



ENOVIA SmarTeam

SmarTeam – Web Editor Customization Guide

© Dassault Systèmes, 2009. All rights reserved.

CATIA, ENOVIA, DELMIA, SMARTEAM and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries in the US and/or other countries and are used under license by IBM.

Viewing Technology Copyright © 1993-2006 Cimmetry Systems, Inc. All Rights Reserved.

AutoVue and Cimmetry are trademarks or registered trademarks of Cimmetry Systems, Inc. © Copyrights of Oracle Corporation. All rights reserved. © Copyright IBM Corporation 1994, 2006. All rights reserved. Other portions of the software and all other trademarks or service marks referred to herein belong to their respective owners.

Any use of the Software Products contained in this media, is subject to the LICENSED SOFTWARE TERMS AND CONDITIONS accompanying this media and/or the previous release. The Software Products are protected by international copyright laws and international treaties. Unauthorized use, reproduction and/or distribution of any of the Software Products, 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.

Part No: WED-C1-200009

Table of Contents

| | |
|---|-----------|
| Chapter 1: Introduction | 1 |
| General Guidelines | 1 |
| Customizable Features | 1 |
| Software Location | 2 |
| Internet Site | 2 |
| Chapter 2: Preferences | 3 |
| Navigation Pane | 3 |
| Default Settings | 3 |
| Public Views | 3 |
| Private Views | 3 |
| Color Settings (Link Type) | 3 |
| Related Objects Pane | 3 |
| Public Settings | 4 |
| Private Settings | 4 |
| Chapter 3: User-defined Tools | 5 |
| Adding a User-Defined Tool | 5 |
| Define userDefinedTools key in SmarTeam Configuration Settings | 6 |
| Example of Building a User-defined Page | 6 |
| Getting SmarTeam User-defined Page Parameters | 9 |
| Getting SmarTeam User-defined Page Parameters (before V5R16 SP4) | 9 |
| User-defined Tools Appearance | 10 |
| Chapter 4: Workflow Events | 13 |
| Creating a User-defined Page for a Workflow Operation | 14 |
| Using the Selected Objects in a User-defined Page | 17 |
| Chapter 5: Application Toolbar | 19 |
| Manually Defining an applicationBar key in the SmarTeam System Configuration: | 24 |
| Chapter 6: Adding Your Own Cockpit | 26 |
| Chapter 7: Look and Feel | 28 |
| Adding a Company Logo | 29 |
| Replacing Login with SmarTeam – Web Editor Login Page | 30 |

| | |
|---|-----------|
| Chapter 8: Profile Cards | 32 |
| Adding Client-side Scripts | 32 |
| Example: Adding a New Button to a Profile Card | 33 |
| Chapter 9: Adding Server-Side Scripts | 38 |
| Prerequisites | 38 |
| When are Server-Side Scripts Used? | 38 |
| Script Execution Logic | 39 |
| Server-Side Script Parameters | 40 |
| Generic Hooks | 41 |
| Library-Specific Hooks | 41 |
| Return a Value On a Script Error | 43 |
| With Reference to SmartWebAppUtils | 43 |
| Without Reference to SmartWebAppUtils | 44 |
| Server-Side Script Naming Convention | 44 |
| Packaging Server-Side Scripts into an Interface Object | 45 |
| SmarTeam Hook Interfaces | 45 |
| Standard Server-Side Script (By Script Maintenance Name) | 47 |
| Standard Server-Side Script (By the Event Name) | 49 |
| Low-Level Server-Side Scripts | 50 |
| Low-Level Server-Side Script using a TypeHook1 Interface | 51 |
| Low-Level Server-Side Script using a TypeHook2 Interface | 55 |
| Chapter 10: Visual Components | 58 |
| Profile Card | 58 |
| Grid | 58 |
| Tree | 58 |
| SmartBox | 58 |
| Chapter 11: Customize External Viewers | 59 |
| Appendix A: Client-Side Scripts Examples | 61 |
| Example 1: Use of ListItemsCollection for DropDownList control | 61 |
| Example 2: Cycles through Profile Card Controls/Shows the Details | 62 |
| Example 3: Shows Profile Card Mode | 62 |
| Example 4: Hides Profile Card Controls | 63 |
| Example 5: Shows Controls that were Hidden | 63 |
| Example 6: Retrieves Current SmarTeam Object Data | 64 |
| Example 7: Changes Value of Profile Card Controls | 64 |
| Example 8: Changes Text in Profile Card Controls | 64 |
| Example 9: Changes Background Color of Profile Card Controls | 65 |
| Example 10: Disables Profile Card Controls | 65 |
| Example 11: Enables Profile Card Controls | 66 |
| Example 12: Changes the Fonts of Profile Card Controls | 66 |
| Example 13: Called Before Loading Profile Card | 66 |

Chapter 1: Introduction

SmarTeam – Web Editor enables you to customize the application. This function is usually performed by the system administrator of the organization.

General Guidelines

In order to start customizing SmarTeam – Web Editor, the following issues need to be considered:

- After making changes to the SmarTeam system configuration settings, you need to restart the SmarTeam System Configuration Service and run **iisreset**.
- You can use the different levels of configuration to define settings, or perform customizations for different groups in the company. However, remember that the system configuration uses hierarchical inheritance, i.e., that the lower levels override the upper level definitions. For example, if a setting is defined at the system\<DB> level and the same value is defined at the user\<username> level, the user level definition prevails
- Before you start a new customization verify that you have backed up your DB and all Configuration settings
- For the Setup Development Environment you must copy SmarTeam .Net Interops from the GAC of Microsoft® Windows to any developer folder. You can use a batch file:
If it does not exist, type:

```
"C:\Program Files\SmarTeam\Bin\Assembly" md "C:\Program  
Files\SmarTeam\Bin\Assembly" or /R %WINDIR%\assembly\GAC %%f in  
(SmarTeam.Std.*.dll) do copy %%f "C:\Program  
Files\SmarTeam\Bin\Assembly"
```

- The **JavaAppletLoadAttempts** key in the web.config file specifies how many times the application attempts to load Java Applets. If the number of attempts is insufficient, you are prompted with the message: "Java Applet failed to load. Please contact your System Administrator to reconfigure the number of attempts". The Administrator must increase the value in the web.config file. The key is located in the appSettings section of the web.config file in the SmarTeam - Web Editor or any Visual Components application. For example: **<add key = "JavaAppletLoadAttempts" value = 10"/>**

For more information, refer to the SmarTeam System Configuration documentation that can be found on the Documentation CD.

Customizable Features

The following SmarTeam – Web Editor features can be customized:

- **Preferences:** Enables you to define views and settings for :
 - Display Attributes
 - Language Settings
 - Tree Filters
 - Navigation Pane
 - Related Objects Pane
- **User-defined Tools:** Enables you to add items to the Action menus by adding user-defined pages that can be used to perform custom operations on the objects selected by the user
- **Workflow Events:** Enables you to add a user defined page (similar to the user defined tools) that executes during a Workflow operation.
- **Application Bar:** Enables you to customize the Application toolbar, by adding your own entries to the Application toolbar menus
- **Cockpit:** Enables you to add your own cockpits, so they can be seen by different users in the organization and show them specific data.
- **Look and Feel:** Enables you to customize the application's look and feel, by making changes in the Cascading Style Sheets (CSS) files that define the style of the application.
- **Profile Cards:** Enables you to add a script of programming commands to be executed before a profile card is opened.
- **Server-side Scripts:** Enables you to add Server-Side Scripts.
- **Visual Components:** Enables you to customize the way SmarTeam components are displayed.

Software Location

The customization procedures described in this document is for the SmarTeam – Web Editor software, which is available on the [SmarTeam website](#).

Internet Site

You are highly recommended to frequently visit our website for the latest updates and plug-in products, including the latest Service Packs, Program Directory (Release Notes), Hotfixes and technical support at <http://www.3ds.com/support/>.

In addition, you will also be able to view any installation known issues.

Chapter 2: Preferences

This chapter describes SmarTeam – Web Editor, Preferences features.

Navigation Pane

In the Navigation pane, you can customize which view to use to display objects. The views available are defined in the View Manager in the Navigation Pane settings.

Default Settings

Enables you to define a default view for each super-class whose object is presented as the root object in the navigation pane. For details, refer to SmarTeam – Web Editor Online Help, Defining Default View Settings topic.

Public Views

Consists of views that are presented to all users in an organization. Public views are defined and can be modified by administrators only. For details, refer to SmarTeam – Web Editor Online Help, View Manager topic.

Private Views

Consists of views defined by a user/administrator for himself only. When other users log in they do not see these views. For details, refer to SmarTeam – Web Editor Online Help, View Manager topic.

Color Settings (Link Type)

Enables you to configure the colors of the presented links objects in the Navigation Pane. You can configure different colors for the directional link type groups. For example, all Normal Hierarchical links can be configured a specific color. With administrator permissions, you can set any color scheme as Public for the entire database. For details, refer to SmarTeam – Web Editor Online Help, Using Links Color Settings topic.

Related Objects Pane

In the Related Objects pane, you can configure a list of objects to appear in this pane that are linked to the source object.

Public Settings

Consists of related objects settings that are presented to all users in an organization. Public settings are defined and can be modified by administrators only. For details, refer to SmarTeam – Web Editor Online Help, Configuring the Related Objects Pane topic

Private Settings

Consists of settings defined by a user/administrator for himself only. When other users log in they do not see these settings. For details, refer to SmarTeam – Web Editor Online Help, Configuring the Related Objects Pane topic

Chapter 3: User-defined Tools

SmarTeam – Web Editor enables you to add items to the User Defined Tools menu by adding user-defined pages that can be used to perform custom operations on the objects selected by the user.

Note: SmarTeam – Web Editor (web-based interface) does not support the use of commands outside of the application, such as Refresh (F5), Back or Forward buttons of the Web browser, etc.

User-defined tools can be accessed from the main page menu at the top of the screen. The User Defined Tools can be presented in two different modes according to the configuration:

- A single entry under the User Defined Tools menu that leads to the User Defined Tools screen
- All User Defined Tools are presented directly under the User Defined Tools menu

The list of User Defined Tools can contain the following User-defined tools:

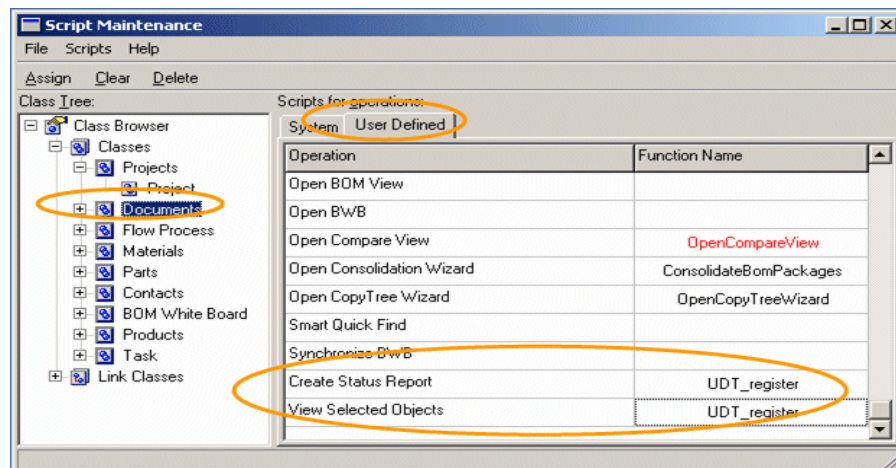
- Those defined for the common parent class of the selected objects in SmarTeam Script Maintenance.
- Those with an entry with the same name in the System Configuration file `smarteam.std.webEditor.config`.
- Those whose operation is enabled on selected objects.

Adding a User-Defined Tool

To add a User-defined tool:

- 1 Define the tool using Script Maintenance and attach it to any script for the appropriate class or classes. The User-defined tools in SmarTeam – Web Editor ignores this script.

Note: To add empty rows to the Script Maintenance window, User Defined tab pane, click in a row and press the down arrow key.



Define userDefinedTools key in SmarTeam Configuration Settings

To define the userDefinedTools key:

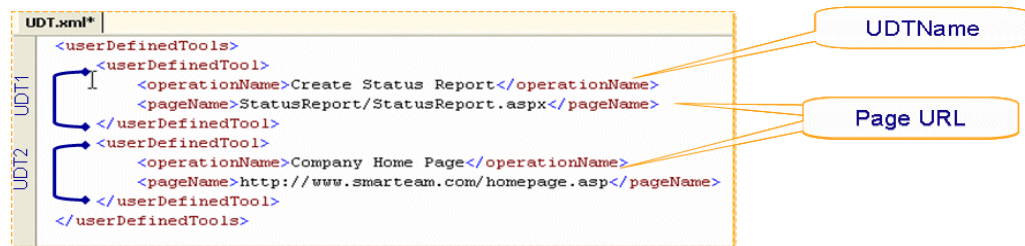
- 1 Define a hook for the new User-defined tool in Script Maintenance.
- 2 Build the tool as an ASPX.NET page.
- 3 Define the userDefinedTools key in SmarTeam Configuration settings:
 - The key contains the schema:



- Define the userDefinedTools key manually:

IMPORTANT! Before making changes to the XML files, it is highly recommended to backup the [SmarTeam home directory]\ConfigurationSettings\Data folder.

- Locate the userDefinedTools key in <SmarTeam>/ConfigurationSettings/Data folder
- Using a text editor, open **smarteam.std.webEditor.config.xml**
- Add a new entry in the userDefinedTools key

**Notes:**

- a Copy the following to the aspx page that was added to the project content of the file [path to SmarTeam – Web Editor installation]:
`<smarteam dir> \WebEditor\Web\Views\UserDefinedTools\UserDefinedToolDemo1.aspx`
- b Backup the configuration settings located in the <SmarTeam>\ConfigurationSettings\Data folder
- c Copy the customer .DLL (code behind) to the <homeDir>\WebEditor\Web\bin directory
- d The root folder for UDT is <SmarTeam>\WebEditor\Web\Views\UserDefinedTools.

Example of Building a User-defined Page

If you do not use code behind in a .ASPX page:

- 1 Register the **SmarTeam.Std.Applications.WebEditor** assembly in your tool using the page directive:

```
<% @ Assembly Name="SmarTeam.Std.Applications.WebEditor" %>
```

- 2 Import the **SmarTeam.Std.Applications.WebEditor.Views.UserDefinedTools** namespace in your tool using the page directive:

```
<% @ Import
```

```
Namespace="SmarTeam.Std.Applications.WebEditor.Views.UserDefinedTools" %>
```

If you use code behind in a .ASPX page:

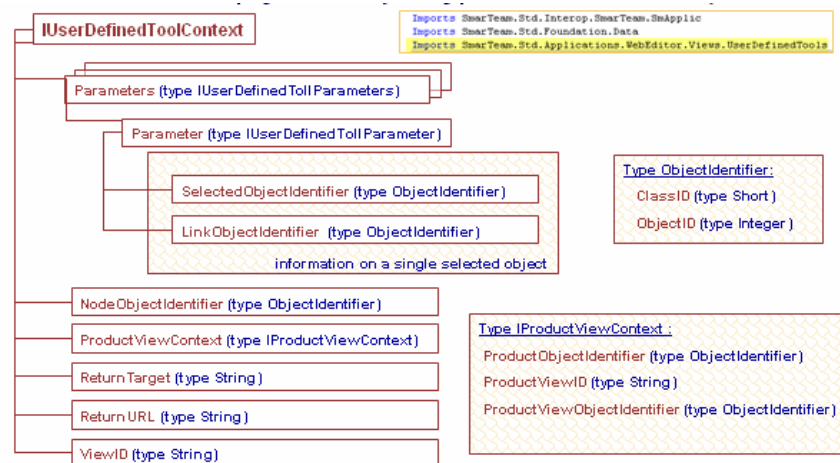
- Add a reference to **SmarTeam.Std.Applications.WebEditor.dll** for using the interface **IUserDefinedToolContext**
(librarySmarTeam.Std.Applications.WebEditor.Views.UserDefinedTools)

Follow the example on the sample page below to understand how to retrieve **UserDefinedToolParameters** and extract information on each selected object.

```
void Page_Load(Object sender, EventArgs e)
{
    //get session key from QueryString
    string sessionKey =
    Request.QueryString["UserDefinedToolParametersKey"];
    //retrieve UserDefinedParameters from user Session using session key
    object obj = Session[sessionKey];
    if(obj != null && obj is UserDefinedToolParameters)
    {
        //cast object to the appropriate SmarTeam Data type
        UserDefinedToolParameters parameters
        =(UserDefinedToolParameters)obj;
        int i=1;
        //loop thru parameters in the collection and add a row to Table1
        //for each parameter
        for each (UserDefinedToolParameter param in parameters)
        {
            TableRow row = new TableRow();
            Table1.Rows.Add(row);
            TableCell cell1 = new TableCell();
            row.Cells.Add(cell1);
            cell1.Text = "Param" +i.ToString();
            i++;
            TableCell cell2 = new TableCell();
            row.Cells.Add(cell2);
            cell2.Text =
            param.SelectedObjectIdentifier.ClassId.ToString();
            TableCell cell3 = new TableCell();
            row.Cells.Add(cell3);
            cell3.Text =
            param.SelectedObjectIdentifier.ObjectId.ToString();
            TableCell cell4 = new TableCell();
            row.Cells.Add(cell4);
            cell4.Text = param.LinkObjectIdentifier.ClassId.ToString();
            TableCell cell5 = new TableCell();
            row.Cells.Add(cell5);
            cell5.Text = param.LinkObjectIdentifier.ObjectId.ToString();
            TableCell cell6 = new TableCell();
            row.Cells.Add(cell6);
            cell6.Text = param.NodeObjectId.ToString();
        }
    }
}
```

Getting SmarTeam User-defined Page Parameters

SmarTeam provides all of the parameters for User-defined tools (UDT) in the `IUserDefinedToolContext` interface. The selected objects in the diagram (see below) are stored in the interface `IUserDefinedToolContext`. This collection is stored in the ASPX Session and the key to that entry is transferred to the page in a QueryString parameter named: **UdtKey**.

