



ENOVIA SmarTeam | Dassault Systèmes

www.smarteam.com

www.3ds.com

ENOVIA SMARTTEAM

***Customizing Using Server-Side
Hooks for
Server-Based Applications***

Programmer's Guide

Important Notice

© Dassault Systèmes, 2004, 2008. 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-A2-180007

Table of Contents

Important Notice	iii
Table of Contents	iv
1. Introduction	1-1
Script Hooks in Server-Based Applications	1-1
Setting Server Mode for an Application	1-2
Using SmarTeam Hooks in Server Mode	1-2
SmarTeam Hooks Available in Server Mode	1-3
Two SmarTeam Hook Interfaces	1-5
2. Quick Start.....	2-1
3. Standard SMARTEAM Hook Interface	3-1
Naming Hook Functions	3-1
Running a Function not Specified in SmarTeam	3-4
Hook Function Parameters	3-4
Library-Specific Hooks	3-7
Packaging the Hook Functions into an Interface Object.....	3-9
Integrating the COM Component.....	3-9
4. Low-Level SMARTEAM Hook Interface	4-1
Implementing the Interface Class ISmServerHook.....	4-1
Hook Types	4-2
Init Method.....	4-4
Type1HookExists Method.....	4-4
Type1Execute Method	4-6
Type2HookExists Method.....	4-8
Type2Execute Method	4-8
Running a Function not specified in SmarTeam	4-10
Naming Hook Functions	4-11
Hook Function Parameters	4-11
Packaging the Functions into an Interface Object.....	4-11
Integrating the COM Component.....	4-11

Using **SmarTeam** Hooks in Server Mode

1. Introduction

This document describes how to use the SmarTeam API in Server Mode. In particular, it describes interface mechanisms provided by SmarTeam to allow you to use the SmarTeam hooks in Server Mode.

Any application that works in Server Mode uses the server-side hooks described in this document. Examples are: SmartWeb, EmbeddedScript, and any user application for which the ServerMode flag is set to True.

This document assumes that you are familiar with the following topics, which are described in the documents Script Hooks and COM API Programmer's Guide:

- Script event hooks, on which scripts can be attached

- The two types of script hooks, generic and library-specific, and the format of the script argument structure for each type of hook

- Recommendations on how to write a script

Script Hooks in Server-Based Applications

The mechanism by which script hooks are activated in server-based applications is different from that of client-based applications.

In client-based applications, the scripts are registered in the Script Maintenance facility on the client side and are invoked by the client-side application. See the document Client-Side Hooks for Client-Based Application for more information.

In server-based applications, the code is located on the server-side along with the application. When the client activates an operation on the server application that involves calling a hook, the server application calls the relevant code.

The diagram below shows how the mechanism works.

