList")
TaskRL.AddHeader NM_CLASS_ID, 2, sdtSmallInt
TaskRL.AddHeader NM_OBJECT_ID,SIZE_OBJ_ID, sdtInteger
TaskRL.AddHeader NM_OPER_ID,2, sdtSmallInt
TaskRL.AddHeader NM_LFCYC_CHECKIN_MODE, 2, sdtSmallInt
TaskRL.AddHeader NM_DSC_NOTES, 256, sdtChar
'Set up DefaultTask Record List
Set DefaultTask=CreateObject("SmRecList.SmRecordList")
'Put NM_OPER_ID in DefaultTask list - applies to all objects
DefaultTask.AddHeader NM_OPER_ID, 2, sdtSmallInt
'Put NM_PROPAGATED in object list - applies to all objects
DefaultTask.AddHeader NM_PROPAGATED, 2, sdtSmallInt
DefaultTask.AddHeader NM_DSC_NOTES, 256, sdtChar
' Conver input parameter to COM object
CONV_RecListToComRecordList FirstPar,FirstRec
CONV_RecListToComRecordList SecondPar,SecondRec
CONV_RecListToComRecordList ThirdPar,ThirdRec

OperName = NM_OPER_CHECKIN

```

```
Set Operation = MetaInfo.SmOperationByName(OperName)
' Default tasks record for operation - necessary
DefaultTask.ValueAsSmallInt(NM_OPER_ID,0) = Operation.Id
DefaultTask.ValueAsSmallInt(NM_PROPAGATED,0) = 1
DefaultTask.ValueAsString(NM_DSC_NOTES,0) = "Checkin DefaultTask
notes"
RecordCount = FirstRec.RecordCount
If RecordCount <> 0 Then
    TaskRLIndex      = 0
    For LOIndex = 0 To RecordCount-1 'loop over leading objects
        Set LeadingObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(LOIndex),true)
        Result = LeadingObjects.Add(LeadingObject)
        Set Operation =
MetaInfo.OperationsForClass(LeadingObjects.Item(LOIndex).ClassId,
False).ItemByName(OperName)
        LeadingObjects.Data.ValueAsSmallInt(NM_OPER_ID,LOIndex) =
Operation.Id
        LeadingObjects.Data.ValueAsSmallInt(NM_PROPAGATED,LOIndex) = 1
        'fill TaskRL list for leading object
        TaskRL.ValueAsSmallInt(NM_CLASS_ID,TaskRLIndex) =
LeadingObject.ClassId
        TaskRL.ValueAsInteger(NM_OBJECT_ID,TaskRLIndex) =
LeadingObject.ObjectId
        TaskRL.ValueAsSmallInt(NM_OPER_ID,TaskRLIndex) = Operation.Id
        TaskRL.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE,TaskRLIndex) =
LFCYC_PrevRev
        TaskRL.ValueAsString(NM_DSC_NOTES,TaskRLIndex) = "Checkin TaskRL
notes"
        TaskRLIndex = TaskRLIndex + 1
        ' Retrieve LeadingObject's children
        Set QueryDefinition = Nothing
        Set Children = LeadingObject.RetrieveChildren(QueryDefinition)
        ChildrenCount = Children.Count
        ' fill task list for children - same operation as parent
        If ChildrenCount <> 0 Then
            For k = 0 To ChildrenCount-1
                Set Child = Children.Item(k)
                'check in folders with previous revision
                If Child.ClassId = FolderClassId Then
                    TaskRL.ValueAsSmallInt(NM_CLASS_ID,TaskRLIndex) =
Child.ClassId
```