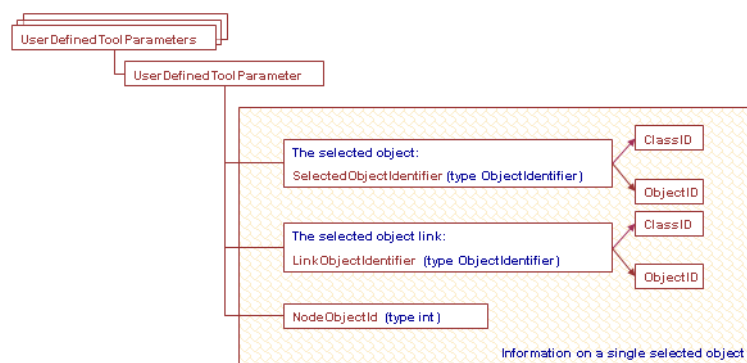


Getting SmarTeam User-defined Page Parameters (before V5R16 SP4)

The selected objects in the diagram (see below) are stored in the `UserDefinedToolParameters`. This collection is stored in the ASPX Session and the key to that entry is transferred to the page in a QueryString parameter named: **UserDefinedToolParametersKey**

Note: Each `UserDefinedToolParameter` contains information on a single selected object:

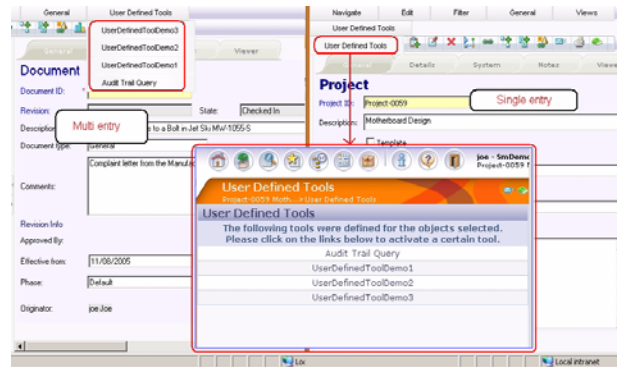
- a SelectedObjectIdentifier** (Type `ObjectIdentifier`): contains information about the selected object itself
- b LinkObjectIdentifier** (Type `ObjectIdentifier`): contains information about the selected object link
- c NodeObjectId** (Type `int`): identifier of the selected node, if the object selected is a workflow process, if not it is -1



User-defined Tools Appearance

User-defined Tools (UDT) can be displayed two ways, as a:

- Single entry in the menu
- Multiple entries in the menu



There is the Boolean key: `userDefinedToolsInOneMenuItem` in the system configuration settings that determines the appearance of the user defined tools:

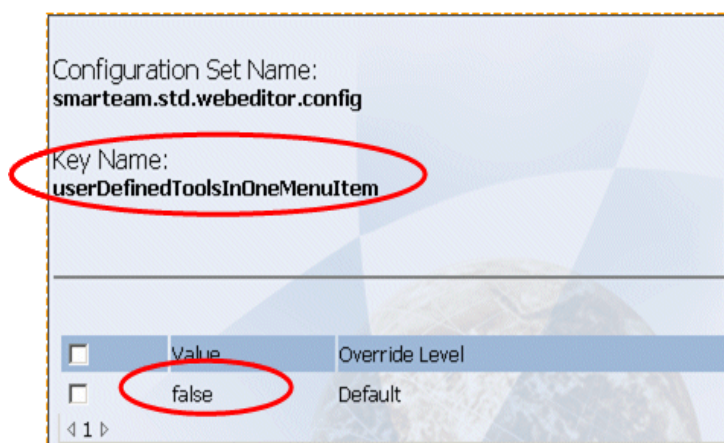
- Value = **false** – Multiple entry (default):

The User-defined tools are presented as a list of items in the Main menu.

- Value = **true** – Single entry:

One menu item named “User Defined tools” is displayed and when clicked, it redirects the user to a page with a list of available User-defined tools.

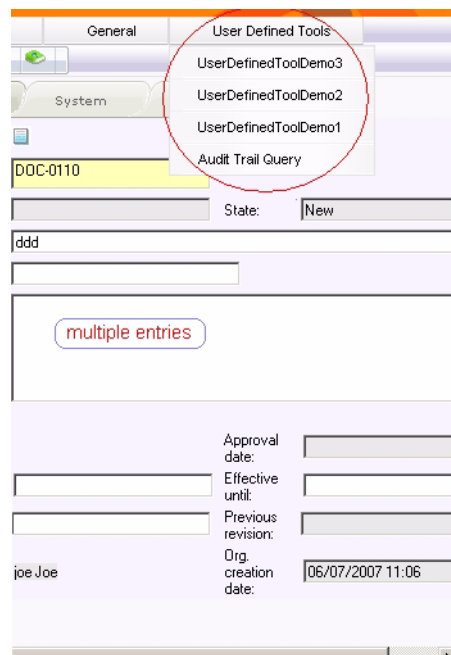
Note: The value is case sensitive, only the strings “true” and “false” are valid.



Example 1

The following example shows the Multi-objects entries Menu with two newly added user-defined tools. These entries were added using the configuration:

```
<userDefinedTools>false</userDefinedTools>
```

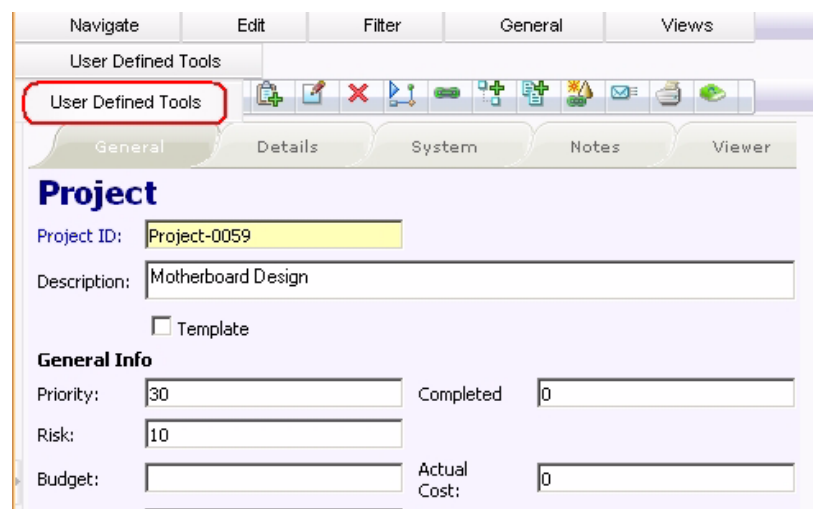


Example 2

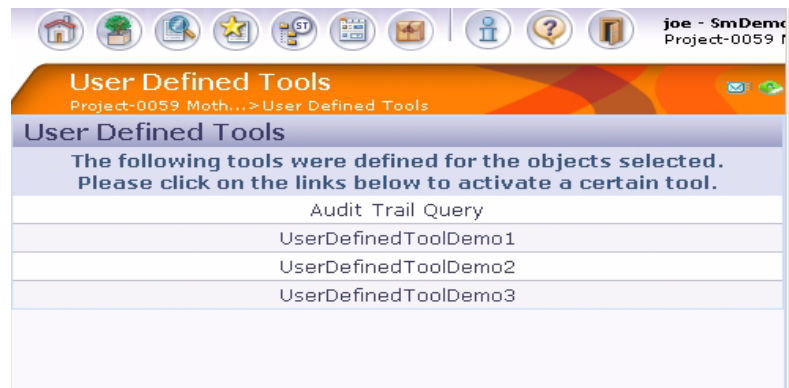
The following example shows the Single-object Actions Menu with one added entry. This entry was added using the configuration:

```
<userDefinedTools>true</userDefinedTools>
```

After highlighting an object, the user can select User Defined Tools sub-menu.



Then the **User Defined Tools**, page opens as shown in this example:



Chapter 4: Workflow Events

SmarTeam – Web Editor enables you to add a user defined page (similar to the user defined tools) that executes during a Workflow operation. SmarTeam – Web Editor enables you to customize the following Workflow events:

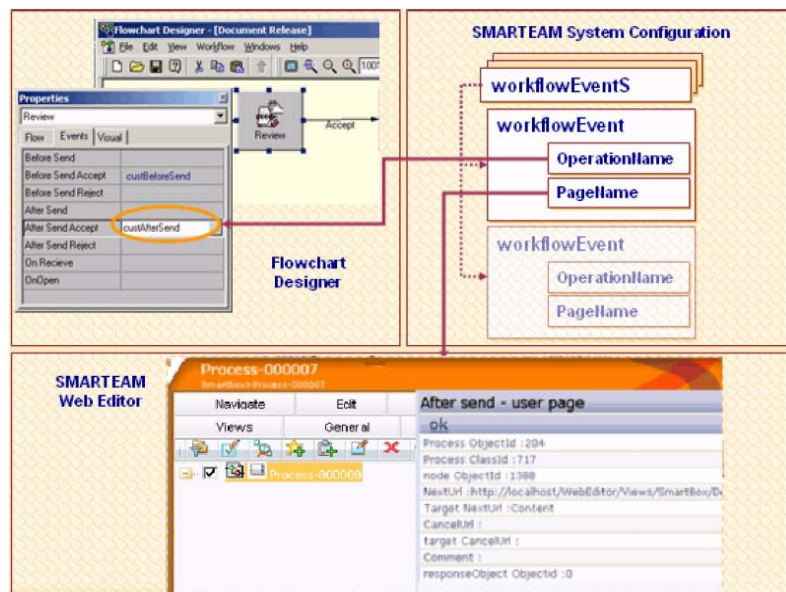
- Before Send Accept.
- Before Send Reject.
- After Send Accept.
- After Send Reject

The screenshot displays two configuration panels for workflow events in SmarTeam Web Editor. Both panels are titled 'Before send - user page' and 'After send - user page' and are enclosed in dashed orange borders. The 'Before send - user page' panel has a header bar with 'ok' and 'Cancel' buttons. The 'After send - user page' panel has a header bar with an 'ok' button. Both panels contain a list of properties and their values.

| Property | Value |
|-------------------------|--|
| Process Objectid | :204 |
| Process Classid | :717 |
| Node Objectid | :1388 |
| NextUrl | :/WebEditor/Views/SmartBox/Response.aspx?vd |
| Target NextUrl | : |
| CancelUrl | :http://localhost/WebEditor/Views/Object/Def |
| Target CancelUrl | :Content |
| Comment | : |
| responseObject Objectid | :3 |

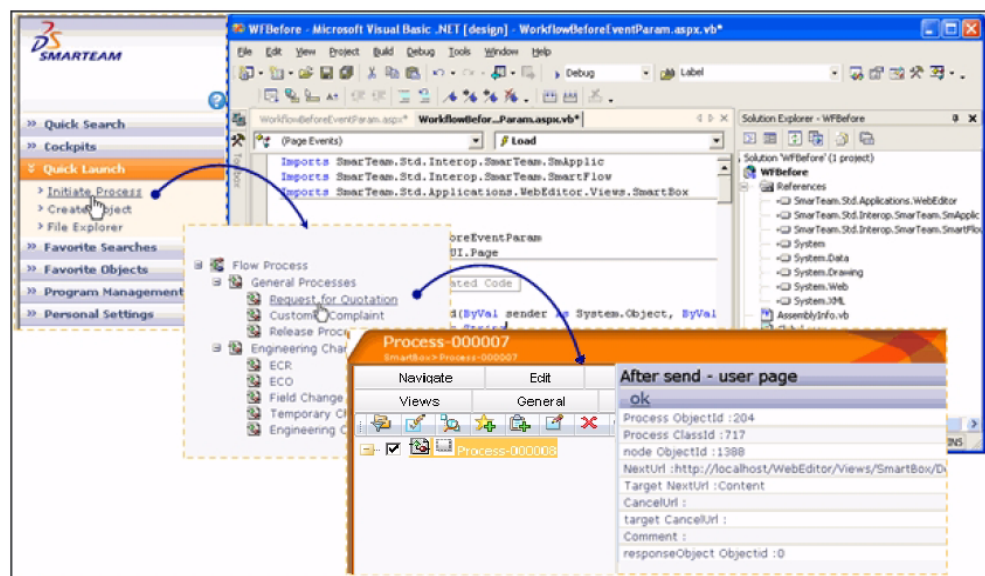
| Property | Value |
|-------------------------|--|
| Process Objectid | :204 |
| Process Classid | :717 |
| Node Objectid | :1388 |
| NextUrl | :http://localhost/WebEditor/Views/SmartBox/D |
| Target NextUrl | :Content |
| CancelUrl | : |
| target CancelUrl | : |
| Comment | : |
| responseObject Objectid | :0 |

Creating a User-defined Page for a Workflow Operation



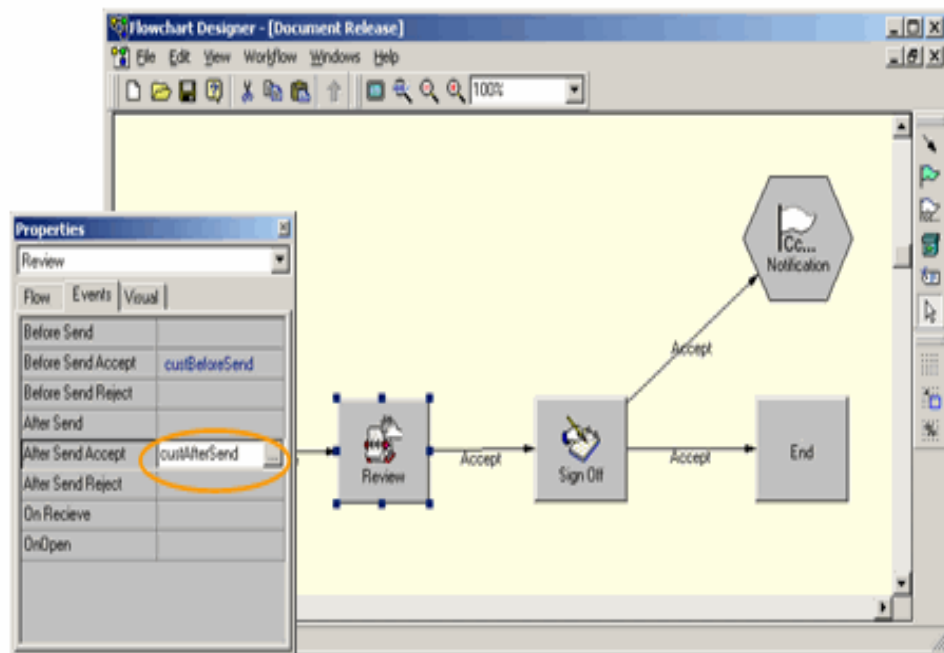
To connect a user-defined page:

- 1 In VB.Net, create User-defined .ASPX page.



- 2 There are two ways to define a script in the Flow Chart Designer:
 - Write the name of the function in the Node Property (Tab Event) for one or more of the workflow events:
 - Before Send Accept
 - Before Send Reject
 - After Send Accept
 - After Send Reject

- In the Flow Chart Designer, connect the dummy function of the Workflow event using the browse button:



- 3 Verify that the workflow functions have special parameters:

The screenshot shows the 'WebCustomize.BS - SmartScript Editor' application. The editor contains the following code:

```

*
* This Dummy script is used only As a label for assigning
* User defined page to the given classWorkflow Event
*
-----
Function custBeforeSend(ActiveProcess As Object, Response As Object) As Integer
End Function

Function custAfterSend(FlowSession As Object, FlowProcess As Object, _
    Node As Object, Response As Object ) As Integer
End Function

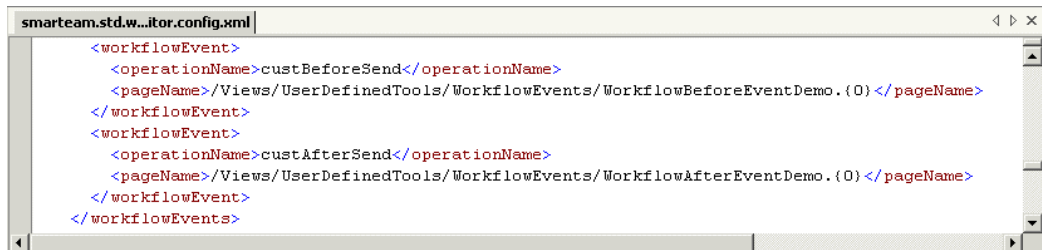
```

The status bar at the bottom indicates 'Ready...', 'Line: 16', 'Col: 79', and 'Modified'.

