CA Mainframe Network Management

WebCenter SDK Developer Guide

r12

CA Product References

This document references the following CA products:

- CA Mainframe Network Management
- CA NetMaster® Network Management for TCP/IP (CA NetMaster NM for TCP/IP)
- CA NetMaster® File Transfer Management (CA NetMaster FTM)
- CA ACF2™ for z/OS
- CA Top Secret® for z/OS

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short customer survey, which is also available on the CA Support website, found at http://ca.com/docs.

Contents

Chapter 1: Introduction

Using WebCenter SDK, you can develop your own web pages and add these pages into the distributed WebCenter menu structure. You can use these web pages to call user NCL procedures to provide data for the web page.

WebCenter SDK includes a collection of HTML fragment files that allow users to develop web pages with the WebCenter look-and-feel with a minimum of coding effort.

This section contains the following topics:

Product Region Requirements

For WebCenter SDK to be activated, your region must be at a currently-supported release, and the region must be licensed for at least one WebCenter-enabled product.

Scope of this Guide

It is assumed that if you are reading this document, then you already know enough HTML, JavaScript, and CSS language syntax to code a web page. This document only describes those features of WebCenter SDK that enable you to build web pages that interact with CA Mainframe Network Management products, and have the look-and-feel of other WebCenter web pages.

These features include:

- Menu registration files
- $W3MH01X web menu user exit
- HTML fragments
- WebCenter SDK JavaScript variables
- WebCenter SDK JavaScript functions
- WebCenter SDK CSS style classes
- ESP directives

Chapter 2: Implementation Overview

This section contains the following topics:

Pre-Implementation Planning

Successfully implementing WebCenter SDK requires a variety of skills and is a team effort. WebCenter SDK requires technical skills in areas that the traditional mainframe skill-set of this product does not otherwise use; therefore, make sure to enlist assistance from someone with a programming experience in web-based languages.

Do not attempt to implement WebCenter SDK until you have arranged access to people with the following skills:

- z/OS UNIX System Services and the hierarchical file system (HFS)—to set up a functional HFS and UNIX environment
- Security server (CA ACF2 for z/OS, CA Top Secret for z/OS, or RACF), the security product used to authorize access to resources
- SMP/E and JCL

WebCenter SDK Software Components

The following components are required:

- **CA Mainframe Network Management Region**—The CA Mainframe Network Management region (or product region) provides WebCenter SDK web services and access to your web pages.
- **HFS File System**—HFS format files are standard for Java and UNIX System Services. They are physically stored in VSAM files, which are 'mounted' to correspond to a certain directory path. They contain the distributed WebCenter SDK sample web pages.

WebCenter SDK Planning for Security Administrators

WebCenter SDK implementation will require security to be granted to a WebCenter user ID in the following areas.

- Security system access to USS/OMVS
- USS access to HFS files

More details are provided in the implementation task descriptions.

Disk Space Required

Ensure that you have sufficient disk space for the installation. On a 3390 DASD, the required space is:

- Installation = 5 cylinders
- SMP/E temporary libraries = 1 cylinder
- HFS = 14 cylinders

Software Products Required

You must have one of the following products before you can install WebCenter SDK:

- CA NetMaster NM for TCP/IP
- CA NetMaster FTM

After installing your CA Mainframe Network Management product, you can then install the additional WebCenter SDK feature.

Expertise and Personnel Required

The WebCenter SDK implementation process consists of a sequence of manual activities, SMP and non-SMP job streams, and USS operations.

To implement WebCenter SDK, you will need the expertise of staff in these areas:

- z/OS systems programming
- CA Mainframe Network Management systems programming and administration
- UNIX System Services administration
- CA ACF2 for z/OS, CA Top Secret for z/OS, or RACF security administration

For technical assistance, contact Technical Support.

Implementation Steps

To implement WebCenter SDK, you must perform the following steps:

1. Install WebCenter SDK.

   Use the product's installation facility to generate the JCL to perform the SMP/E installation of WebCenter SDK. You must define and mount an HFS file system.

2. Add WebCenter SDK to the required product region by repeating the setup for the region.

   Use the installation facility to generate the JCL to set up the region.

3. Configure your WebCenter SDK region.

More information:

Installation Overview (see page 17)
Region Setup Overview (see page 21)
Updating and Actioning the $NM WebCenter Parameter Group (see page 25)

Chapter 3: Installing WebCenter SDK

This section describes how to install WebCenter SDK HFS code. The WebCenter SDK code resides in USS HFS directories.

This section contains the following topics:

Preparing for WebCenter SDK Installation

The following sections describe what you need to do before you start installing WebCenter SDK.

Install the CA Mainframe Network Management Product and Set Up the Region

Before installing the WebCenter SDK HFS code, you must have already done the following:

- Completed CA Mainframe Network Management product installation
- Decided which product regions will include WebCenter SDK
- Completed setup and customization of those regions

Ensure that you have completed the tasks in the *Installation Guide* for your product.

Apply CA Mainframe Network Management Product Maintenance

You install WebCenter SDK from the same source as your CA Mainframe Network Management product so that WebCenter SDK and your product are at the same service pack maintenance level.