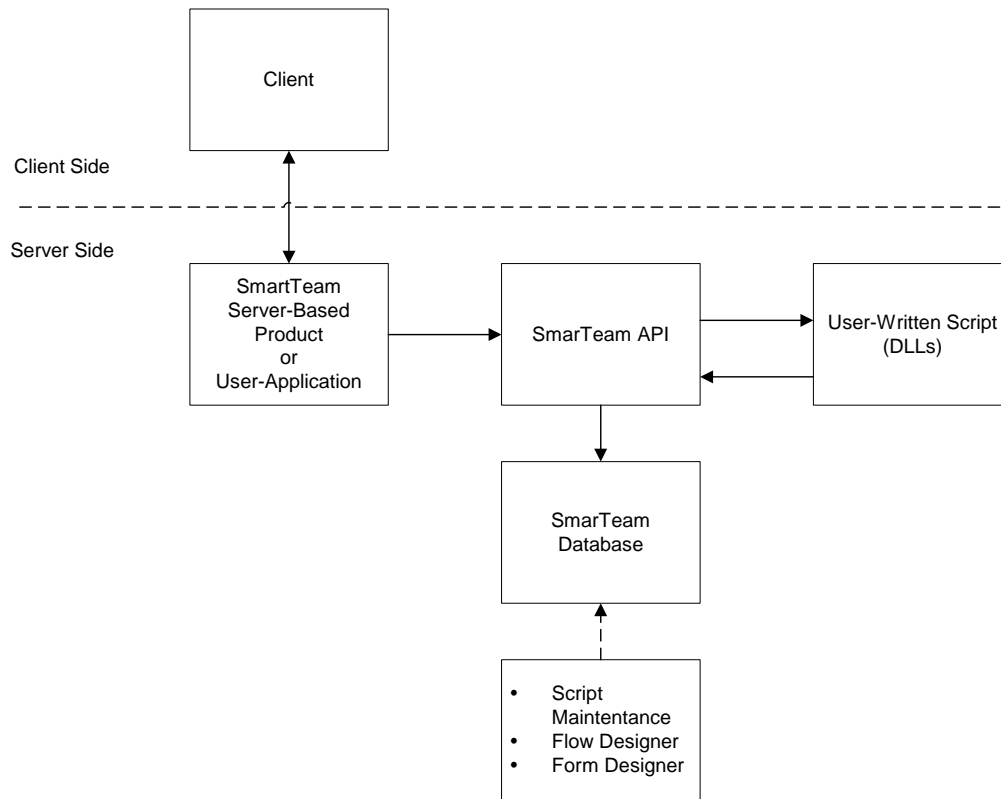


Figure 1 Script Hooks in SmarTeam Server-Side Applications

Setting Server Mode for an Application

An application using the SmarTeam API can initialize SmarTeam to work in Server Mode by setting the property `ServerMode` of the `SmFreeThreadedEngine` object to `True`:

```
SmEngine.ServerMode = True
```

For more information about the `ServerMode` property, see the SmarTeam API documentation.

Using SmarTeam Hooks in Server Mode

You use SmarTeam hooks in Server Mode differently than you do in Stand-alone Mode in several important ways. These are listed below and detailed in the following sections:

The term “hook function” is used to refer to the code that is attached to a SmarTeam hook.

Existing BasicScripts scripts that work in non-Server Mode will not operate in Server Mode. They need to be converted using COM-compatible scripting language such as VBScript or into a COM-compatible development tool such as Visual Basic.

You can run hook functions that were not configured in SMARTEAM by specifying the hook’s event-based name, following the convention described in the section Naming Hook Functions.

You cannot use any of the SmarTeam GUI-based hooks, such as hooks for Controls and Forms Events, in Server Mode

SmarTeam Hooks Available in Server Mode

The same set of script hooks are available for use in the Server Mode as in the non-Server Mode, except for those hooks involving a GUI.

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

The following is a list of hooks available in the Server Mode:

Table 1 ¹Script Hooks for SmarTeam Operations

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

¹ Note that a SmarTeam server application may not implement all of these Server Hooks. Check the documentation for the server application to see which Server Hooks are implemented.

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	

File Operations			
Edit	X	X	X
View	X	X	X
RedLine	X	X	X
Print	X	X	X
Copy File	X	X	X

Authorization Operations			
OnRetrieveObjects		X	
OnLogin	X	X	

CAD Operations			
Object identification for CAD	X		X

² WorkFlow Operations
³ On Receive
OnSendBefore
OnSendAfter

Two SmarTeam Hook Interfaces

SmarTeam provides two types of hook interfaces that allow you to use SmarTeam hooks in Server Mode:

- A standard interface, which is simple and easy to use

- A low-level interface for maximum flexibility and control

In the standard interface, you only need to supply the hook functions. You package them as COM Automation object functions, and you use a specific function naming convention, as described in the section Naming Hook Functions. The system executes the hook functions at the appropriate time according to the fixed rules implemented in the `SmartServerHookStd.SmDispatch` module provided by SMARTEAM.

In the low-level interface, in addition to the hook functions, you need to supply an object which implements the `ISmServerHook` interface. In the implementation, you determine the rules by which the hook functions are executed. Thus, this interface is useful when you require that the hook functions execute in a different way than that provided with the standard interface. For additional details, see the section Low-Level SmarTeam Hook Interface.

² See the *SmarTeam Object Model Programmer's Guide* for information about Workflow Task scripts.

³ For the StartNode of a Flow Process in Server Mode, use the OnReceive hook instead of the client-side OnOpen hook.