- 4 Define the workflowEvents key in SmarTeam Configuration settings:
 - The key contains the schema:



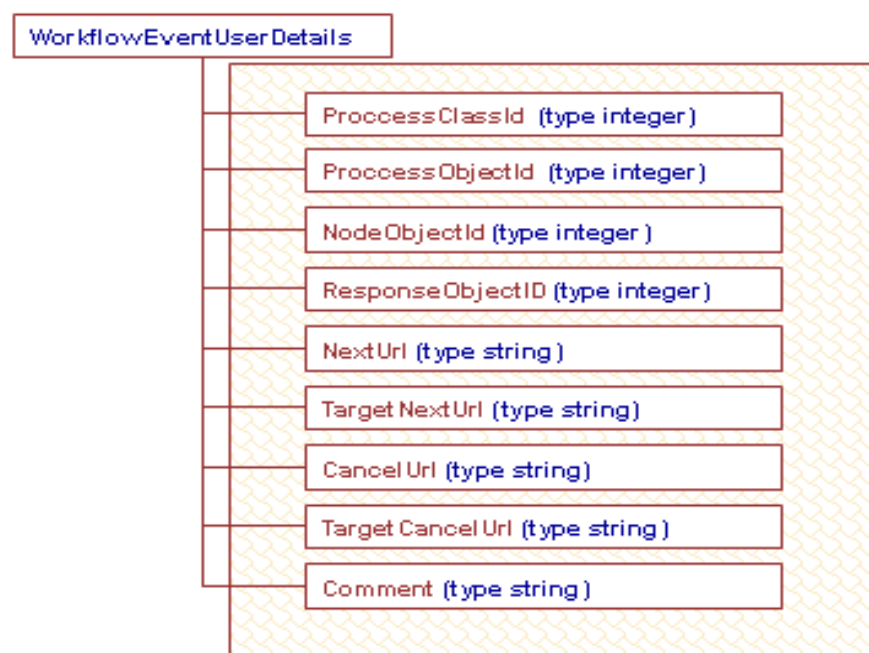
- 5 In SmarTeam System Configuration, manually define the workflowEvents key:
 - Navigate to the **workflowEvents** key at the path: <SmarTeam>/ConfigurationSettings/Data folder, using the Search feature
 - Use a text editor to open **smarteam.std.webEditor.config.xml**
 - Add a new entry in the workflowEvents key



- a Root folder for WorkFlow events is <SmarTeam>\WebEditor\Web\views\userdefined tools\workflowevents
- b Backup the configuration settings located in the <SmarTeam>\ConfigurationSettings\Data folder
- c See examples of work flow events ASPX files at path: <smarteam dir> \WebEditor\Web\Views\UserDefinedTools\WorkflowEvents\ **WorkflowAfterEventDemo.aspx** and **WorkflowBeforeEventDemo.aspx**

Using the Selected Objects in a User-defined Page

The Workflow Event Parameters are stored in the WorkflowEventUserDetails. This collection of parameters is stored in the ASPX Session and the key to that entry is transferred to the page in a QueryString parameter named: **guildForWFEvent**.



- 1 Set a reference to **SmarTeam.Std.Applications.WebEditor.dll** from the WebEditor Bin directory.
- 2 Verify that the interface **WorkflowEventUserDetails** is present in the library:
SmarTeam.Std.Applications.WebEditor.Views.SmartBox

```
Imports SmarTeam.Std.Interop.SmarTeam.SmApplic
Imports SmarTeam.Std.Interop.SmarTeam.SmartFlow
Imports SmarTeam.Std.Applications.WebEditor.Views.SmartBox

Private Sub Page_Load(ByVal sender As System.Object, ByVal
    Dim sessionKey As String
    'Get session key from QueryString
    sessionKey = Request.QueryString("guiIdForWFEEvent")

    'Retrive WorkflowEventUserDetails from
    'ASP.NET Session using session key
    Dim obj As Object
    obj = Session(sessionKey)

    'Casting an Object to WorkflowEventUserDetails
    Dim eventDetails As WorkflowEventUserDetails
    eventDetails = obj
    ...
```

The Object transferred by the session is **WorkflowEventUserDetails**. The Properties of the object are:

- **ProcessObjectId**: The selected process' object ID
 - **ProcessClassId**: The selected process' class ID
 - **NodeObjectId**: The selected node's ID
 - **ResponseObject**: The response Object's ID
- 3 Open SmarTeam and assign the parameters to **OK** or **Cancel** hyperlinks:
 - **NextUrl**: URL that the page navigator is directed to after a successful operation
 - **TargetNextUrl**: The Target URL refers to the target frame name (on screen)
 - **CancelUrl**: URL that the page navigator is directed to when a operation is canceled
 - **TargetCancelUrl**: This is the cancel frame name (on screen)
 - **Comment**: The process' comment
 - **SendMail**: Whether or not to send mail

```
HyperLinkOK.NavigateUrl = eventDetails.NextUrl
HyperLinkOK.Target = eventDetails.TargetNextUrl
HyperLinkCancel.NavigateUrl = eventDetails.CancelUrl
HyperLinkCancel.Target = eventDetails.TargetCancelUrl
```

Note: See an example at path: <smarteam dir>
 \WebEditor\Web\Views\UserDefinedTools\WorkflowEvents\WorkflowBeforeEventDemo.aspx.

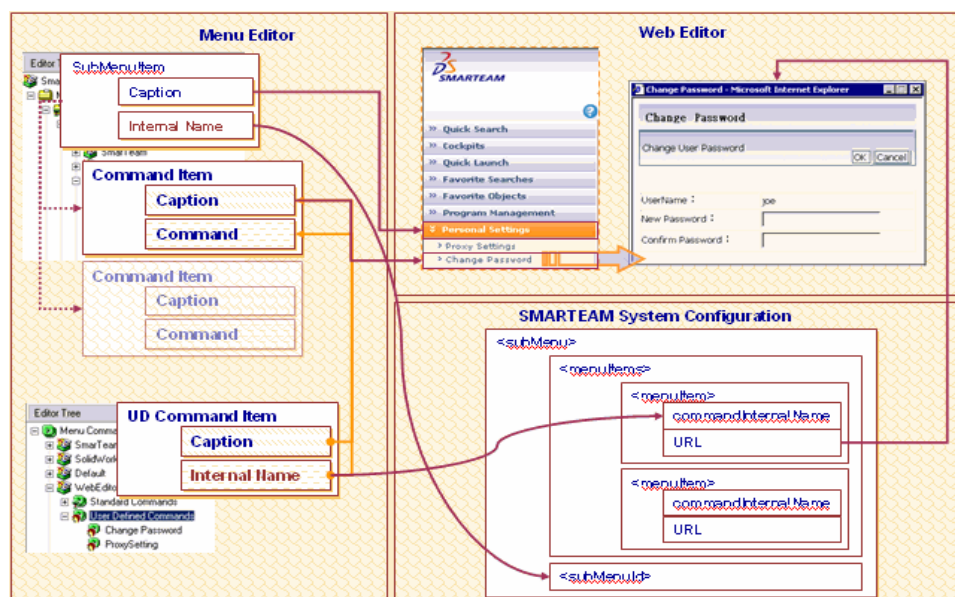
Chapter 5: Application Toolbar

SmarTeam – Web Editor enables you to customize the Application toolbar, by adding your own entries to the Application toolbar menus.

Example:



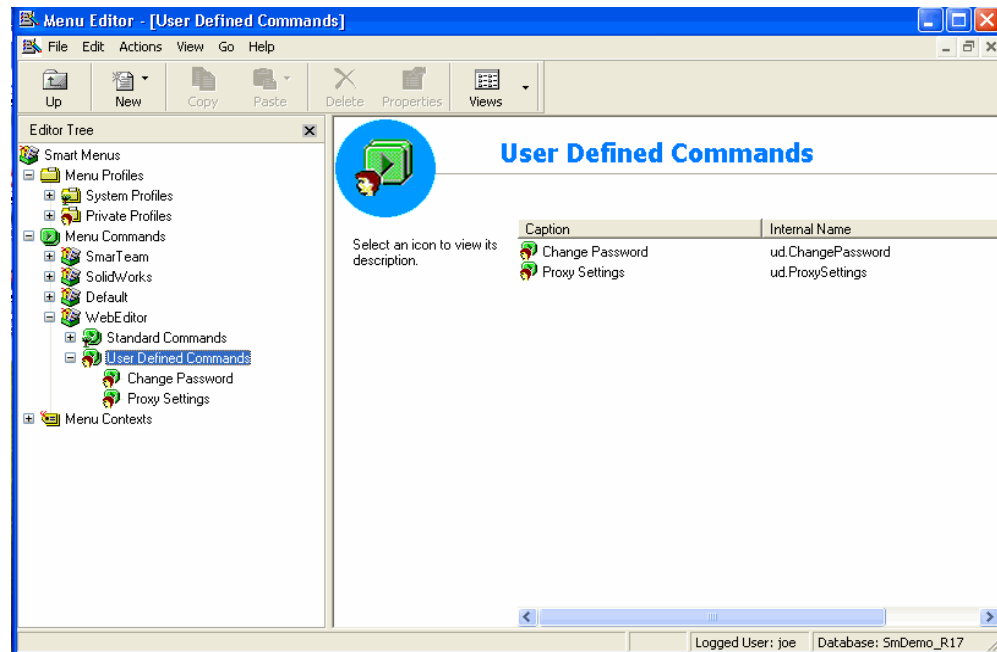
The Schema connects the UDPage to a command in the application toolbar:



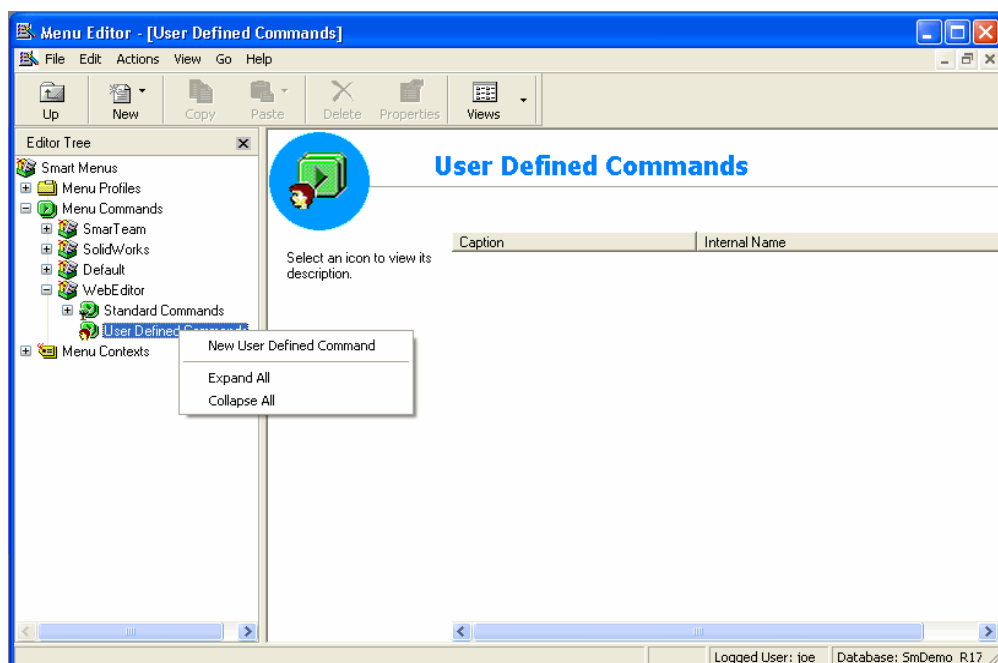
To add a new entry to the Application Toolbar:

- 1 In SmarTeam – Editor, open the **Menu Editor**.
- 2 Select **Menu Commands > WebEditor > User Defined Commands**


The User Defined Commands window appears:

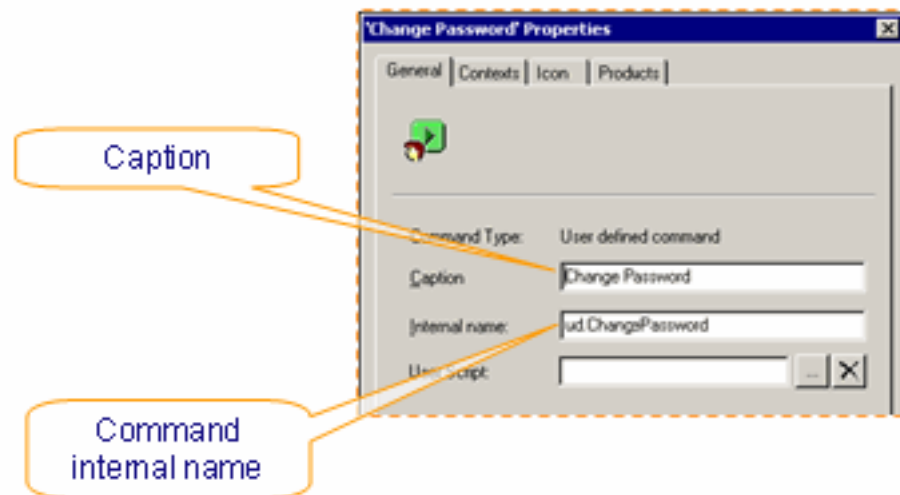


- 3 Right-click **User Defined Commands** and select **New User Defined Command**.



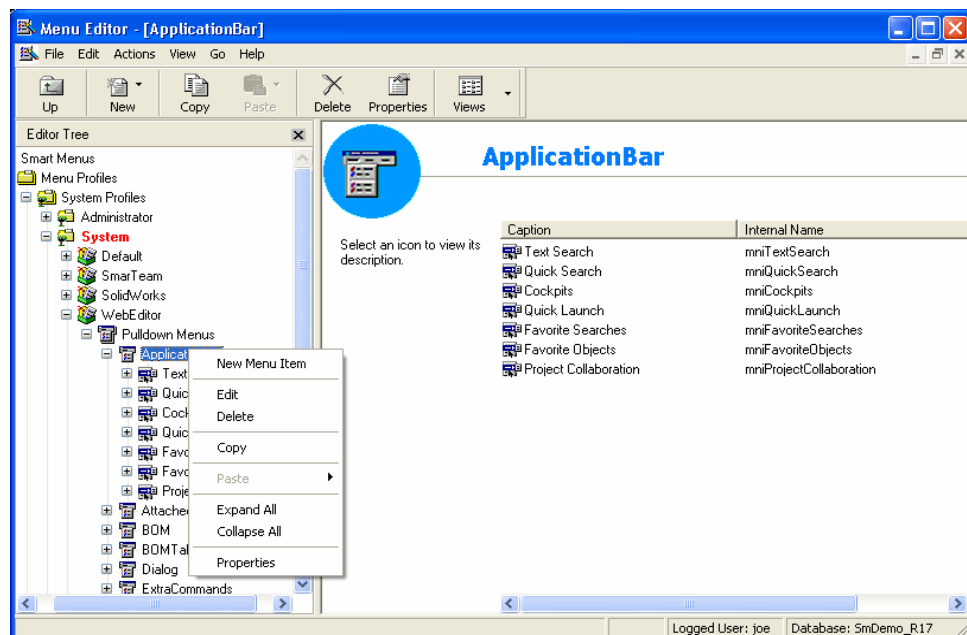
- 4 In the New User Defined Command dialog:
 - Enter a new user defined command name (e.g., Proxy Setting) in the Caption field
 - The corresponding internal name appears automatically in the Internal name field

- Click  and select a **User Script** from the list
- Repeat previous 3 steps for each new user defined command you need
- Click **OK**



5 In the Menu Editor tree:

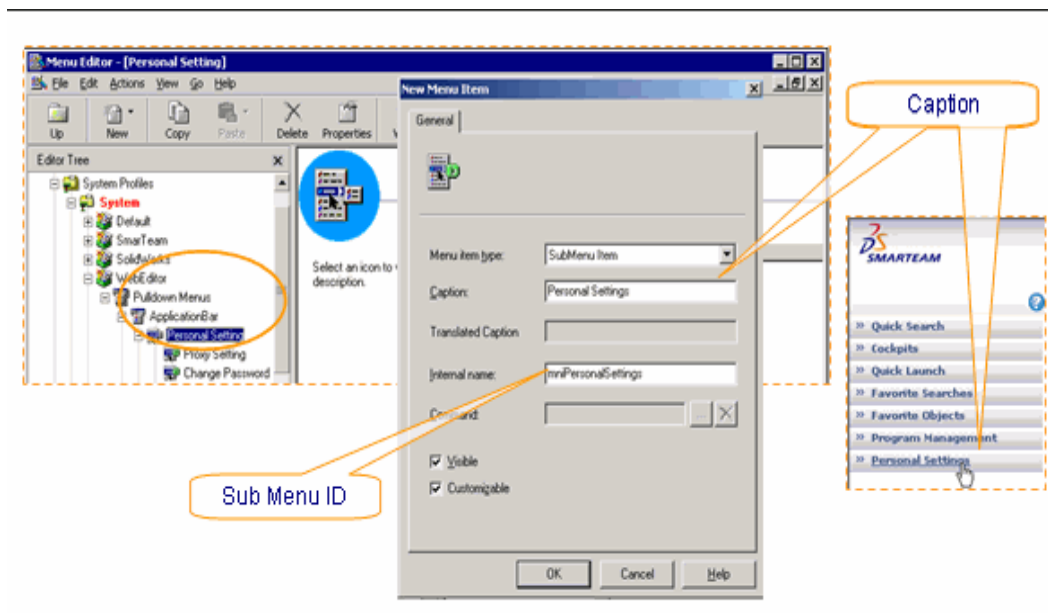
- Select **System Profiles > System > WebEditor > Pulldown Menus > Application Bar**
- Right-click **Application Bar** and select **New Menu Item**



Note: The caption is the name of the submenu that will be displayed in the SmarTeam - Web Editor Application bar. Also note that the Internal Name appears automatically when you enter the name in the Caption field.