If you plan to use more recent source for the WebCenter SDK installation, you must apply all maintenance from the same source to your CA Mainframe Network Management products before installing WebCenter SDK. For information about maintenance for a particular product, see the *Installation Guide* for that product.

Provide an HFS directory for WebCenter SDK Use

WebCenter SDK requires an HFS. During installation, you must specify the name of an HFS path. This path is where SMP/E installs the WebCenter SDK code.

Depending on your organization's requirements, you may install the WebCenter SDK code into an existing HFS or a new HFS. The HFS path name must not include quotes and spaces.

Ask your UNIX System Services administrator to implement an HFS for WebCenter SDK and let you know the path name that is the mount point.

More information:

HFS Considerations (see page 105)

HFS Requirements and Security Settings

To check your security access:

- Ensure that you have UPDATE authority to the UNIX System Services hierarchical file system (HFS).
- If you want to mount a new HFS, you may need to update the SYS1.PARMLIB(BPXPR*xx*) initialization parameter data set members.

Verify UNIX System Services Security Requirements

The user ID installing WebCenter SDK must have authorization to use UNIX System Services. The installation process includes creating and populating HFS directories.

Ask your security administrator to define UNIX System Services authorization for the SMP/E program name and the TSO user ID of the user who will submit the installation JCL.

More information:

Authorities Required (see page 108)

Installation Overview

The installation process is done with *one* of the following:

CA Mainframe Software Manager

This manager provides a web interface that simplifies and unifies the management of CA mainframe products on z/OS systems. As other CA products adopt this level of standardization, you can manage them in a common way.

If you choose to use this method, complete the required tasks in the *CA Mainframe Software Manager Product Guide* and then continue with the section Setting up Your Region to Include WebCenter SDK (see page 21).

Install Utility

This utility installs the product into an IBM System Modification Program Extended (SMP/E) environment. The utility collects your site-specific values such as data set prefixes, DASD volume serial numbers, and JCL parameter values. It then uses these values to generate the jobs necessary to perform the installation of your product.

If you choose to use this method, see the section Preparing for WebCenter SDK Installation (see page 15).

Generating the Installation JCL

For WebCenter SDK installation, you must provide the HFS directory path name (see page 16). The Install Utility uses this information to generate the WebCenter SDK installation JCL.

To generate the WebCenter SDK installation JCL, follow these steps:

1. At the ISPF/PDF TSO command prompt, execute the following command:

   ```
   EXEC 'dsnpref.NMC0.CAIJCL(INSTALL)'
   ```

   The Install Utility panel opens.

   On each of the Install Utility panels, you can press:

   - ENTER to proceed to the next panel
   - F1 to display help
   - F3 to return to the previous panel
   - F4 to exit and return to the main menu

2. Press Enter.

   The Install Utility Primary Menu panel appears.

3. Complete the following panels as they open. Press Enter at the completion of each panel. You must complete all panels before you can setup the product. You can take the default options or specify site-specific values.
4. Enter **2** (Install Products).

   The INSTALLATION Primary Menu panel appears.
5. Enter **1** (Select Products to Install).

   The INSTALLATION Product Selection panel appears.
6. Enter **S** beside WebCenter SDK, and press Enter.

   The INSTALLATION Product Confirmation panel appears to confirm your selections.
7. Press Enter, and complete each of the INSTALLATION panels as they are displayed. For information about the fields press F1 (Help).

   The various INSTALLATION panels ask you for the following, among other things:

   - An HFS path name for the WebCenter SDK directory on the INSTALLATION WebCenter SDK HFS Information panel. This directory is used as the SMP/E target library for WebCenter SDK.
   - A name for the Installation JCL data set on the INSTALLATION JCL Library Creation panel. This is the data set in which the generated installation jobs are placed. If you installed from tape or with ESD, the default name is *dsnpref*.NMC0.INSTALL.JCL. If you installed with MSM, the default name is *dsnpref*.CAIJCL.JCL. To ensure that WebCenter SDK has its own dedicated installation JCL data set, you should change this to another name, such as *dsnpref*.NMC0.INSTALL.JCL.WCSDK, and make note of this new name.
8. When you get to the INSTALLATION JCL Generation - Confirmation panel, press Enter to start the JCL generation. It may take a few minutes to generate all the jobs.

Submitting the WebCenter SDK Installation JCL

Important! Do not proceed with this step until the WebCenter SDK HFS directory is correctly implemented and permanently mounted. If it is not, your installation jobs will fail. Run the installation jobs on a system that has this HFS mounted for read/write access.

After the Install Utility generates the installation JCL procedures, submit them as follows:

1. Run the WebCenter SDK jobs from the WebCenter SDK Installation JCL data set in the following order. Do not proceed with any job until the previous job has completed successfully. Each job returns condition code 0 when successfully completed, unless otherwise indicated.

   I21ALLWS

   The SMP/E data sets are allocated.

   I22INIWS

   The SMP/E data sets are initialized.