2. Quick Start

This section provides a quick tutorial of how to attach a function to a SmarTeam hook in Server Mode using Visual Basic:

1. In SmarTeam, using the Script Maintenance tool, assign a dummy script named “BeforeUpdateHook” with the generic parameter list to the “Before Update” hook.
2. Create a new Visual Basic ActiveX DLL project.
3. Create a VB Class named “Sample”.
4. Add a reference to the libraries “SmarTeam Engine Library” (SmApplic.DLL), “SmarTeam Record List Library” (SmRecList.DLL), and “Smart Server Hook Library” (SmartServerHook.TLB)
5. Add a function to the Class called “BeforeUpdateHook” (the same name as the one assigned in Script Maintenance), with the following function signature:

```
Public Function BeforeUpdateHook(  
    Session As SmApplic.SmSession,  
    ClassId As Integer,  
    Operation As Object,  
    Stage As HookStageEnum,  
    Str As String,  
    FirstPar As SmRecList.SmRecordList,  
    SecondPar As SmRecList.SmRecordList,  
    ThirdPar As SmRecList.SmRecordList) As ErrorCodeEnum
```

6. Add the following sample code in the function body. This example code allows only the creator of a folder to update it:

```
Public Function BeforeUpdateHook(  
    Session As SmApplic.SmSession,  
    ClassId As Integer,  
    Operation As Object,  
    Stage As HookStageEnum,  
    Str As String,  
    FirstPar As SmRecList.SmRecordList,  
    SecondPar As SmRecList.SmRecordList,
```

```

ThirdPar As SmRecList.SmRecordList) As ErrorCodeEnum
    Dim Folder As SmApplic.ISmObject

    On Local Error GoTo HandleError
    Set Folder = Session.ObjectStore.ObjectFromData(
    FirstPar.GetRecord(0), True)
    If Folder.Data.ValueAsInteger("USER_OBJECT_ID") =
    Session.UserMetaInfo.UserId Then
        BeforeUpdateHook = ecNone
    Else
        BeforeUpdateHook = ecNotOwned
    End If
    Exit Function
HandleError:
    BeforeUpdateHook = ecGen
End Function

```

7. Compile the code and register the DLL

Note: The physical location of a COM DLL is not important as long as you have registered it.

8. Edit the System Configuration and add the following Key value, which defines the CLSID of the SmartServerHookStd.SmDispatch module provided with SMARTEAM and always has the same value:

```
ServerHooks.CLSID={82F7EBD2-61D9-4CEB-8FD8-535EF32DEB2C}
```

9. Edit the System Configuration and add the following Key value, which defines the ProgID or CLSID of the custom ActiveX DLL and varies with the application:

```
ServerHooks.Init=Project1.Sample
```

10. In a Server-side SmarTeam -based application that was created by the user, set `SmEngine.ServerMode = True` (In Server-side SMARTEAM applications such as SMARTEAM – Web Editor, this parameter is already set.) Make sure you use `SmFreeThreadedEngine` and not `SmEngine`.

The `BeforeUpdateHook` code will be automatically invoked whenever you attempt to Update a Persistent Object in the database.

3. Standard SMARTTEAM Hook Interface

This section describes the standard SmarTeam hook interface and includes the following topics:

- Naming hook functions
- Running a hook function that was not specified in SmarTeam
- Hook function parameters
- Packaging the hook functions into an interface object
- Integrating the COM component into SmarTeam

Naming Hook Functions

In the standard interface, the system recognizes the name of a hook function in two different forms:

- The name that was specified for the function in SmarTeam
- Standard “event-based” name, which is searched for and recognized by the system when the event occurs.

Using the name specified in SmarTeam

You can specify the name of a hook function in SMARTTEAM in the following ways:

- For a generic hook – in the Script Maintenance utility
- For a library-specific hook – using the appropriate designer, such as the Flow Chart Designer.

Using an event-based name

If you did not specify a function name for a hook in SmarTeam, the system looks for a function name that conforms to an “event-based” naming convention.

There are two event-based naming conventions corresponding to the two types of hooks:

- Generic hooks
- Library-specific hooks

Event-Based Naming for Generic Hooks

For generic hooks, you combine the stage and the SmarTeam operation into a single event-based name in the following format:

Before_[OperationName]

After_[OperationName]

Instead_[OperationName]

where OperationName is the *internal name* for the operation as shown in the right column of Table 2).

Examples of event_based names are: Before_ADD, Instead_LifeCycle1.

Note: No spaces are allowed in the internal name. Replace any spaces with underscores.