6 In the New Menu Item dialog:

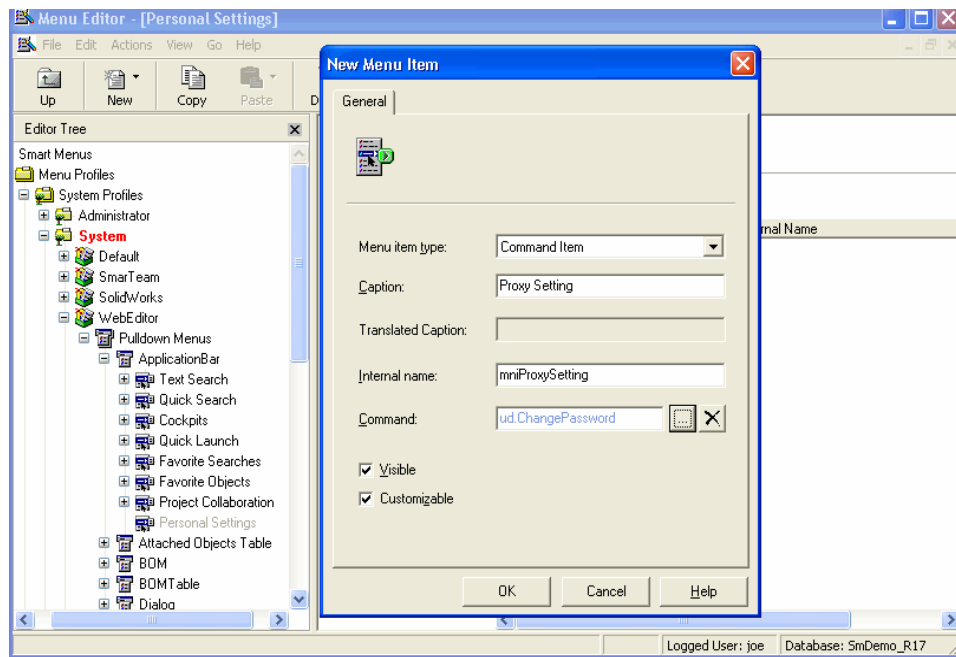
- Select **SubMenuItem** from the Menu Item Type field list
- Enter a New Menu Item name (e.g., **Personal Settings**) in the Caption field
- The corresponding internal name appears automatically in the Internal name field
- Click **OK**



7 Right-click **SubMenuItem** (e.g., Personal Settings) and select **New Menu Item**.

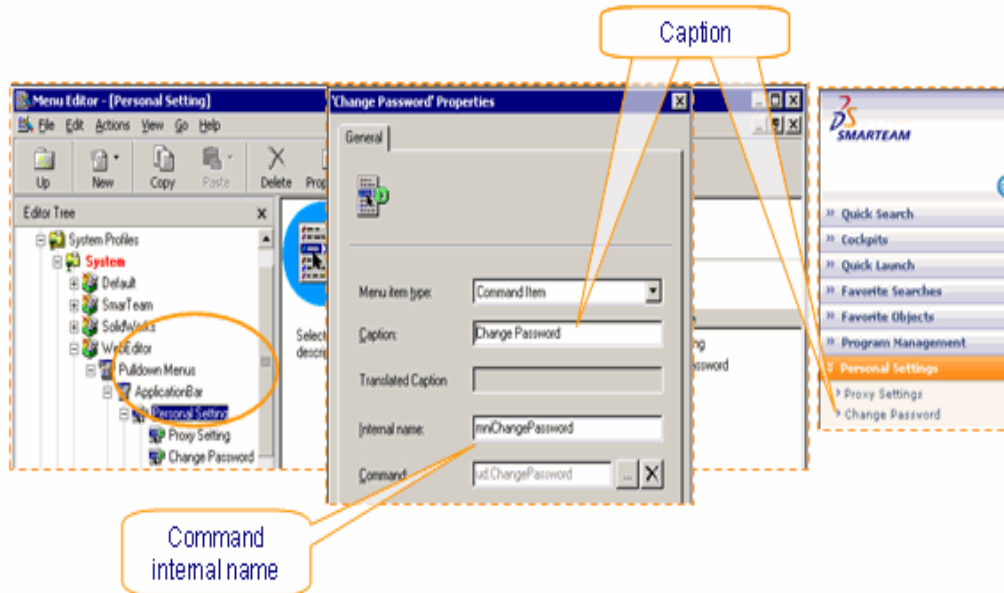
8 In the New Menu Item dialog:

- Add a new menu item of type **Command Item** to the new submenu.
- Enter a New Menu Item name (e.g., **Proxy Setting**) in the Caption field
- The corresponding internal name appears automatically in the Internal name field
- Repeat previous 3 steps for each new command item you need
- Click **OK**



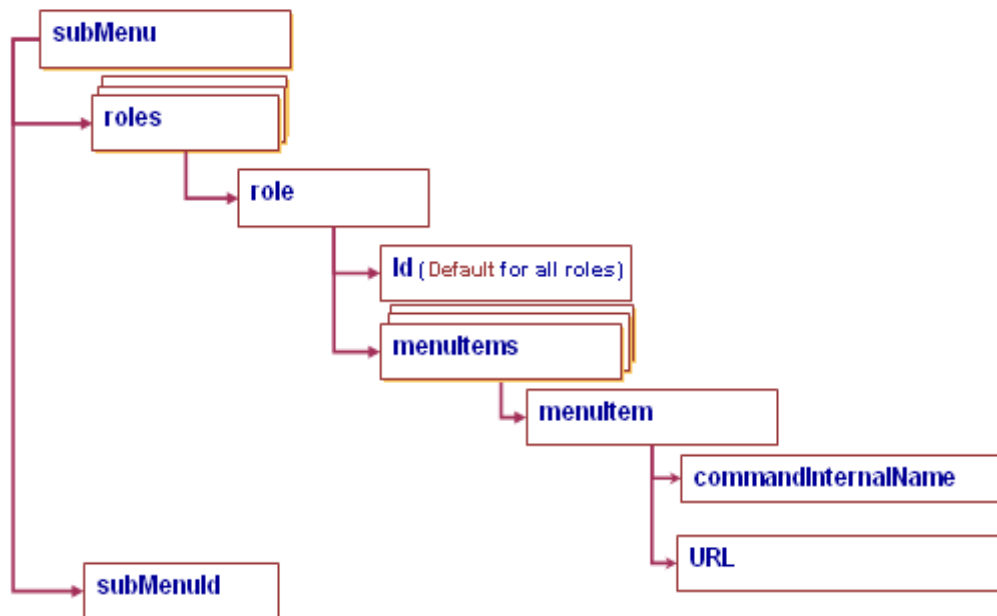
Example:

Add two submenus - the first with a caption "Proxy Setting" and internal name "ud.ProxySetting" and the second with a caption "Change Password" and internal name "ud.ChangePassword". Attach any command to both of them.



- 9 Define the applicationBar key in SmarTeam Configuration settings.

The key contains the following schema.



Manually Defining an applicationBar key in the SmarTeam System Configuration:

Manual changes need to be made to the SmarTeam System Configuration if changes were not made by the System Configuration Editor.

To make the manual changes to the System Configuration:

IMPORTANT! Before making changes to the XML files, it is highly recommended to backup the [SmarTeam home directory]\ConfigurationSettings\Data folder.

- 1 On SmarTeam Services server navigate to <SmarTeam >\Configuration Settings\Data.
- 2 In the file **smarteam.std.webEditor.config.xml**, change the configuration settings as you require.



Note: The path to the root Application Toolbar folder is: <SmarTeam>\WebEditor\Web

Chapter 6: Adding Your Own Cockpit

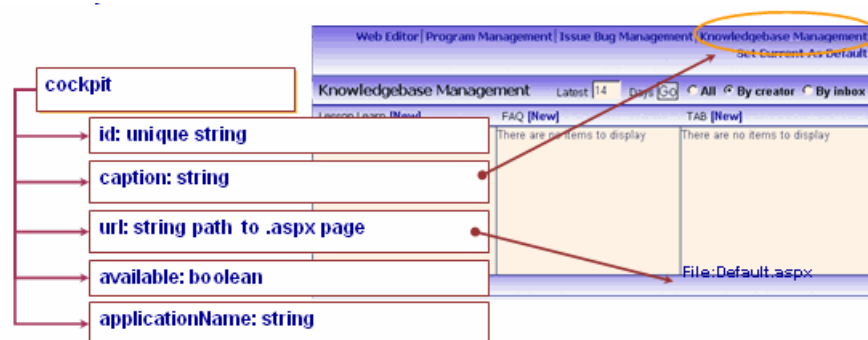
SmarTeam – Web Editor enables you to add your own cockpits. You can create any cockpit required.



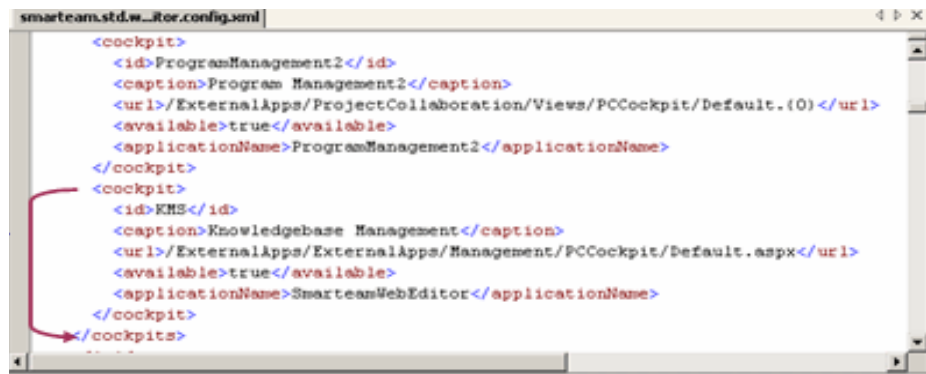
To add a Cockpit:

- 1 In VB.Net, create User-defined .ASPX page.
- 2 Define the **cockpit's** key in SmarTeam Configuration settings:

The key contains the schema:



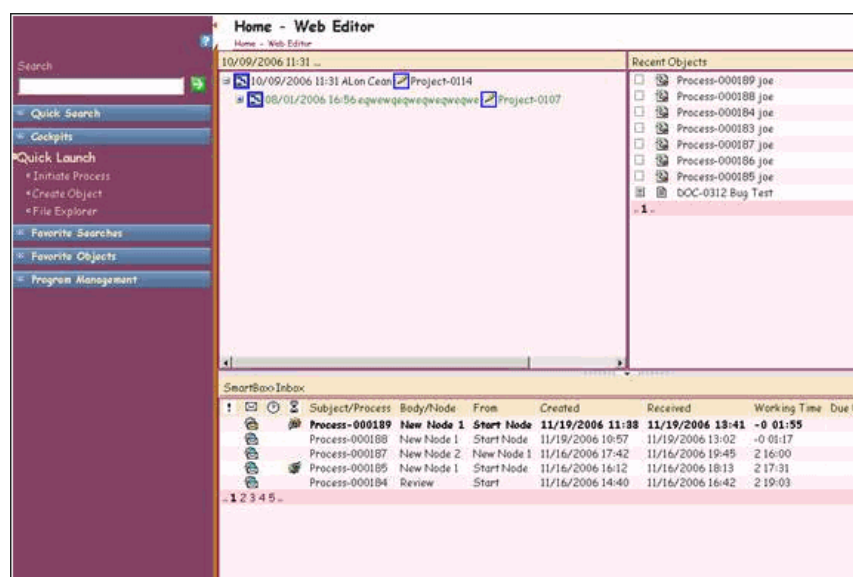
- 3 Define the cockpit's key using SmarTeam System Configuration manually:
 - Locate the cockpit's key using Search: <SmarTeam>/ConfigurationSettings/Data folder
 - Using a text editor, open **smarteam.std.webEditor.config.xml**
 - Add a new entry in the cockpit's key

**Notes:**

- a** Root folder for cockpits is <SmarTeam>\WebEditor\Web\
- b** All SmarTeam cockpits are located in <SmarTeam>\WebEditor\Web\Views\Cockpit.
In this case your url key will be:
<url>\Views\Cockpit\CustomerCockpit.aspx</url>
- c** Backup the configuration settings located in the <SmarTeam>\ConfigurationSettings\Data folder.

Chapter 7: Look and Feel

SmarTeam – Web Editor enables you to customize the application's look and feel, by making changes in the Cascading Style Sheets (CSS) files that define the style of the application. These files are located on the server machine.



The application identifies the browser's platform and uses the styles defined in the corresponding style sheets.

The following table shows the location of the different style sheets available:

| Browser and Platform | Location of Style Sheet |
|----------------------------------|--|
| Netscape and Mozilla on UNIX | <smarteam dir>\WebEditor\Web\StyleSheet\UNIX\NETSCAPE |
| Netscape and Mozilla on Windows | <smarteam dir>\WebEditor\Web\StyleSheet\Windows\NETSCAPE |
| All other browsers and platforms | <smarteam dir>\WebEditor\Web\StyleSheet |

These directories contain .css files that define different areas of the application:

| .css File | Description |
|-------------------|---|
| GridSpecific.css | Defines styles for all the grids |
| LeftAreaStyle.css | Defines styles for the left area (search results, trees, etc) |

| | |
|---------------------------|---|
| LifecycleStyleSheet.css | Defines styles for the lifecycle process |
| LoginStyle.css | Defines styles for the login pages |
| RightAreaStyle.css | Defines styles for the left area (profile card, etc) |
| ApplicationArea-Style.css | Defines styles for the remainder of the application (toolbars, application bar, etc.) |

In the CSS files, you find entries describing fonts, colors, sizing ,etc.

For example: To change the color of all the default titles in the application to RED, you modify the color value of the following entry in the ApplicationAreaStyle.css

```
.DefaultTitle
{
    font-size: x-small;
    color: #FF0000;
}
```

Note: Be aware that changing font sizes can affect the look of the view in general.

Adding a Company Logo

You can customize the SmarTeam – Web Editor Main Toolbar and login page by adding a Company logo of your choice.

To add a logo to the Main Toolbar:

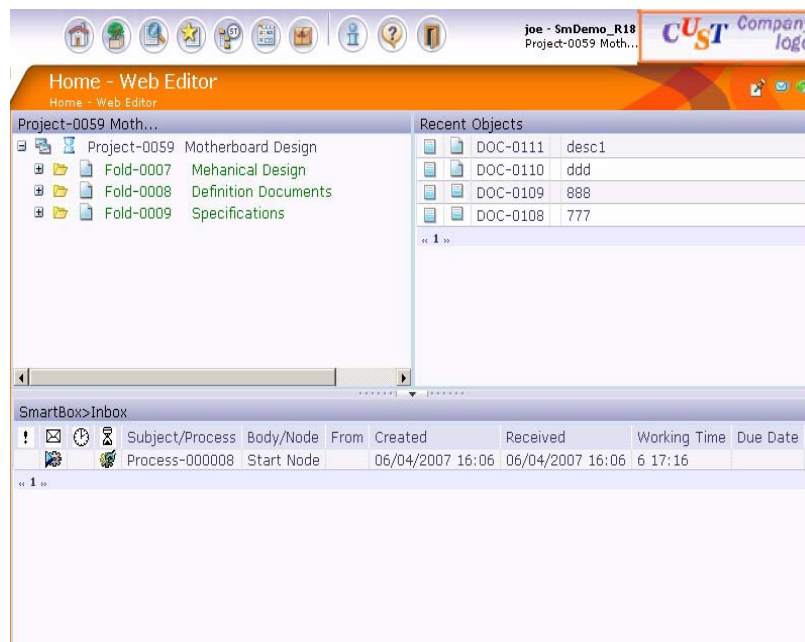
- 1 Browse to the SmarTeam folder (default path: C:\Program Files\SmarTeam).
- 2 In the SmarTeam folder, browse to **CompanyLogoDefault.gif** (path: SmarTeam\WebEditor\Web\Media\Images)
- 3 Rename CompanyLogo.gif to CompanyLogoDefault.gif (or any name you select)

Note: The Company logo file you add must be in Graphics Interchange Format (GIF), with .gif file extension. The default logo dimensions are 165 X 45 pixels. Verify that your logo fits the specified dimensions. Otherwise, it will not display correctly.

- 4 Copy the file containing the selected Company logo to this folder.

Note: If you do not rename your company logo picture: CompanyLogo.gif, it will not display.

- 5 Rename the file to **CompanyLogo.gif**.



- 6 To replace your company logo, replace it with a new logo name **CompanyLogo.gif**.
- 7 To remove the Company logo from the file, delete the file and rename the CompanyLogoDefault.gif file to CompanyLogo.gif.