```
TaskRL.ValueAsInteger(NM_OBJECT_ID,TaskRLIndex) =  
Child.ObjectId  
TaskRL.ValueAsSmallInt(NM_OPER_ID,TaskRLIndex) =  
Operation.Id  
TaskRL.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE,TaskRLIndex)  
= LFCYC_PrevRev 'check in folder as previous  
TaskRL.ValueAsString(NM_DSC_NOTES,TaskRLIndex) = "Checkin  
TaskRL notes"  
TaskRLIndex=TaskRLIndex + 1  
End If  
Next  
End If  
Next  
End If  
If Not (LeadingObjects Is Nothing) Then  
    SessionUtil.ExecuteOperationOnTrees LeadingObjects, TaskRL,  
    DefaultTask  
End If  
CheckInTrees = Err_None  
Exit Function  
ErrorHandler:  
    MsgBox Err.Description  
    CheckInTrees = Err_Gen  
End Function
```



## Appendix A Attributes Passed by SmarTeam – Editor

Table 11 shows common, useful object attributes that are frequently passed by SmarTeam – Editor to a script through FirstPar. The particular attributes that are passed to the script depend on which operation is being performed.

If your script is to run under a different application than SmarTeam – Editor, you need to verify which attributes will be passed to the script.

To verify if a specific attribute is passed to a script in a record list parameter, use the function

```
IsmRecordList.Headers.HeaderExists ("HeaderName")
```

where HeaderName is the attribute name appearing in the left column of Table 11.

**Warning:** Many of the attributes in Table 11 are used internally for SmarTeam operations and should not be changed in the script. They are for information only.

*Table 11 Common Object Attributes Passed to a Script*

Attribute Attribute Name	Description	Data Type
<b>Super/Internal Class</b>		
NM_OBJECT_ID 'OBJECT_ID'	Object ID	sdtObjectIdentifier
NM_CLASS_ID 'CLASS_ID'	Class ID	sdtSmallInt
NM_STATE 'STATE'	Life-Cycle state:  New Checked In Checked Out Released Obsolete	sdtObjectIdentifier Lookup table
NM_CREATION_DATE 'CREATION_DATE'	Date and time of object creation	sdtTimeStamp
NM_USER_OBJ_ID 'USER_OBJECT_ID'	User (in class Users) that created the object	sdtObjectIdentifier

NM_USER_ID_MOD 'USER_ID_MOD'	User (in class Users) that modified the object	sdtObjectIdentifier
NM_MODIFICATION_DATE 'MODIFICATION_DATE'	Date and time of object modification	sdtTimeStamp
<b>Revision Control</b>		
NM_REVISION 'REVISION'	Revision of current object	sdtChar
NM_PHASE 'PHASE'	Name of the phase for the revision of the object under the current operation:  Default Preliminary Design Design Prototype Production	sdtObjectIdentifier Lookup table
NM_PAR_REVISION 'PAR_REVISION'	Previous revision	sdtChar
NM_APPROVAL_DATE 'APPROVAL_DATE'	Date and time of approval (release)	sdtDate
NM_EFFECTIVE_FROM 'EFFECTIVE_FROM'	“Effective From” date of the revision of the object	sdtEffectiveDateFrom
NM_EFFECTIVE_UNTIL 'EFFECTIVE_UNTIL'	“Effective Until” date of the revision of the object	sdtEffectiveDateUntil
NM_REVISION_STG 'REVISION_STG'	0 – Not last public revision 1 – Last public revision	sdtInteger
NM_ORG_USER_ID 'TDM_ORG_USER_ID'	Object ID of user that created the first revision of this object	sdtObjectIdentifier
NM_ORG_CREATEDATE 'TDM_ORG_CREATEDATE'	Date of creation of the first revision of this object	sdtTimeStamp
NM_APPROVED_BY 'TDM_APPROVED_BY'	Object ID of user that approved this revision	sdtObjectIdentifier
<b>File Control</b>		
NM_FILE_TYPE 'FILE_TYPE'	File Type General, SolidWorks Part, ...	sdtObjectIdentifier Lookup table:
NM_FILE_NAME 'FILE_NAME'	File Name	sdtChar

---

---

**Attributes Passed by SmarTeam – Editor**

---

NM_DIRECTORY 'DIRECTORY'	Directory name	sdtChar
NM_CAD_REF_FILE_NAME 'CAD_REF_FILE_NAME'	Original file name of the file.	sdtChar
NM_CAD_REF_DIRECTORY 'CAD_REF_DIRECTORY'	Original directory of the file.	sdtChar
NM_CAD_DIRTYFLAG 'CAD_DIRTYFLAG'	A long representation of the modification date of the file. The attribute can be updated by an integration, or any external application	sdtInteger

---