Table 2 Internal Name for SmarTeam Operations

Operation in Script Maintenance	Operation Name (Internal Name)
Operations on Objects	
Add	ADD
Add As Copy (Optional)	AddAsCopy
Update	UPDATE
Delete	DELETE
Scripts for Simple Life-Cycle Operations	
Check Out	CheckOut
Check In	CheckIn
Release	Approve
New Release	NewRelease
Obsolete	Freeze
Undo Check Out	UndoCheckOut
Scripts for Advanced Life-Cycle Operations	
Life-Cycle Stage 1	LifeCycle1
Life-Cycle Stage 2	LifeCycle2
Scripts for File Operations	
Edit	Edit
View	View
RedLine	RedLine
Print	Print
Copy File	CopyFile
Scripts for Authorization Operations	
On RetrieveObjects	RetrieveObjects
On Authenticate User	OnAuthenticateUser

Event-Based Naming for Library-Specific Hooks

For Smart Flow event-driven scripts, the system recognizes the following standard names:

OnSendBefore
OnSendAfter
OnReceive

For a Smart Flow task, unlike a Smart Flow event, the task procedure name must be defined in the Flow Chart Designer for the system to execute it. If no such name is found, the system does not look for a standard name. Therefore, for a Smart Flow task function, you can only use the function name that was specified in the Flow Chart Designer.

Running a Function not Specified in SmarTeam

Event-based naming lets you run a hook function that was not specified in SmarTeam. As mentioned above, you do that by including a function in the COM object with a standard event-based name that denotes the hook event for which you want the hook function to run. When any hook event occurs, SmarTeam looks for a hook function with the standard event-based name for that hook. If you have provided such a function, SmarTeam runs it.

If you want to disable such a hook function so that it doesn't run when the event occurs, you need to remove it from the component you have added (or rename it) and then re-compile and re-register the component.

Note: If you have specified both a name for the hook function in SmarTeam and also an event-based name, the hook function with the name specified in SmarTeam is run instead of the function with the event-based name.

Hook Function Parameters

In the standard interface, the hook function parameters depend on the type of SmarTeam hook: Generic hooks or library-specific hooks.

Generic Hooks

For the generic hooks, you use the following COM-based parameters for a hook function. These parameters are used in Server Mode instead of the parameters you use in the generic procedural-based hooks that are described in the Script Hooks document.

Using **SmarTeam** Hooks in Server Mode

Table 3 Hook Function Parameters

Parameter	Type	Description
Session	SmApplic.SmSession	Current SMARTEAM session
ClassId	Integer	Class ID under which hook is defined, for example Folders.
Operation	SmOperation	Operation associated with hook, for example, ADD, UPDATE
Stage	HookStageEnum	Stage of hook: <ul style="list-style-type: none">▪ hsAfter▪ hsBefore▪ hsInstead
Str	String	Name of operation
FirstPar	SmRecList.SmRecordList	Input (see Script Hooks)
SecondPar	SmRecList.SmRecordList	Input/Output (see Script Hooks)
ThirdPar	SmRecList.SmRecordList	Output (see Script Hooks)
Return Value	ErrorCodeEnum	Return value

Example

This example shows how to use a generic hook function that was defined in Script Maintenance. It assumes that the function `UserIsCreator` was defined in Script Maintenance for the Before Update SmarTeam hook.

`UserIsCreator` determines if the current user is the creator of the folder being updated. It can be used to allow the folder to be updated only by its creator.

```
Public Function UserIsCreator(  
    Session As SmApplic.SmSession,  
    ClassId As Integer,  
    Operation As Object,  
    Stage As HookStageEnum,  
    Str As String,  
    FirstPar As SmRecList.SmRecordList,  
    SecondPar As SmRecList.SmRecordList,  
    ThirdPar As SmRecList.SmRecordList) As ErrorCodeEnum  
    Dim Folder As SmApplic.ISmObject  
  
    On Local Error GoTo HandleError  
    Set Folder = Session.ObjectStore.ObjectFromData(  
        FirstPar.GetRecord(0), True)  
    If Folder.Data.ValueAsInteger("USER_OBJECT_ID") =  
        Session.UserMetaInfo.UserId Then  
        UserIsCreator = ecNone  
    Else  
        UserIsCreator = ecNotOwned  
    End If  
    Exit Function  
HandleError:  
    UserIsCreator = ecGen  
End Function
```

Example

This example illustrates how you can use the naming convention to cause a script to execute even when the user did not define it in Script Maintenance. This example assumes that the function `After_Approve` was not defined in Script Maintenance. It is named with a standard event-based name and executes each time the hook event `After_Approve` occurs.

```
' This function executes in Server Mode on the After_Approve hook
' It prints the list of objects and attributes to the log file
Public Function After_Approve(
    Session As SmApplic.SmSession,
    ClassId As Integer,
    Operation As Object,
    Stage As HookStageEnum,
    Str As String,
    FirstPar As SmRecList.SmRecordList,
    SecondPar As SmRecList.SmRecordList,
    ThirdPar As SmRecList.SmRecordList) As ErrorCodeEnum
    FirstPar.PrintToFile "Approved objects list", "C:\Approved.log"
    After_Approve = ecNone
End Function
```

Library-Specific Hooks

For the library-specific hooks you use exactly the same parameters that you use for each type of script hook, for example, in Smart Flow (see the document *COM API Programmer's Guide* for the parameters of the SmarTeam Flow hooks).