Replacing Login with SmarTeam – Web Editor Login Page

To replace login with SmarTeam – Web Editor Login:

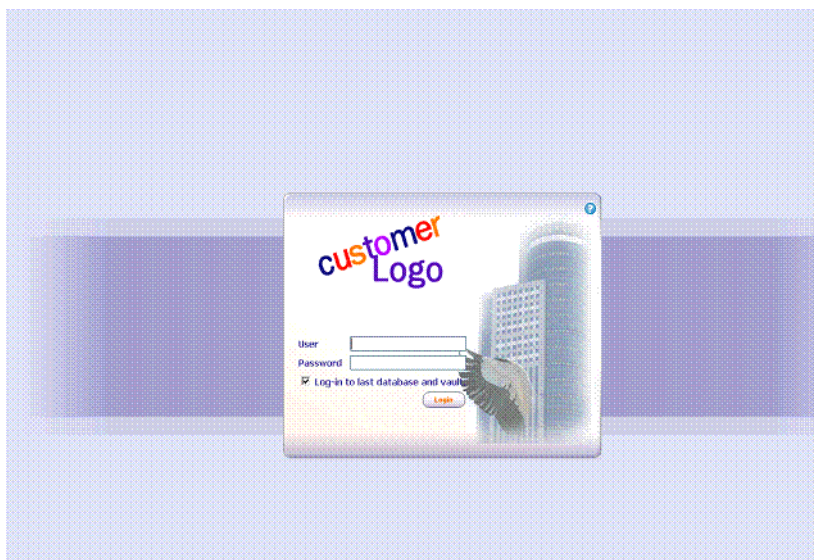
- 1 Browse to the SmarTeam folder (default path: C:\Program Files\SmarTeam).
- 2 In the SmarTeam folder, browse to the **Login_picture.gif** (SmarTeam\WebEditor\Web\Media\Images>LoginImages)
- 3 Rename Login-picture.gif to Login_Picture_Default.gif (or any name you select).

Note: The Company logo file you add must be in Graphics Interchange Format (GIF), with .gif file extension. The default logo dimensions are 165 X 45 pixels. Verify that your logo fits the specified dimensions. Otherwise, it will not display correctly.

- 4 Copy the file containing the selected Company logo to this folder.

Note: If you do not rename your login picture: Login_picture.gif, it will not display.

- 5 Rename the file to **Login_picture.gif**.



- 6 To replace your login page, replace it with a new login name **Login_picture.gif**.
- 7 To remove the Company logo from the file, delete the file and rename the Login_Picture_Default.gif file to Login_picture.gif.

Chapter 8: Profile Cards

Adding Client-side Scripts

SmarTeam - Web Editor enables you to add a script of programming commands to be executed before a profile card is opened.

Note: SmarTeam – Web Editor Scripts cannot be attached to the UnLoad event.

Client-Side scripts can be hooked to a:

- Form: On Load. Event On Unload does not implement
- Control: On Enter or On Exit
- Button and URL control on the Profile Card – On Click
- Image without events

Client-Side scripts are written in JavaScript and must be entered via the WEB Form Designer. Each attached JavaScript has the following format:

```
function MyFunction ( ProfileCardClientObj, ProfileCardObj )
{
    // The content JavaScript function
}
```

Where the:

- ProfileCardClientObj (Control) is not implemented yet
- ProfileCardObj contains the actual data of the object
- Name of the parameter is fixed and cannot be changed

Several sample scripts are included in the [Appendix A](#) that illustrates what you can do with client-side scripts. You can:

- Retrieve Control/Controls details
- Retrieve Profile card details
- Retrieve Object values
- Use the ListItemsCollection
- Make a control visible
- Set a control/value text
- Change background's property

Example: Adding a New Button to a Profile Card

The Web Form Designer is used to design and modify the layout and content of profile cards for SmarTeam – Web Editor.

The Profile Card provides an organized interface that displays a Class' attributes. A Class refers to a set of objects that share common structure and common attributes. Each Class that is defined in SmarTeam, e.g., Documents, Materials, Items, has its own type of Profile Card.

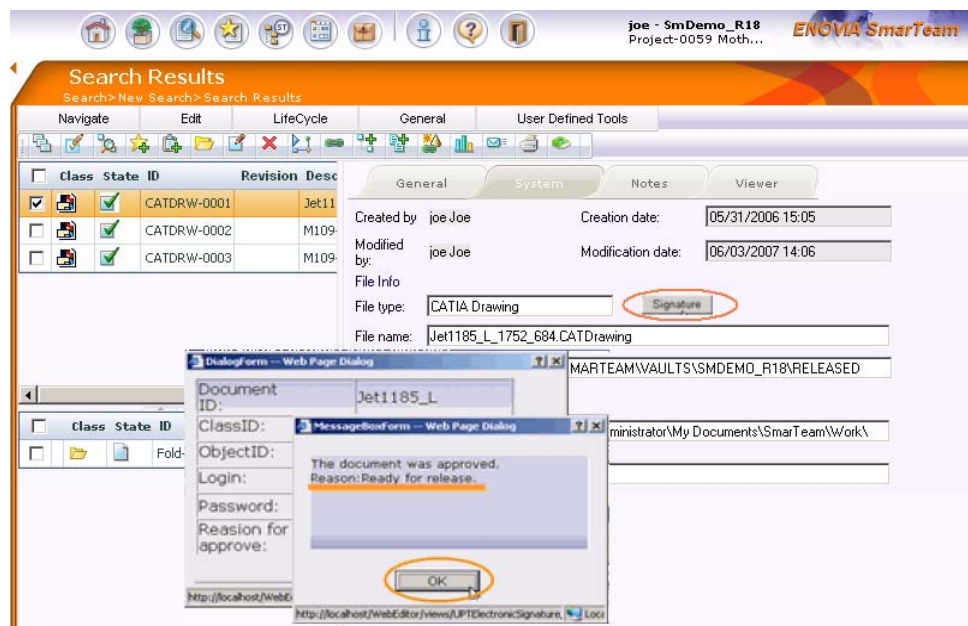
You design and customize a Profile Card using the Web Form Designer's working form. All changes made in the Web Form Designer are reflected in the finished Profile Card.

Profile Card definitions are stored in the database per class and are migrated into SmarTeam – Web Editor, the first time it is loaded.

For more information, refer to Web Form Designer topic in the SmarTeam – Editor Online Help.

The following example shows how to:

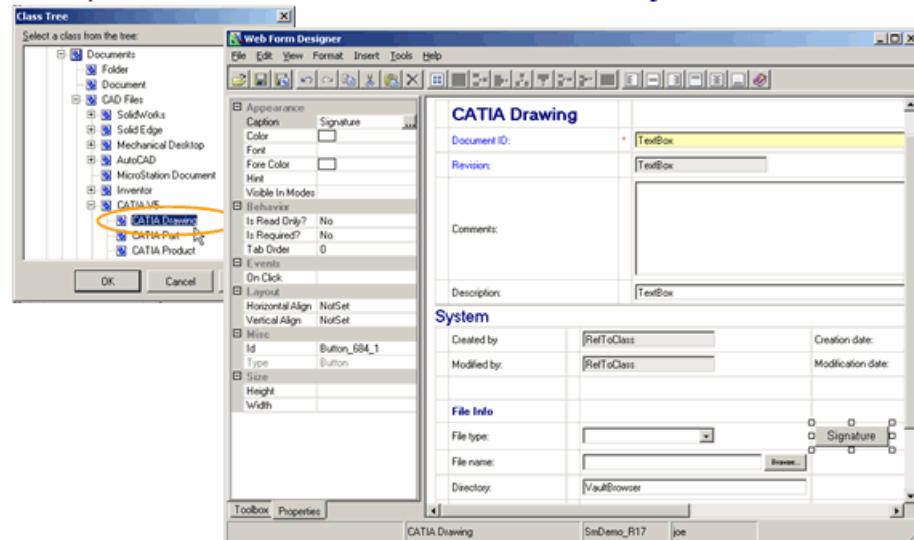
- Add a new button for profile card
- Open a customer application on click
- Return result value to SmarTeam – Web Editor



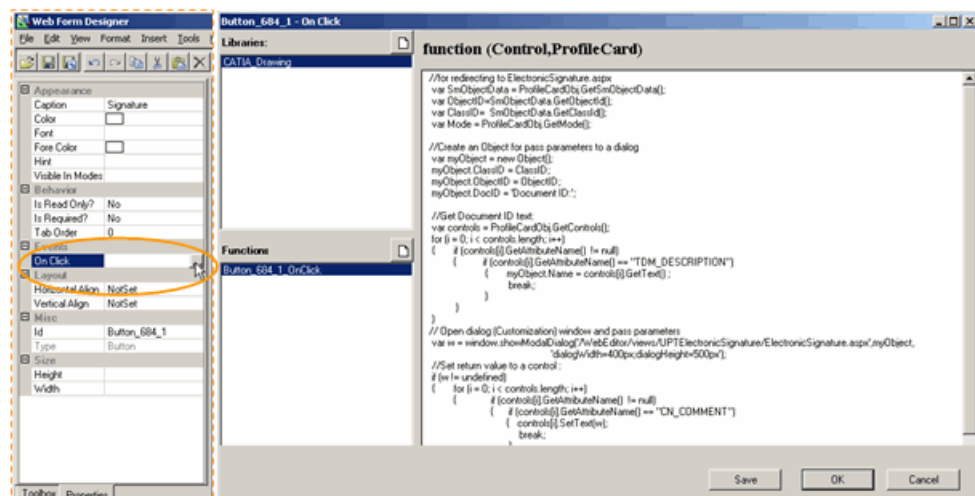
To customize a Profile Card:

- 1 In the Web Form Designer, open the project Profile Card.
- 2 Add a new button named **Electronic Sign** near the Approval Date text box:
The script opens a new window, ElectronicSignature.aspx.
- 3 Create an ElectronicSignature.aspx page that runs when the button is clicked:
 - Type login and password for validation
 - Type Approval Reason
 - Click **OK** to run Validation.

- 4 Create **Validate.aspx** for Validation of all parameters and return result:
 - Type the **Approval Reason** value in the Profile Card (text box) TDM_SIGNATURE field
- 5 Customize the Profile Card in Web Form Designer.



- 6 Add Script OnClick button **Electronic Sign**.



The following code examples show how to:

- Get parameters from a SmarTeam application
- Open a customer page
- Return a parameter to the SmarTeam application

a Get parameter from SmarTeam – Web Editor

```
//for redirecting to ElectronicSignature.aspx
var SmObjectData = ProfileCardObj.GetSmObjectData();
var ObjectID=SmObjectData.GetObjectId();
var ClassID= SmObjectData.GetClassId();
var Mode = ProfileCardObj.GetMode();
```

Create an Object to pass parameters to a custom dialog:

```
var myObject = new Object();
myObject.ClassID = ClassID;
myObject.ObjectID = ObjectID;
myObject.DocID = 'Document ID:'
```

b Open Custom dialog:

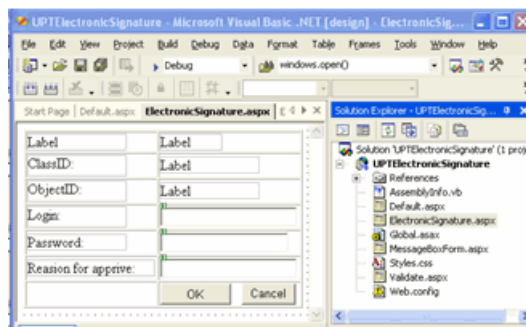
```
var w = window.showModalDialog
('/WebEditor/views/UPTElectronicSignature/ElectronicSignature.aspx',myObject,
'dialogWidth=400px;dialogHeight=500px');
```

c Return parameter to Web Editor:

```
for (i = 0; i < controls.length; i++)
{
    if (controls[i].GetAttributeName() !=
null)
    {if (controls[i].GetAttributeName() ==
"CN_COMMENT")
    { controls[i].SetText(w);
break;
}
}
}
```

7 Create an ElectronicSignature.aspx page in Visual Studio .Net.

■ Design the page



■ Client-Side, write Functions in Java Script:

- On Load add function in Java Script for load parameters:

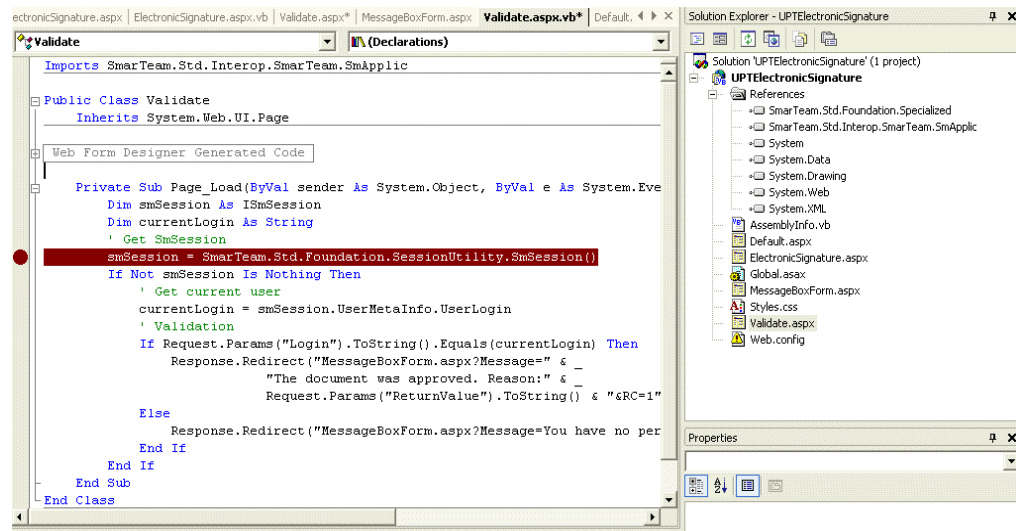
```
function Dialog_OnLoad()
{
    var lblClassID, lblObjectID;
    lblClassID = document.getElementById('lblClassID');
    lblObjectID = document.getElementById('lblObjectID');
    var myObject = window.dialogArguments;
    lblClassID.innerText = myObject.ClassID;
    lblObjectID.innerText = myObject.ObjectID;
}
```

- OnClick **OK** button call to Validate.aspx and after close check if you need to return value:

```
function OK_OnClick()
{
    var w =
    window.showModalDialog('Validate.aspx?ReturnValue=' +
    document.getElementById('txtReturnValue').value + '&Login=' +
    document.getElementById('txtLogin').value + '&ClassID=' +
    document.getElementById('lblClassID').value + '&ObjectID=' +
    document.getElementById('lblObjectID').value );

    if (w.toString() == '1')
    {
        window.returnValue =
        document.getElementById('txtReturnValue').value;
        window.close();
    }
}
```


8 Work with SmarTeam objects (Validate.aspx).



9 Build and Test.

Chapter 9: Adding Server-Side Scripts

SmarTeam - Web Editor enables you to add Server-Side Scripts. This chapter provides you with the information required to:

- Implement Server-Side scripts in SmarTeam Web applications
- Develop and implement Server-Side scripts
- Understand the difference between and usage of the standard hook and the low level interfaces

Prerequisites

To perform the tasks related to adding Server-Side scripts you need the following knowledge:

- Script Event hooks
- Generic and library-specific script hooks (e.g., SmarTeam – Workflow)
- Format of the script argument structure for each type of hook
- SmarTeam Object Model (COM API)

Note: For more information refer to:

- Server-Side Hooks for Server-Based Applications.pdf
- Client-Side Hooks for Client-Based Applications.pdf
- API/SDK

When are Server-Side Scripts Used?