Example

The example illustrates how you can use the naming convention to cause a library-specific hook script to execute even when the user did not define it in the Flow Chart Designer. This example assumes that the function OnSendAfter was not defined in the Flow Chart Designer. It is named using the standard event-based naming described above and executes each time the hook event occurs. The function is written with the same parameter list used for the OnSendAfter script hook functions.

```
' This function notifies users that a new process is waiting
' in the SmartBox
Public Function OnSendAfter(
FlowSession As SmartFlow.SmFlowSession,
FlowProcess As SmartFlow.SmFlowProcess,
Node As SmartFlow.SmNode,
Response As SmartFlow.SmResponse) As ErrorCodeEnum
    Dim Nodes As SmartFlow.SmNodes
    Dim I As Integer
    Dim J As Integer
    Dim Mail As SmartMessages.SmExternalMessage
    Dim MessageStore As SmartMessages.SmMessageStore
    Set MessageStore = FlowProcess.FlowStore.Session.GetService(
"SmartMessages.SmMessageStore")
    Set Mail = MessageStore.NewExternalMessage

    Set Nodes = Node.GetOutgoingNodes(Response)
    For I = 0 To Nodes.Count - 1
        For J = 0 To Nodes(I).Users.Count - 1
            Mail.AddRecipient Nodes(I).Users.GetUsers(J).
                Data.ValueAsString("USER_EMAIL"), mrTo
        Next
    Next
    Mail.Send False
    OnSendAfter = ecNone
End Function
```

Packaging the Hook Functions into an Interface Object

The hook functions need to be packaged into an Automation-compatible COM object. This COM object is required in addition to the SmartServerHookStd.SmDispatch module provided by SmarTeam, which controls the execution of the hook functions.

The COM object can be implemented by one of the following methods

You can continue to use a scripting language like VBScript to produce a Windows Script Component

You can implement hook functions using a COM-compatible development tool such as Visual Basic or Visual C++ and then compile them to produce a COM component.

For more information about using VBScript to write Windows Script Components, see <http://msdn.microsoft.com/scripting>.

Integrating the COM Component

To integrate the COM component you have produced into SmarTeam:

1. Register the COM component you have produced in the Windows Registry
2. Get the CLSID or ProgID for the COM component from the Windows Registry.
3. Edit System Configuration and add the following Key value, which defines the CLSID of the SmartServerHookStd.SmDispatch module provided with SMARTEAM and always has the same value:

```
ServerHooks.CLSID={82F7EBD2-61D9-4CEB-8FD8-535EF32DEB2C}
```

4. Edit System Configuration and add the following section, which defines the ProgID or CLSID of the of the COM component you have produced:

```
ServerHooks.Init=ProgID or CLSID
```

Example

This example assumes that the ProgID of the COM object you created is "SampleServerHook.WSC":

In System Configuration, the Key values are:

```
ServerHooks.CLSID={82F7EBD2-61D9-4CEB-8FD8-535EF32DEB2C}  
ServerHooks.Init=SampleServerHook.WSC
```


4. Low-Level SmarTeam Hook Interface

In addition to the standard SmarTeam hook interface described above, SmarTeam provides a low-level SmarTeam hook interface in Server Mode.

The low-level interface has the following advantages:

- Flexible hook function naming

- Flexible control – enable and disable hook function execution, determine execution rules of event-based hook functions and SmarTeam hook functions

- Higher performance

Note: Since the standard interface meets most requirements, the low-level interface is usually not required. Using the low-level interface requires experience in creating COM objects.

This section describes the low-level SmarTeam hook interface and includes the following topics:

- Implementing the interface ISmServerHook

- Naming SmarTeam hook functions

- Running a hook function that was not specified in SmarTeam

- Hook function parameters

- Packaging the functions into an interface object

- Integrating the COM component into SmarTeam

Implementing the Interface Class ISmServerHook

The interface ISmServerHook is located in the library SmartServerHook.tlb.

You need to implement the five control methods:

- Init – Executes optional user functions prior to executing event hook functions.
- Type1HookExists – Determines if hook functions are available for execution for this Type1 hook
- Type1Execute – Executes the Type 1 hook functions
- Type2HookExists – Determines if hook functions are available for execution for this Type2 hook.

- Type2Execute – Executes the Type 2 hook functions

Hook Types

The above methods are divided into Type1 methods and Type2 methods corresponding to the two types of hooks:

- Type1 hooks – Generic hooks. These can be defined in Script Maintenance.
- Type2 hooks – Library-specific hooks. These can be defined, for example, in the Flow Chart Designer

The system runs the Type1Execute and Type1HookExists methods when Type1 hook events occur. The system runs the Type2Execute and Type2HookExists methods when Type2 hook events occur.

Function Parameters

These control functions parameters are described in the following table:

Table 4 ISmServerHook Functions

Function	Parameter	Type	Description
Init	None		
Type1HookExists	Session	SmSession	Current session
	ClassId	Integer	Class ID for class under which hook is defined
	Operation	ISmOperation	Operation associated with hook, for example, ADD, UPDATE
	Stage	HookStageEnum	Stage of hook: <ul style="list-style-type: none"> ▪ hsAfter ▪ hsBefore ▪ hsInstead
	FunctionName	String	Name of hook function in Script Maintenance if it exists there.
	Return Value	Boolean	If true, SmarTeam runs Type1Execute, Else it doesn't.

Using **SmarTeam** Hooks in Server Mode

Type1Execute	Session	SmSession	Current session
	ClassId	Integer	Class ID for class under which hook is defined
	Operation	ISmOperation	Operation associated with hook, for example, ADD, UPDATE
	Stage	HookStageEnum	Stage of hook: <ul style="list-style-type: none">▪ hsAfter▪ hsBefore▪ hsInstead
	FunctionName	String	Name of hook function in Script Maintenance if it exists there.
	Str	String	Name of operation
	FirstPar	SmRecordList	Input (see Script Hooks)
	SecondPar	SmRecordList	Input, Output (see Script Hooks)
	ThirdPar	SmRecordList	Output (see Script Hooks)
	Return Value	ErrorCodeEnum	Error code
Type2HookExists	Session	SmSession	Current SmarTeam session
	HookName	String	Hook name in Flow Chart Designer or other Designer: <ul style="list-style-type: none">▪ OnSendBefore▪ OnSendAfter▪ OnReceive
	FunctionName	String	Hook function name in Flow Chart Designer or other Designer
	Return Value	Boolean	If true, SmarTeam runs Type2Execute, Else it doesn't.
Type2Execute	Session	SmSession	Current SmarTeam session
	HookName	String	Hook name in Flow Chart Designer or other Designer: <ul style="list-style-type: none">▪ OnSendBefore▪ OnSendAfter▪ OnReceive

FunctionName	String	Hook function name in Flow Chart Designer or other Designer
Parameters	SmRecord	Parameters for this Smart Flow hook function, packaged into a record (see COM API Programmer's Guide)
Return Value	ErrorCodeEnum	Error code

The following sections describe the use of these methods.

Init Method

The Init method is executed prior to the hook functions. Put into the Init method all user functions that need to execute prior to the hook functions.

This example illustrates the use of the Init method

```
Sub ISmServerHook_Init(ByVal Param As String)
    MsgBox "Initialize server hooks"
End Sub
```

When ISmServerHook_Init is called by SMARTEAM, it passes the parameter Param to the subroutine. The value of Param is the value of the System Configuration Key `ServerHooks.Init`.

For example, if the Key value is:

```
ServerHooks.Init=Project1.Sample
then Param = Project1.Sample.
```

Type1HookExists Method

The purpose of the Type1Exists method is to determine whether a hook function is available to execute for the current event hook. If one is found, the system runs the Type1Execute Method to execute the function.

When the Method is Called

The system calls this method every time any one of the Type1 hook events occurs. The method parameter `FunctionName` passes any function name that has been associated with the current hook event in Script Maintenance. For example, if the hook function `UserIsCreator` was defined for the hook event `After_Approve` in Script Maintenance, the parameter `FunctionName` passes `UserIsCreator` every time the `After_Approve` event occurs.

Hook Function Execution Logic

The function compares `FunctionName` with a hook function name that is known by the script writer to have been defined in Script Maintenance. If the comparison succeeds, the method returns true and the system runs the `Type1Execute` method, which executes the function.

If the known hook function name is not identified, the method identifies the current hook by operation name and event stage. In case an event-based hook function is available for execution, the script writer has the method set the return value true. Then, the event-based hook function is run by the `Type1Execute` method, as shown below.