Server-Side hooks are used for customization of any SmarTeam Web based application:

- SmarTeam – Web Editor
- SmarTeam – Community Workspace
- SmarTeam – Program Management
- Embedded Scripts

Also, they can be used for custom applications that use the SmarTeam API and the ServerMode flag that is set to True:

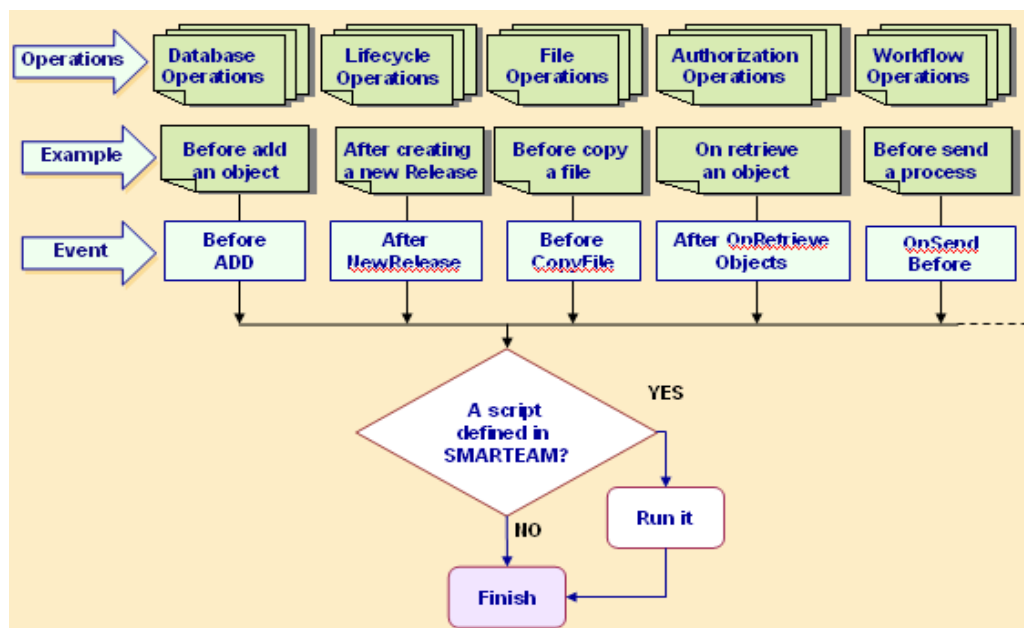
```
SmEngine.ServerMode = True
```

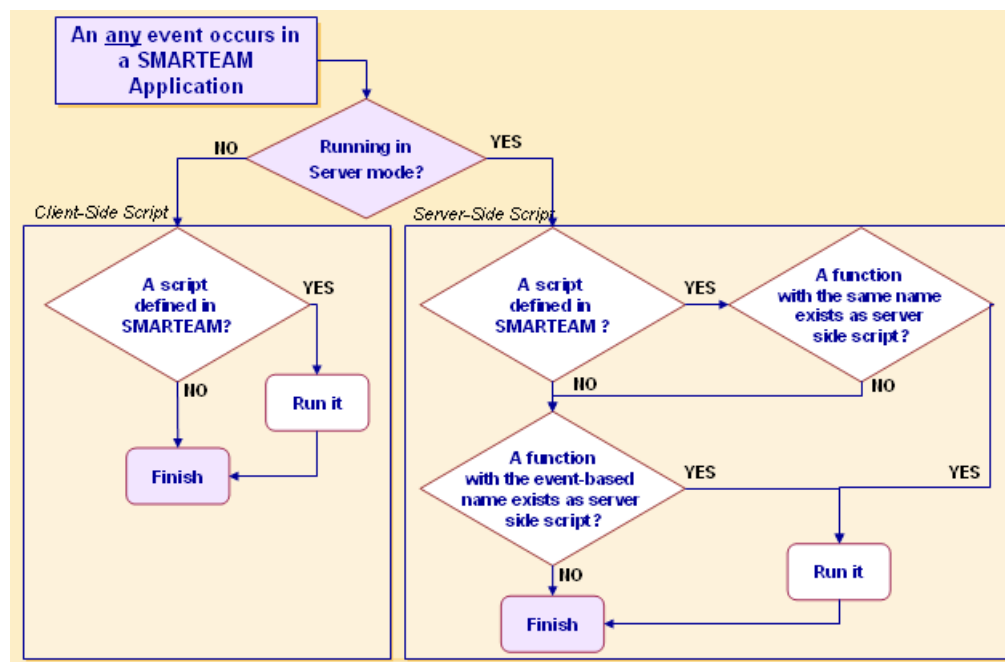
Existing SmartBasic client scripts that work in non-Server Mode do not work in a Server Mode environment. SmarTeam GUI-based hooks cannot be used in Server Mode. You can handle these hooks with client side scripts:

- Controls - On Enter, On Exit
- Forms - On Load, On Unload
- Buttons - On Click

Script Execution Logic

An **any** event occurs in a SmarTeam Application:





Server-Side Script Parameters

Server-Side Script parameters depend on the type of SmarTeam hook:

■ Generic hooks:

- Database operations on Objects
- Lifecycle Operations
- File Operations
- Authorization Operations
- CAD Operations
- Import/Export Operations

■ Library-Specific hooks:

- Workflow Operations

Generic Hooks

For Generic hooks, you can use the following COM-based parameters:

```
Function DoSomething ( Session As SmApplic.SmSession,
  ClassId As Integer, _
  Operation As SmApplic.ISmOperation, _
  Stage As SmartServerHook.HookStageEnum, _
  Str As String, _
  FirstPar As SmRecList.SmRecordList, SecondPar As
  SmRecList.SmRecordList, _
  ThirdPar As SmReclis.SmRecordList ) As
  SmartServerHook.ErrorCodeEnum
```

| | |
|---------------------------|---|
| Session As SmSession | The current SmarTeam Session |
| ClassId As Integer | The Class ID under which the hook is activated. For example: <i>Folder</i> . |
| Operation As ISmOperation | The operation when this hook was activated. For example: <i>ADD, UPDATE</i> |
| Stage As HookStageEnum | Stage of the hook: hsAfter, hsBefore, hsInstead |
| Str As String | Name of the hook |
| FirstPar As SmRecordList | Input |
| SecondPar As SmRecordList | Input/Output |
| ThirdPar As SmRecordList | Input/Output |
| ErrorCodeEnum | Function return Error Code |

Library-Specific Hooks

For Library-specific hooks, use exactly the same parameters that you use in Client-Side scripts for each type of script hook.

Example: SmarTeam - Workflow

On a **Before** event:

```
Function OnSendBefore (ActiveProcess As
  SmartFlow.SmActiveProcess,
  Response As SmartFlow.SmResponse ) As
  SmartServerHook.ErrorCodeEnum
```

On a **After** event:

```
Function OnSendAfter (FlowSession As SmartFlow.SmFlowSession, _  
    FlowProcess As SmartFlow.SmFlowProcess, Node As  
    SmartFlow.SmNode, _  
    Response As SmartFlow.SmResponse) As SmartServerHook.ErrorCodeEnum
```

On a **Receive** process:

```
Function OnReceive (FlowSession As SmartFlow.SmFlowSession, _  
    FlowProcess As SmartFlow.SmFlowProcess, Node As  
    SmartFlow.SmNode) _  
    As SmartServerHook.ErrorCodeEnum
```

On a **Task** event:

```
Function OnTaskStart (ActiveProcess as SmActiveProcess , _  
    Task as ISmTask , LinkedObjects as ISmMultiObjects)
```

On **Before Attach an Object**

```
Function BeforeAttachObject (ByVal FlowProcess As  
    SmartFlow.ISmFlowProcess, _  
    ByVal SmObjects As SmApplic.ISmObject, _  
    ByVal LinkObject As SmApplic.ISmObject)
```

On **After Attach an Object**:

```
Function AfterAttachObject (ByVal FlowProcess As  
    SmartFlow.ISmFlowProcess, _  
    ByVal SmObjects As SmApplic.ISmObject, _  
    ByVal LinkObject As SmApplic.ISmObject)
```

On **BeforeDetachObject**

```
Function BeforeDetachObject (ByVal FlowProcess As  
    SmartFlow.ISmFlowProcess, _  
    ByVal SmObjects As SmApplic.ISmObject)
```

On **AfterDetachObject**

```
Function AfterDetachObject (ByVal FlowProcess As  
    SmartFlow.ISmFlowProcess, _  
    ByVal SmObjects As SmApplic.ISmObject)
```

Return a Value On a Script Error

Note: For details about Server Side Scripting Error Codes, refer to the SmarTeam COM API Reference Guide, Library: **SmUtil** section.

Starting with V5R16 you have the functionality to return to a customer an error message when scripts fail.

This is relevant for the Before stage.

For example:

- Before add a new object
- Before send Accept in Workflow

When a server side script fails, an error message "This is a customer error" is sent

There are two ways to do this:

- With reference to SmartWebAppUtils:
- Without reference (not version dependent)

With Reference to SmartWebAppUtils

```
Public Function SendingAccept(ByVal ActiveProcess As
SmActiveProcess, ByVal Response As
SmResponse) As ErrorCodeEnum

Dim iSession As Long
Dim smSession As SmarTeam.Std.Interop.SmarTeam.SmApplic.SmSession
Dim oWebAppUtils As
SmarTeam.Std.Interop.SmarTeam.SmartWebAppUtils.SmWebAppUtils

smSession = ActiveProcess.Session

iSession =
System.Runtime.InteropServices.Marshal.GetComInterfaceForObject(s
mSession,
GetType(SmarTeam.Std.Interop.SmarTeam.SmApplic.ISmSession))

oWebAppUtils =
smSession.GetService("SmartWebAppUtils.SmWebAppUtils")

oWebAppUtils.ErrorHandlingHelper.SetErrorBuffer(iSession,
"SmarTeam", " This is a customer error")

SendingAccept = ErrorCodeEnum.ecGen

End Function
```


Without Reference to SmartWebAppUtils

```
Public Function SendingAccept (ByVal ActiveProcess As
SmActiveProcess, _
    ByVal Response As SmResponse) As ErrorCodeEnum
    Dim iSession As Long
    Dim oWebAppUtils As Object
    iSession = ObjPtr (ActiveProcess.Session)
    Set oWebAppUtils =
CreateObject ("SmartWebAppUtils.SmWebAppUtils")
    oWebAppUtils.ErrorHandlingHelper.SetErrorBuffer iSession,
"SmarTeam", _
        " This is a customer error "
    SendingAccept = ErrorCodeEnum.ecGen
End Function
```

Server-Side Script Naming Convention

SmarterTeam recognizes the Server-Side script for execution in two different ways:

| | By Name | By Event-based Name |
|------------------------|---|--|
| Generic hooks | Any name that is defined in Script Maintenance | Function name format: [Stage]_[OperationName] For example: Before_ADD After_LifeCicle1 |
| Library-Specific hooks | Any name that is defined in the Flow Chart Designer | OnSendBefore OnSendAfter OnReceive |

Packaging Server-Side Scripts into an Interface Object

Server-Side Scripts need to be packaged into a single class that must be registered in the Windows Registry of the server. You can implement this class using a development tool such as VB.Net, C#, Visual Basic or Visual C++ and then compile them to produce a .DLL.

Note: For .NET, register this .DLL with the RegAsm program and use the codebase option:
For example:

```
C:\Windows\Microsoft.NET\Framework\ v2.0.50727\RegAsm.exe  
C:\SmarTeam\bin\SSSLowLevelScripts.dll /codebase
```

Note: For VB6, whenever you create a new COM component (.DLL), after the first compilation of the component you must set the project properties for binary compatibility with the newly created .DLL. Binary compatibility ensures that each time you compile the .DLL you do not generate a new globally unique identifier (GUID) for the object.
Register the .DLL with the RegSvr32 program. For example:

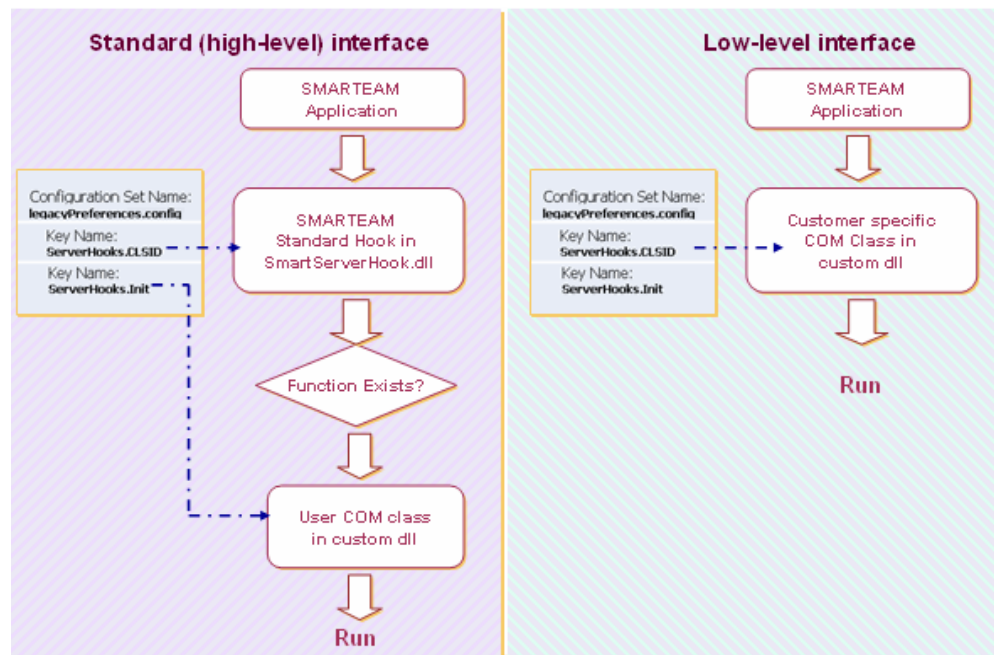
```
C:\Windows\System32\RegSvr32 C:\SmarTeam\bin\SSSLowLevelScripts.dll
```

SmarTeam Hook Interfaces

SmarTeam – Editor provides you with two types of hook interfaces that enable you to use SmarTeam hooks in Server Mode:

- **Standard** (high-level) interface:
 - Simple, easy to use and deploy
 - Similar to SmarTeam Standard script interface
 - Probably adequate for 80 percent of customer scenarios
- **Low-level** interface:
 - More advanced skills required to implement
 - Flexible control – enable/disable hook function execution, determines execution rules of event-based hook functions and SmarTeam hook functions
 - Required when high performance is mandatory or flexible hooks execute exact necessary events

Note: You can implement only one type of SmarTeam Hook Interface for a specified SmarTeam database.

**Standard (high-level) interface**

- Create a COM class
- Supply the hook functions.
- Register the .DLL on the server
- Register the ProgID of this class for this SmarTeam server, using the System Configuration Editor for the setting ServerHooks.Init

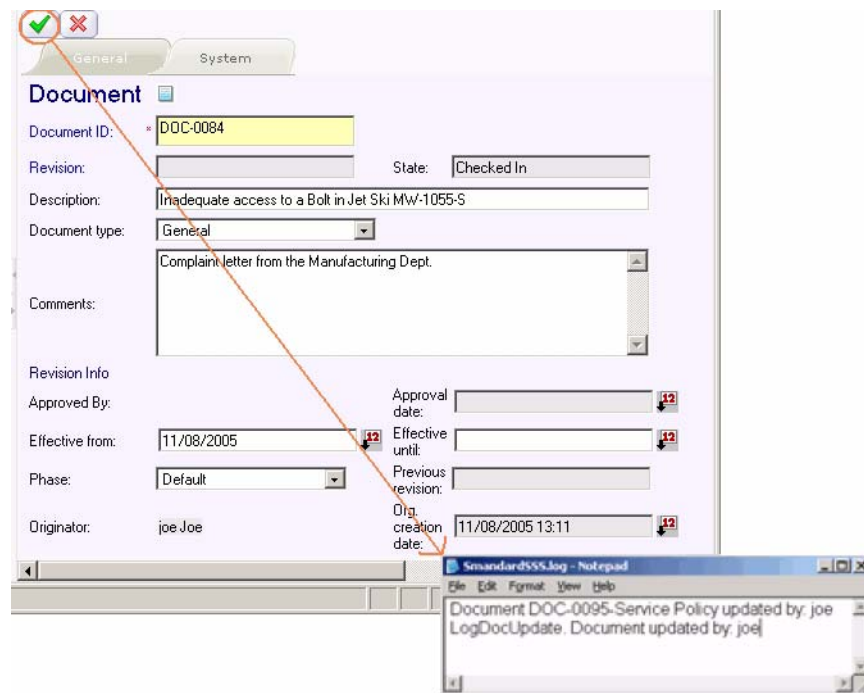
Low-level interface

- Create a COM class that implements the ISmServerHook interface
- Supply the five functions required for this interface
- Register the .DLL on the server
- Register the GUID of the class for this SmarTeam server, using the System Configuration Editor for the setting ServerHooks.CLSID

Note: We recommend that you use the Standard hooks interface. Only when required, use the Low-Level hooks interface.

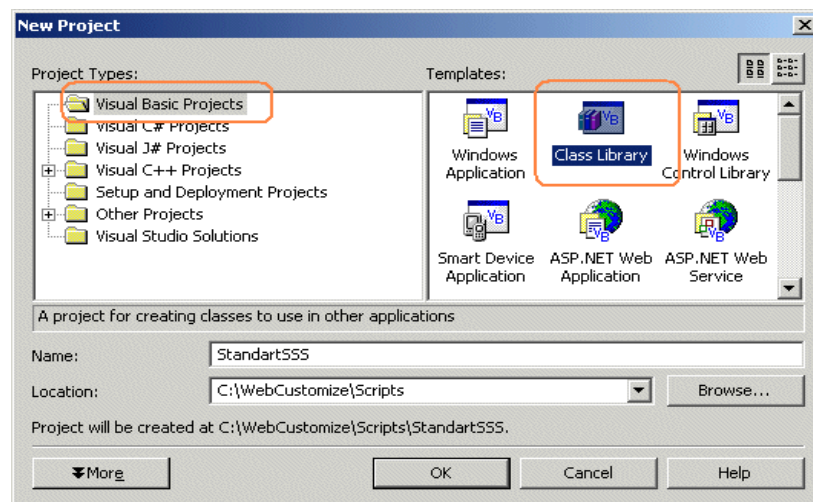
Standard Server-Side Script (By Script Maintenance Name)

Example Scenario 1: After updating a document, write a line of text to a log file.



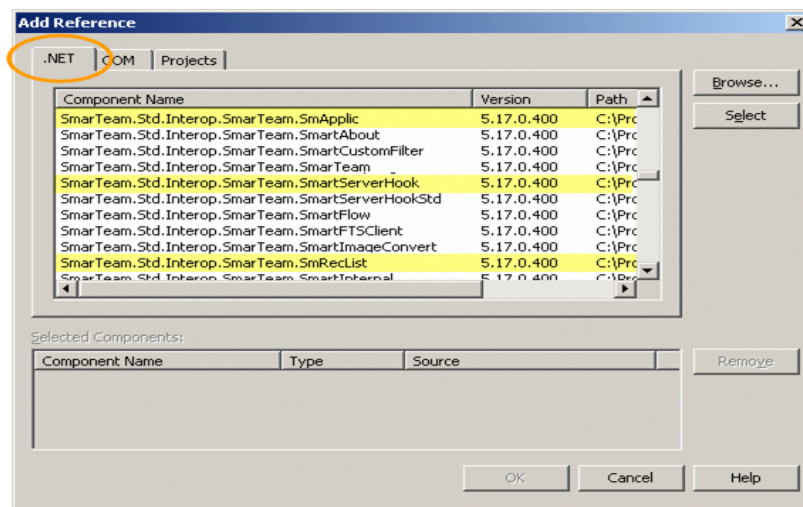
To create a Standard Server-Side script by the Name:

- 1 Open **VB.Net** and create a COM class using a new Class Library.

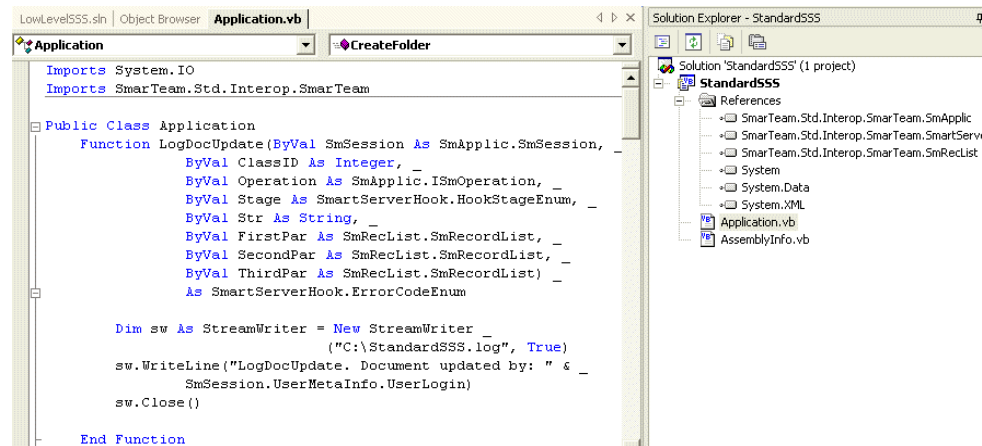


- 2 Add references to the libraries:

- SmarTeam Engine
- SmarTeam Record List
- Smart Server Hook



3 Write the **LogDocUpdate** function:



4 Build **StandardSSS.dll** and register it in Windows:

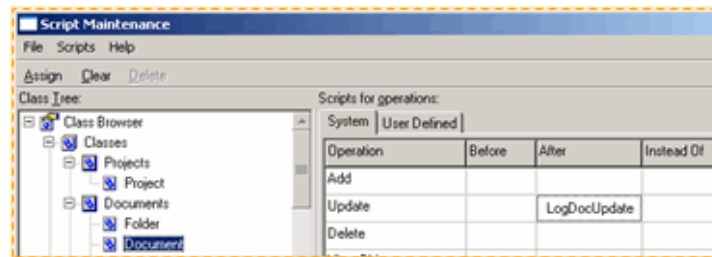
```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe
C:\WebCustomize\Scripts\StandardSSS\bin\StandardSSS.dll /codebase
```

5 Define your script in SmarTeam:

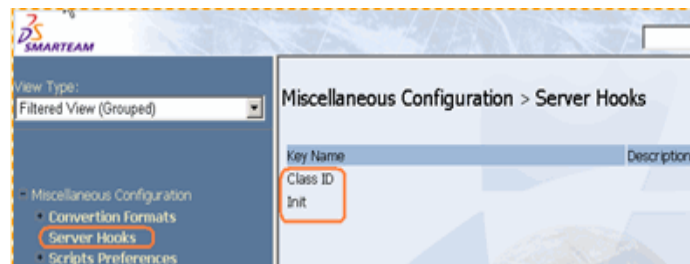
- Write a dummy script **LogDocUpdate** in the Basic Script Editor

```
Function LogDocUpdate ( ApplHndl As Long,Sstr As String,FirstPar As
Long, _
                        SecondPar As Long,ThirdPar As Long ) As Integer
'client side
End Function
```

- Connect the dummy script LogDocUpdate to After Update hook for leaf class Document in SmartBasic Script Maintenance:



- 6 Register your Server-Side Script in SmarTeam System Configuration:



- Do not change key ServerHooks.CLSID:

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

Note: StandardSSS = Name of the .DLL (project). Application = Name of Class.

- Type the .DLL name and Class Name in ServerHooks.Init key:

```
ServerHooks.Init = StandardSSS.Application
```

- 7 Run your SmarTeam Web application:

- Update any document
- Verify that you can see who did it

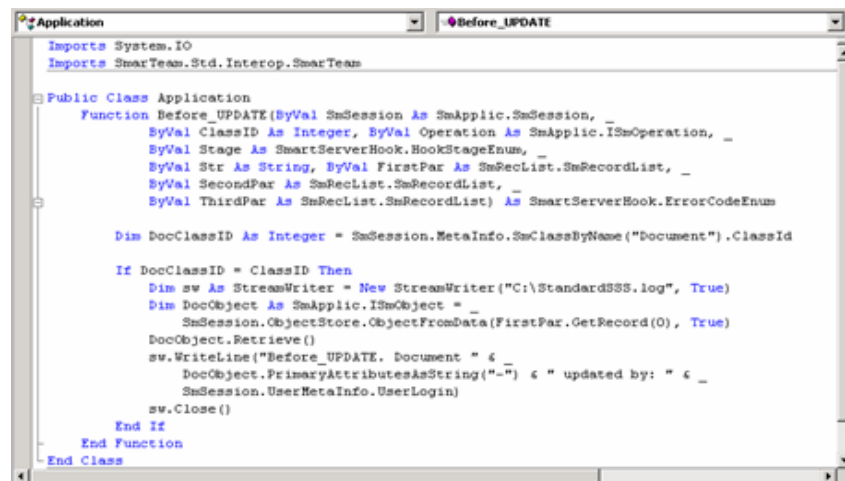
Standard Server-Side Script (By the Event Name)

Example Scenario 2 (same as Scenario 1): After updating a document write a line of text to a log file.

To create a Standard Server-Side script by the Event Name:

- 1 Write a function with the event-based name, i.e., **Before_UPDATE**.

This function runs on each update event so you must filter it to run only for the document leaf class.



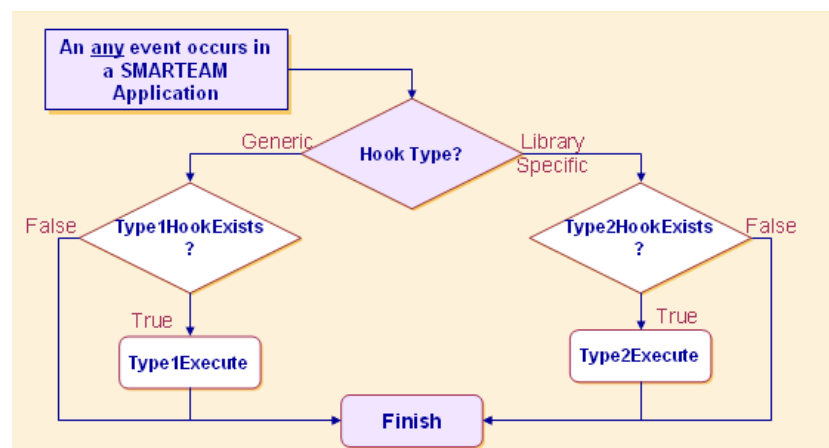
Note: You do not need to register the .DLL again. You only need to register it again when the name of the .DLL or the class name changes.

- 2 Rebuild the solution.
- 3 Test the solution.

Low-Level Server-Side Scripts

Low-Level Server-Side Script methods are divided into Type1 and Type2 and correspond to the two types of hooks:

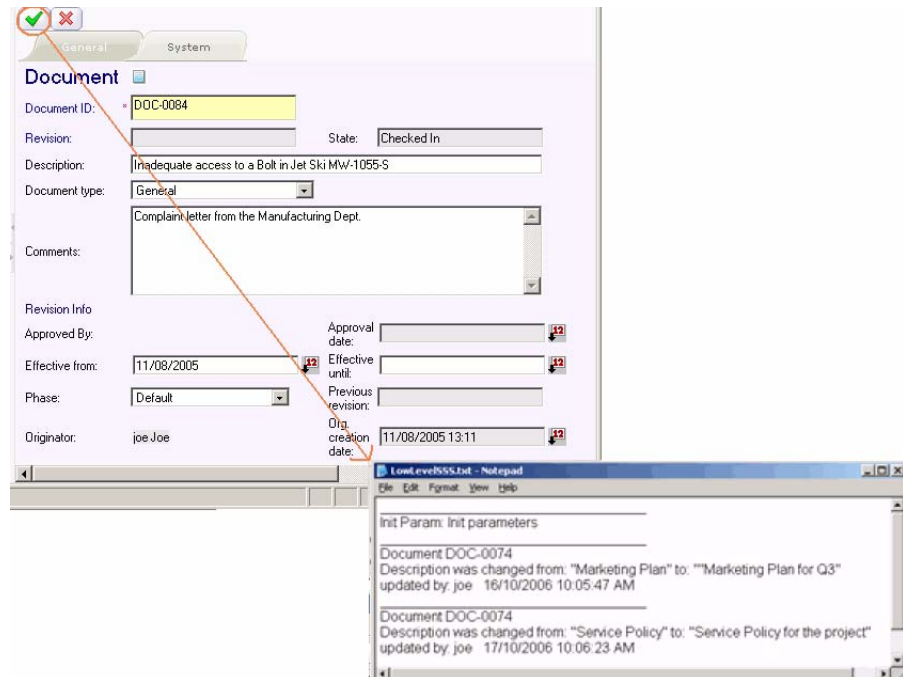
- Type1 hooks – used for generic hooks
- Type2 hooks – used for library-specific hooks



Note: The system runs the Type1HookExists method when any generic hook events occur. If you want to catch this hook, your Type1HookExists method must return **True**. The Type1Execute method runs only in cases that Type1HookExists method returned **True**. The same explanation applies to Type2 Hooks.

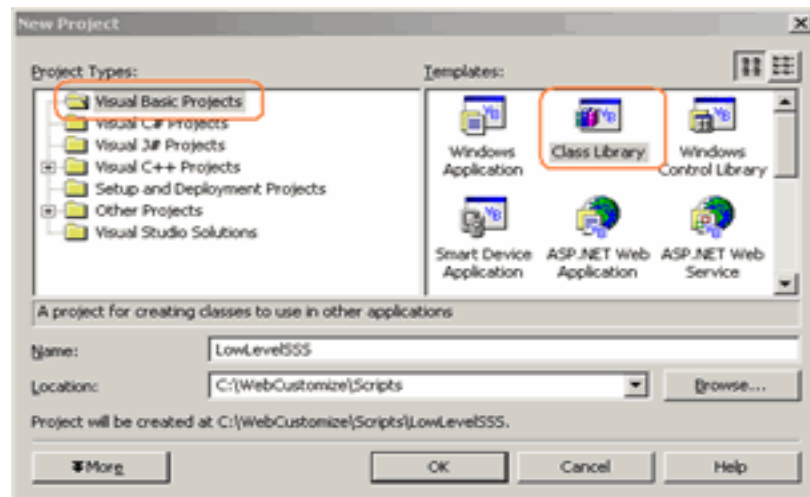
Low-Level Server-Side Script using a TypeHook1 Interface

Example Scenario 3: Create History log of all changes in a document that includes: who modified it, when and what were the differences in the metadata.



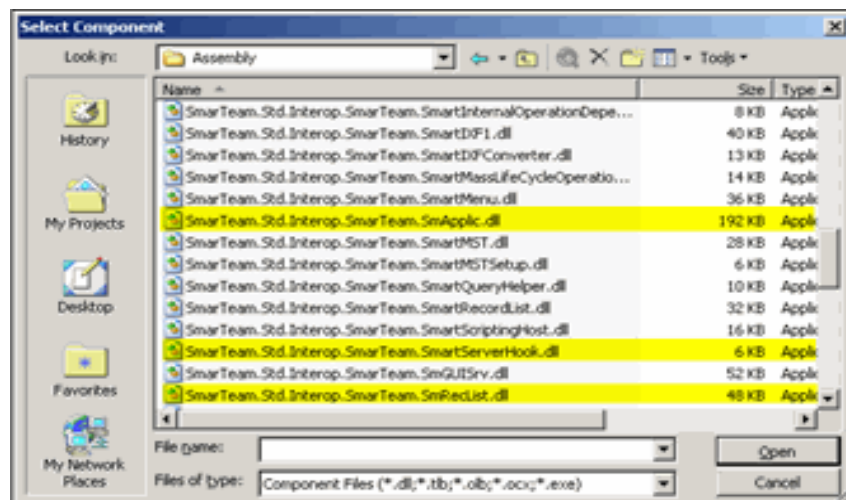
To create a Low-Level Server-Side script (TypeHook1):

- 1 Start a new .DLL and create a new Class Library in VB.Net.

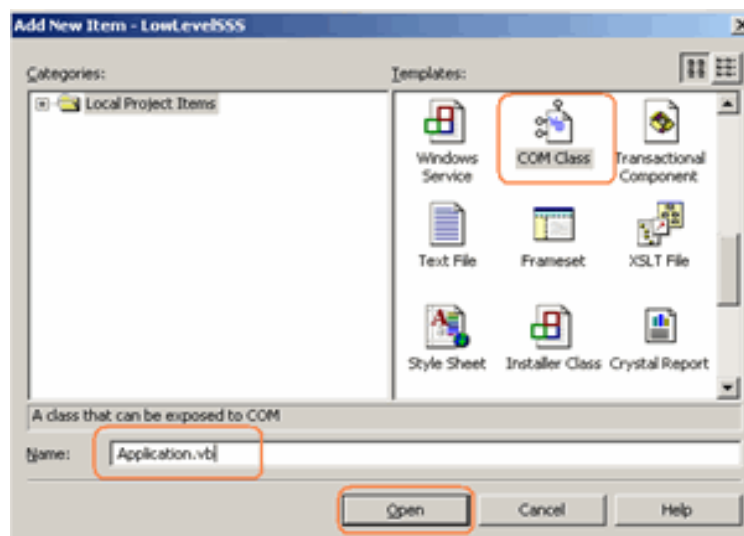


- 2 Add references to the libraries:

- SmarTeam Engine
- SmarTeam Record List
- Smart Server Hook



- 3 Create a new COM Class and rename Class1 that you created to **Application**.



- 4 Implement the **ISmServerHook**:

```
Public Class Application
    Implements ISmServerHook
End Class
```

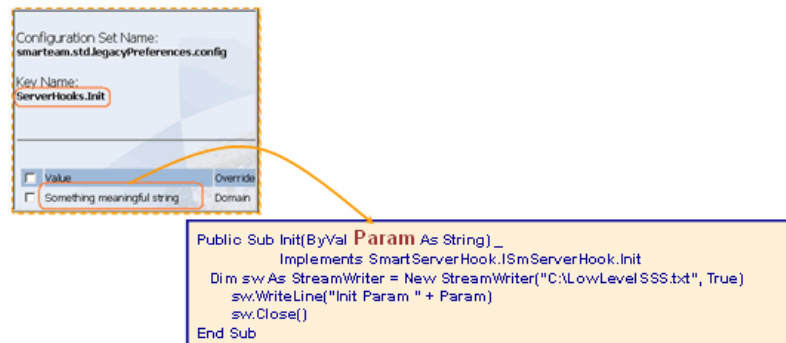
- 5 Define the five control methods:

- Init
- Type1HookExists
- Type1Execute
- Type2HookExists
- Type2Execute

- 6 Implement the **Init** method:

- This function runs an initiation of the custom .DLL
- Parameter(s) are defined in the SmartTeam System Configuration

- This function can remain empty



7 Implement the **Type1HookExists** method:

This function returns True (in our example) only if :

- On an **UPDATE** operation
- On the stage **Before** and **After**
- If the **Type1HookExists** method returns True, the **Type1Execute** method executes

```
Public Function Type1HookExists(...) As Boolean Implements
    ISmServerHook.Type1HookExists
    If ((Stage = HookStageEnum.hsBefore) Or (Stage =
        HookStageEnum.hsAfter)) And _
        Operation.Name = "UPDATE" Then
        Type1HookExists = True
    Else
        Type1HookExists = False
    End If
End Function
```

8 Implement the **Type1Execute** method:

- Run this method (in our example) only on a Document object by checking the ClassID
- Before **UPDATE** save all important metadata in SmarTeam Global Data

- After the update compare the actual metadata with the metadata stored in memory and write the information to a log file

```
Public Function Type1Execute(...) As ErrorCodeEnum Implements
ISmServerHook.Type1Execute

    Dim DocClassID As Integer =
Session.MetaInfo.SmClassByName("Document").ClassId

    ' Run this script only on a Document Object
    If (DocClassID <> ClassId) Then Exit Function

    Select Case Stage

        Case HookStageEnum.hsBefore

            'Save the description in the Global Data Object
        Case HookStageEnum.hsAfter

            'Get the saved document's description from the
Global Data

            ' Write differences info to a log file

            Dim sw As StreamWriter = New StreamWriter(FName,
True)

            sw.WriteLine("Document " &
DocObject.PrimaryAttributesAsString(""))

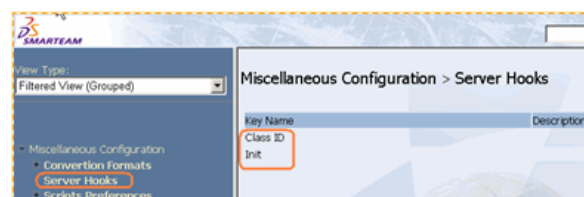
            ...

            sw.Close()

        End Select

    End Function
```

9 Register your Server-Side Script in SmarTeam System Configuration:



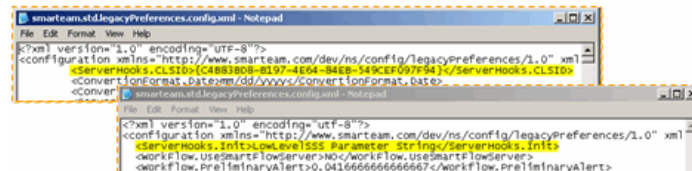
Note: The **Class ID** is the CLSID of the class that implements the ISmServerHook interface. You changed the standard **SmartServerHook.SmDispatch** with your own custom interface:

```
#Region "COM GUIDs"
Public Const ClassId As String = "C4B83BD6-B197-4E64-842E-549CEFD097F94"
Public Const InterfaceId As String = "C2D89148-7426-4436-96F5-BB6AA3E27B2C"
Public Const EventsId As String = "8CE2C80B-DE79-4F41-9BD5-65072F7E7C78"
#End Region
```

Note: The **Init** defines a simple string parameter for the Init function in the **ISmServerHook** interface.