By the default program logic, the method searches first for an actual hook function name and, if not found, it then asks if the current hook is a desired hook.

However, you can control hook function execution logic by reprogramming this method. For example, you can have the method always return false and the hook functions will not execute even if they exist. You do not need to remove the hook functions themselves from the module.

Example

This example illustrates the use of the `Type1Exists` method.

```
Function ISmServerHook_Type1HookExists(  
    ByVal Session As SmSession,  
    ByVal ClassId As Integer,  
    ByVal Operation As ISmOperation,  
    ByVal Stage As HookStageEnum,  
    ByVal FunctionName As String) As Boolean  
    'Check if function name exists in Script Maintenance  
    If StrComp(FunctionName, "UserIsCreator") = 0 Then  
        ISmServerHook_Type1HookExists = True  
    Else  
        'Check if calling hook is After_Approve
```

```

        If StrComp(Operation.Name, "APPROVE") = 0 And Stage = hsAfter Then
            ISmServerHook_Type1HookExists = True
        End If
    End If
End Function

```

Type1Execute Method

This method is called by the system when the method Type1Exists returns true. It passes the Type1 hook parameters and executes the hook functions.

Hook Function Execution Logic

By the default program logic, the method searches first for an actual hook function name and, if not found, then for an event-based name.

However, you can change the hook function execution logic by implementing this method differently. You can even have both hook functions execute. If you do change the program logic, you need to coordinate with the program logic of the Type1Exists method.

Hook Function Naming

In this low-level interface, the hook function is called explicitly by the method in the framework of the program logic. Hence, the name of the hook function that executes doesn't need to be the same as the name of the hook function in Script Maintenance, as it does in the standard interface.

The same is true for event-based naming. The event-based hook function is called explicitly by the method according to the program logic.

Example

This example illustrates the use of the Type1Execute method

```

Function ISmServerHook_Type1Execute(
    ByVal Session As SmSession,
    ByVal ClassId As Integer,
    ByVal Operation As ISmOperation,
    ByVal Stage As HookStageEnum,
    ByVal FunctionName As String,
    ByVal Str As String,
    ByVal FirstPar As SmRecordList,
    ByVal SecondPar As SmRecordList,

```

Using **SmarTeam** Hooks in Server Mode

```
ByVal ThirdPar As SmRecordList) As ErrorCodeEnum
    Dim Res As ErrorCodeEnum
    Res = ecNone

    If StrComp(FunctionName, "UserIsCreator") = 0 Then
        ' Execute function by function name
        Res = UserIsCreator(Session, ClassId, Operation, Stage, Str,
            FirstPar, SecondPar, ThirdPar)
    Else
        If StrComp(Operation.Name, "APPROVE") = 0 And Stage = hsAfter Then
            'Execute function by hook name
            Res = After_Approve(Session, ClassId, Operation, Stage, Str,
                FirstPar, SecondPar, ThirdPar)
        End If
    End If
    ISmServerHook_Type1Execute = Res
End Function
```

Type2HookExists Method

The purpose of the Type2Exists method is to determine whether a hook function is available to execute for the current event hook. If one is found, the system runs the Type2Execute Method to execute the function.

See the introductory comments to the Type1HookExists method.

Example

```
Function ISmServerHook_Type2HookExists(  
    ByVal Session As SmSession,  
    ByVal HookName As String,  
    ByVal FunctionName As String) As Boolean  
    If StrComp(FunctionName, "BeforeSendFunction") = 0 Or  
       StrComp(FunctionName, "ScriptTasks") = 0 Then  
        ISmServerHook_Type2HookExists = True  
    Else  
        If StrComp(HookName, "OnSendAfter") = 0 Then  
            ISmServerHook_Type2HookExists = True  
        End If  
    End If  
End Function
```

Type2Execute Method

This method is called by the system when the method Type2Exists returns true. It passes the Type2 hook parameters and executes the hook functions.

For more information, see the introductory section for the Type1Execute method.

See the notes at the end of the example for specific information about passing parameters for this method.

Example

This example shows how to execute the Type2 hook functions.

```
Function ISmServerHook_Type2Execute(  
    ByVal Session As SmSession,  
    ByVal HookName As String,  
    ByVal FunctionName As String,  
    ByVal Parameters As SmRecord) As ErrorCodeEnum
```

Using **SmartTeam** Hooks in Server Mode

```
Dim Res As ErrorCodeEnum
Dim ActiveProcess As SmartFlow.SmActiveProcess
Dim Response As SmartFlow.SmResponse
Dim Task As Object
Dim MultiObjects As Object
Dim FlowSession As SmartFlow.SmFlowSession
Dim FlowProcess As SmartFlow.SmFlowProcess
Dim Node As SmartFlow.SmNode
'Dim Response As SmartFlow.SmResponse

Res = ecNone
'Check if function name exists in Flow Chart Designer
If StrComp(FunctionName, "BeforeSendFunction") = 0 Then
    ' Extract hook function parameters from record Parameters
    Set ActiveProcess = Parameters.ValueAsObjectByIndex(1)
    Set Response = Parameters.ValueAsObjectByIndex(2)
    ' Execute hook function with parameters
    Res = BeforeSendFunction(ActiveProcess, Response)
    Res = ecNone
Else
    'Check if current hook is OnSendAfter
    If StrComp(HookName, "OnSendAfter") = 0 Then
        ' Extract hook function parameters from record Parameters
        Set FlowSession = Parameters.ValueAsObjectByIndex(1)
        Set FlowProcess = Parameters.ValueAsObjectByIndex(2)
        Set Node = Parameters.ValueAsObjectByIndex(3)
        Set Response = Parameters.ValueAsObjectByIndex(4)
        'Execute the OnSendAfter hook function with parameters
        Res = OnSendAfterFunction(FlowSession, FlowProcess, Node,
            Response)
    Else
        'Check if task name exists in Flow Chart Designer
        If StrComp(FunctionName, "ScriptTasks") = 0 Then
            ' Execute Sub by sub name
            ' Extract task parameters from record Parameters
            Set ActiveProcess = Parameters.ValueAsObjectByIndex(0)
            Set Task = Parameters.ValueAsObjectByIndex(1)
            Set MultiObjects = Parameters.ValueAsObjectByIndex(2)
            'Execute the task hook procedure with parameters
```

```

        ScriptTasks ActiveProcess, Task, MultiObjects
        Res = ecNone
    End If
End If
End If
ISmServerHook_Type2Execute = Res
End Function

```

Notes

1. The hook function parameters need to be extracted from the SmRecord “parameters” as shown. The type of hook dictates the parameters that are passed (refer to the COM API Programmer’s Guide for the actual parameters required for each event hook..) Therefore, if you execute a hook function by the FunctionName parameter as in the first if-clause of the above example, you need to be aware of the type of hook with which the function is associated in Flow Chart Designer or other Designer in order to extract the correct parameters.
2. The parameter indexing in SmRecord is different for events than for tasks. For events, which always correspond to hook functions, the parameters start with index 1 (Parameters.ValueAsObjectByIndex(1)); for tasks, which always correspond to hook procedures, the parameters start with index 0 (Parameters.ValueAsObjectByIndex(0)).

Running a Function not specified in SmarTeam

Event-based naming lets you run a hook function that was not specified in either the SmarTeam as in the standard interface. See the description in the standard interface.

If you want to disable such a hook function so that it doesn’t run when the event occurs, you don’t need to remove it from the component as in the standard interface. You can just change the program execution logic in the control methods so that the function doesn’t run. You still need to re-compile and re-register the component.

Naming Hook Functions

In the low level interface, you have complete flexibility for naming the hook functions themselves.

The two types of function names described in the standard interface:

Naming the function according to the name specified in SmarTeam

Naming the function according to the SmarTeam hook event for which it runs

are used in the low-level interface in the Type1Exists and Type2Exists control methods for the logical comparisons and in Type1Execute and Type2Execute control methods for the execution logic. See the standard interface for a complete description of these function names.

Hook Function Parameters

In the low-level interface, the parameters you use for the SmarTeam hook functions depend on the type of SmarTeam hook: generic hooks or library-specific hooks, as described in the standard interface.

Packaging the Functions into an Interface Object

Normally, the SmarTeam hook functions are packaged with the implementation of the IsmServerHook class into an Automation-compatible COM Object. However, you can package them separately if you provide an appropriate execution mechanism.

Integrating the COM Component

To integrate the COM component you have produced into SmarTeam:

1. Register the COM component you have produced in the Windows Registry
2. Get the CLSID for the COM component from the Windows Registry.
3. In the value of the System Configuration Key `ServerHooks.CLSID`, specify the CLSID of the COM component you have produced.

Example

```
ServerHooks.CLSID={82F7EBD2-61D9-4CEB-8FD8-535EF32DEB2D}
```