10 Register your .DLL in Windows:

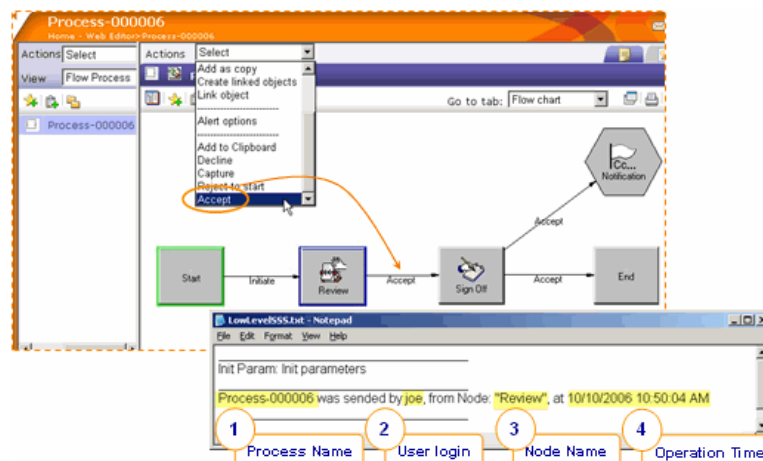
```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe
C:\WebCustomize\Scripts\StandardSSS\bin\SLowLevelSSS.dll
/codebase
```

11 Register your .DLL in SmarTeam System Configuration:**12** Run your SmarTeam Web Application.**13** Test your custom .DLL.

Low-Level Server-Side Script using a TypeHook2 Interface

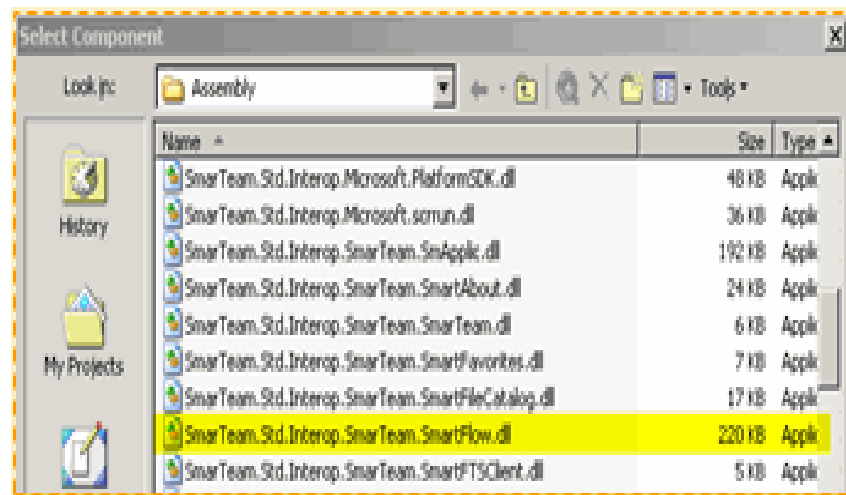
Example Scenario 4: Send a process from one Node saving the information in a log file:

- Process name
- User Login who performed an Accept
- Time of operation



To create a Low-Level Server-Side script (TypeHook2):

- 1 Add an additional reference to the SmartFlow library:



2 Implement the Type2HookExists method:

This function returns True (in our example) only if the event is **OnSendAfter**.

```
Public Function Type2HookExists(...) As Boolean Implements
ISmServerHook.Type2HookExists
    If HookName = "OnSendAfter" Then
        Type2HookExists = True
    Else
        Type2HookExists = False
    End If
End Function
```

3 Implement the Type2Execute method:

Obtain the complete information from the Parameters Record List: **FlowSession**, **FlowProcess**, **Node** and **Response Objects**.

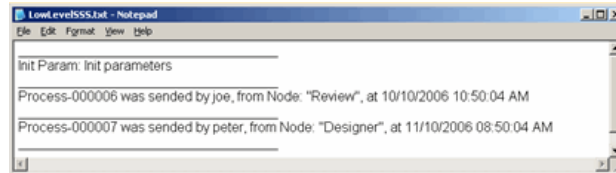
```
Public Function Type2Execute(...) As ErrorCodeEnum Implements
ISmServerHook. Type2Execute

    Dim FlowProcess As ISmFlowProcess =
Parameters.ValueAsObject("FlowProcess")

    Dim Node As ISmNode = Parameters.ValueAsObject("Node")

    Dim sw As StreamWriter = New StreamWriter(FName, True)
    sw.WriteLine(FlowProcess.Name & " was sent by: " &
Session.UserMetaInfo.UserLogin & _
        ", from Node: "" & Node.Description & """,
at " & Now())
    sw.Close()

End Function
```

4 Rebuild your custom .DLL**5** Test the .DLL.

Chapter 10: Visual Components

SmarTeam Visual Components provide you with the building blocks to customize SmarTeam application information and display it the way you require it. SmarTeam – Web Editor enables you to customize the following Visual Components:

- [Profile Card](#)
- [Grid](#)
- [Tree](#)
- [SmartBox](#)

For details, refer to the SmarTeam Visual Components Customization Guide.

Profile Card

The SmarTeam Visual Components enables you to customize the look and feel of the Profile Card. For details, refer to the SmarTeam Visual Components Customization Guide, Profile Card section.

Grid

The SmarTeam Visual Components enables you to customize the dimensions and functionality grids. For details, refer to the SmarTeam Visual Components Customization Guide, Grid section.

Tree

The SmarTeam – Web Editor Visual Components enables you to customize the structure of the Tree. For details, refer to the SmarTeam Visual Components Customization Guide, Tree section.

SmartBox

The SmarTeam Visual Components enables you to customize SmartBox options, e.g., Sending processes. For details, refer to the SmarTeam Visual Components Customization Guide, SmartBox section.

Chapter 11: Customize External Viewers

The new customize external viewers feature enables the customer to create an infinite number of viewers that can be developed and used. All viewers must use the following procedure.

In order to build external viewers, you must build a new viewer adapter and register it to SmarTeam Web Editor via XML file. The adapter must be built with .NET 2.0 framework

To customize External Viewers:

- 1** Create a new .NET 2.0 project.
- 2** Add reference to SmarTeam.std.web.specialized
- 3** Create a new Class.
- 4** Inherit and implement the interface **IWebViewerAdapter**, **IWebViewerAdapter** that has 3 functions you must implement as follows:
 - a** **GetDestinationDirectory()**, this function can return null.
 - b** **PhysicalPathToDocumentID(string root, string path)** - this method gets the path and URL of the file, returns it fixed, and combines with the Viewer URL.

Implement it this way:

```
string URL=Utility.GetViewerTempURL();
```

```
return Utility.PhysicalPathToUrl (path, root, URL);
```

Utility is a viewer class that implements all sorts of methods that helps build the viewer HTML: URL encoding, getpath, etc.

- c** **RenderViewerControl(HtmlTextWriter output, string documentID, string height, string width, string style)**

This method gets the HTMLTextWrite in which all html is built and browsed, the documentID which is a string that contains all information such as, class id, object id, file name, usid. It also contains the height,width, and style of the HTML.

This is the main viewer method, a method that must return Output. That Output is full HTML that will be browsed in the viewer.

You can use all the data in the document id to help build the html.

After you create a new class, inherit and implement **IWebViewerAdapter**, create your XML file.

Viewer.xml is an XML file in the SmarTeam bin directory.

In this file under modules, fill in the assembly name and full class name (including namespace)

The Viewer.xml file should look like this:

```
<?xml version="1.0" encoding="utf-8?>"  
<Modules>
```



```
<Assembly name="C:\Documents and Settings\Administrator\My Documents\Visual Studio
2005\Projects\Smarteam.Std.Web.CostumeViewer\Smarteam.Std.Web.Costume-
Viewer\bin\Debug\Smarteam.Std.Web.CostumeViewer.dll">
<Class name="Smarteam.Std.Web.CostumeViewer.Viewer"></Class>
<Class name="Viewer2"></Class>
</Assembly>
<Assembly name="C:\Documents and Settings\Administrator\My Documents\Visual Studio
2005\Projects\Smarteam.Std.Web.CostumeViewer\Smarteam.Std.Web.Costume-
Viewer\bin\Debug\Smarteam.Std.Web.pdfViewer.dll">
<Class name="Viewer3"></Class>
```

```
</Assembly>
```

```
>/Modules<
```

Under each assembly you need to insert a class name (the class that implements the [IWebViewerAdapter](#)).

All Assembly and Class names must be unique. Each Class/Assembly can appear only once in the XML.

Finally, under configuration system you find the `smarteam.std.viewrs.config.xml`

For each file extension, insert the Class name matching the viewer you have created for this extension.

Please enter a full Class name (including namespace).

The XML should look like this:

```
<adapter>
<extension>.cgr</extension>
<adapter>IBM.New.Viewer.MyClass </adapter>
</adapter>
```

When finished, access the System Configuration Editor and query for the key "UsingExternalViewers" and enter the value "true" (exactly as it is written here), before starting SmarTeam – Web Editor.

Now you can view your files.

Appendix A: Client-Side Scripts Examples

For details about Client-Side scripts see [Chapter 8](#).

Example 1: Use of ListItemsCollection for DropDownList control

This example demonstrates the use of **ListItemsCollection**. **ListItemsCollection** is a collection that is available only for the **DropDownList** control. Only this control (accept **RefToClass**) has several items related to it. **RefToClass** does not expose this collection because RefToClass can only get the ListItemsCollection from the server. Currently, this action is not possible

```
function ListItemsCollectionSample(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    var i;
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            var listItemCollection = controls[i].GetListItemCollection();
            if (listItemCollection != undefined)
            {
                // Clear all the items in the collection
                listItemCollection.ClearAllItems();
                // add several items
                var newListItem = new ListItem("1","First");
                listItemCollection.AddItem(newListItem);
                newListItem = new ListItem("2","Second");
                listItemCollection.AddItem(newListItem);
                newListItem = new ListItem("3","Third");
                listItemCollection.AddItem(newListItem);
                newListItem = new ListItem("4","Fourth");
                listItemCollection.AddItem(newListItem);
                // Remove item in index 2
                listItemCollection.RemoveItem(2);
                // Print all items
                var j;
                var allItemsDesc = 'Items of attribute: ' +
controls[i].GetAttributeName() + ' - \n\n';
                for (j= 0; j < listItemCollection.GetLength(); j++)
                {
                    allItemsDesc += "value: " +
listItemCollection.GetItem(j).Value + " text: " +
listItemCollection.GetItem(j).Text + "\n";
                }
                alert(allItemsDesc);
            }
        }
    }
}
```

Example 2: Cycles through Profile Card Controls/Shows the Details

This script cycles between all the controls in the profile card and shows the details.

```
function GetControlsDetails(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    var controlsDetails;
    var controlDetails;
    controlsDetails = "Profile card control details: \n\n";
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controlDetails = "Attribute name: " + controls[i].GetAttributeName() +
            " | ";
            controlDetails += "Control type: " + controls[i].GetControlType() + " | ";
            controlDetails += "Value: " + controls[i].GetValue() + " | ";
            controlDetails += "Text: " + controls[i].GetText() + " | ";
            controlDetails += "Background: " + controls[i].GetBackground() + " | ";
            controlDetails += "Font: " + controls[i].GetFontFamily() + " | ";
            controlDetails += "ControlId: " + controls[i].GetControlId();
            controlDetails += "IsVisible: " + controls[i].IsVisible();
            controlDetails += "\n";
            controlsDetails += controlDetails;
        }
    }
    alert(controlsDetails);
}
```

Example 3: Shows Profile Card Mode

This script shows the profile card mode, such as show object, update or query mode.

```
function GetProfileCardDetails(ProfileCardClientObj, ProfileCardObj)
{
    var profileCardDetails;
    profileCardDetails = "Profile card control details: \n\n";
    profileCardDetails += "Mode: " + ProfileCardObj.GetMode();
    alert(profileCardDetails);
}
```

Example 4: Hides Profile Card Controls

This script hides controls in the profile card.

```
function SetVisibleFalse(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetVisibility("hidden");
        }
    }
}
```

Example 5: Shows Controls that were Hidden

This script shows the controls that were hidden.

```
function SetVisibleTrue(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetVisibility("visible");
        }
    }
}
```

```
function SetVisibleTrue(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();

    for (i = 0; i < controls.length; i++)
    {
```

Example 6: Retrieves Current SmarTeam Object Data

This script retrieves the current SmarTeam object's data.

```
function GetSmObjectValues(ProfileCardClientObj, ProfileCardObj)
{
    var SmObjectData = ProfileCardObj.GetSmObjectData();

    var smObjectDetails = "sm object of profile card:";
    smObjectDetails += "\n ObjectId: " + SmObjectData.GetObjectId();
    smObjectDetails += "\n ClassId: " + SmObjectData.GetClassId();
    smObjectDetails += "\n ClassName: " + SmObjectData.GetClassName();
    smObjectDetails += "\n Link objectId: " +
        SmObjectData.GetLinkObjectId();
    smObjectDetails += "\n LinkClassId: " +
        SmObjectData.GetLinkClassId();
    smObjectDetails += "\n LinkClassName: " +
        SmObjectData.GetLinkClassName();
    smObjectDetails += "\n Node object id: " +
        SmObjectData.GetNodeObjectId();
    smObjectDetails += "\n Node class id: " +
        SmObjectData.GetNodeClassId();
    smObjectDetails += "\n Node name: " + SmObjectData.GetNodeName();
    alert(smObjectDetails);
}
```

Example 7: Changes Value of Profile Card Controls

This script changes the value of the controls in the profile card.

```
function PC_SetValue(ProfileCardClientObj, ProfileCardObj)
{
    alert ("setting values to '1'");
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetValue("1");
        }
    }
}
```

Example 8: Changes Text in Profile Card Controls

This script changes the text shown in the controls in the profile card.

```
function PC_SetText(ProfileCardClientObj, ProfileCardObj)
{
    alert ("setting text to '1'");
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetText("1");
        }
    }
}
```

Example 9: Changes Background Color of Profile Card Controls

This script changes the background color of controls in the profile card.

```
function SetBackgroundGreen(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetBackground("green");
        }
    }
}
```

Example 10: Disables Profile Card Controls

This script disables the controls (makes the controls read only)

This script enables the controls (makes the controls read/write).

```
function SetDisabledTrue(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetDisabled(false);
        }
    }
}
```

Example 11: Enables Profile Card Controls

This script disables the controls (makes the controls read only)

This script enables the controls (makes the controls read/write).

```
function SetDisabledFalse(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetDisabled(false);
        }
    }
}
```

Example 12: Changes the Fonts of Profile Card Controls

This script changes the fonts of the controls.

```
function SetFontFamilyArial(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetFontFamily("Arial");
        }
    }
}

function SetFontFamilySansSerif(ProfileCardClientObj, ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controls[i].SetFontFamily("MS Sans Serif");
        }
    }
}
```

Example 13: Called Before Loading Profile Card

This script is called before the profile card is loaded.

```
function ProfileCardLoad(ProfileCardObj)
{
    var controls = ProfileCardObj.GetControls();
    var controlsDetails;
    var controlDetails;
    controlsDetails = "Profile card control details: \n\n";
    for (i = 0; i < controls.length; i++)
    {
        if (controls[i].GetAttributeName() != undefined)
        {
            controlDetails = "Attribute name: " +
controls[i].GetAttributeName()
            + " | ";
            controlDetails += "Control type: " +
controls[i].GetControlType()
            + " | ";
            controlDetails += "Value: " + controls[i].GetValue() + " | ";
            controlDetails += "Background: " + controls[i].GetBackground()
            + " | ";
            controlDetails += "Font: " + controls[i].GetFontFamily() + " |
";
            controlDetails += "ControlId: " + controls[i].GetControlId();
            controlDetails += "IsVisible: " + controls[i].IsVisible();
            controlDetails += "\n";
            controlsDetails += controlDetails;
        }
    }
    alert(controlsDetails);
}
```

For more information, refer to the Web Form Designer in the SmarTeam - Editor Online Help.