   I23RECWS

   WebCenter SDK is SMP/E received. This job requires tape mounts if installing from tape.

   I26APPWS

   WebCenter SDK is SMP/E applied. This job populates the HFS target library (TLIB).

   I27ACCWS

   WebCenter SDK is SMP/E accepted. The SMP/E distribution library (DLIB) for the HFS target library is a z/OS data set.

2. Press F3 repeatedly to exit the Install Utility.

Verifying the WebCenter SDK Installation

After successful installation, you can optionally look at the installed files in the WebCenter HFS directory. Use the standard TSO OMVS commands to list the directory contents.

Chapter 4: Setting up Your Region to Include WebCenter SDK

This section contains the following topics:

Region Setup Overview

This chapter describes how to make the WebCenter SDK HFS code that you previously installed (see page 17) accessible to a region.

Setting up your region to include WebCenter SDK is done with the Install Utility. The Setup a Product Region option of this utility collects region-specific values and uses these values to:

- Generate the jobs that add the WebCenter SDK feature to the existing region.
- Identify the WebCenter SDK HFS directory to the region.

Generating the Region Setup JCL

To generate the region Setup JCL to include WebCenter SDK, follow these steps:

1. At the ISPF/PDF TSO command prompt, execute the command:

   `EXEC '`*dsnpref*`.NMC0.CAIJCL(INSTALL)'`

   where *dsnpref*.NMC0.CAIJCL is same utility that you used to set up your CA Mainframe Network Management product.

2. Press Enter.

   The Install Utility Primary Menu panel is displayed.

3. (Optional) If you have installed the product using CA MSM, perform the following steps:

   a. Enter **1**.

      The Software Delivery Method panel appears.

   b. Complete the panel:

      - Enter **S** next to CA MSM.
      - Specify the name of the CSI data set used during product installation in the SMP/E CSI Used field.

   c. Press Enter.

4. Complete each of the panels as they open. Press Enter at the completion of each panel. You must complete all panels before you can setup the product. You can take the default options or specify site-specific values.

5. From the Install Utility Primary Menu panel, enter **5** (Setup a Product Region).

   The SETUP Product Region Primary Menu panel is displayed.

6. Enter **4** (Add products and/or additional features to a region).

   The SETUP Specify Product Region Source panel opens.

7. Enter **S** beside the region where you want to include WebCenter.

8. On the SETUP Additional Products Selection panel, enter **S** beside WebCenter SDK.

   Important! Do not set up any other products or features at the same time as WebCenter SDK; do these separately.

9. Complete each of the SETUP panels as they are displayed. For information about the fields press F1 (Help).

   The SETUP JCL Library Creation panel prompts you for a name for the Setup JCL data set. This is the data set in which the generated setup jobs are placed. The default name is *dsnpref*.NMC0.*rname*.JCL, where *rname* is the region name. To ensure that WebCenter SDK has its own dedicated setup JCL data set, you should change this to another name, such as *dsnpref*.NMC0.*rname*.JCL.WCSDK, and make note of this new name.

10. When you get to the SETUP JCL Library Generation panel, press Enter to start the JCL generation. It may take a minute or so to generate all the jobs.

Submitting the Region Setup JCL

Your Setup JCL data set contains the following jobs:

- S01LCALC—The region-specific (local) data sets are allocated.
- S02SHALC—The shared runtime data sets are allocated.
- S03LDVIP—The shared runtime data sets are loaded.
- S04LDVSM—The local VSAM data sets are loaded from the product-specific sequential data sets.
- S05LDPDS—The PDS members are copied to *dsnpref*.*rname*.TESTEXEC and *dsnpref.*PARMLIB for use by the product region.

  Note: The member name for IIAPARMS includes the domain ID, so appears as II*Admid*.
- S06MIGRT—Any site specific VSAM data from earlier releases is copied.

Setting up the region to include WebCenter SDK is unlike setting up this region the first time, because instead of creating a brand new region, you are adding to an existing one. WebCenter SDK requires no additional local or runtime data sets, or VSAM data. Therefore, although the setup software has generated the jobs, most of them are dummy jobs and do not need to be run.

Perform the following steps to submit the region setup JCL:

1. Submit and run only the S05LDPDS job. The PDS members are copied to *dsnpref*.*rname*.TESTEXEC and *dsnpref*.PARMLIB for use by the product region. Because this feature is being added to an existing region, the RUNSYSIN and IIAPARMS are overwritten.

   Note: The S05LDPDS job should complete with a condition code of 0.
2. Press F3 repeatedly to exit the Install Utility.

Verifying the Region Setup

After successful setup, verify it as follows:

1. If you moved the RUNSYSIN member to a more secure data set when you set up your CA Mainframe Network Management product region, ensure you move this new RUNSYSIN to the secure data set.
2. Shutdown and restart your product region.
3. Log on to the region.
4. Enter *CMD* to display the Command Entry panel.
5. Enter *ST*. Verify that the following message is returned:

   ```
   N11414 SYSTEM CONFIGURED WITH WEBCENTERSDK FEATURE - R6.5
   ```

Chapter 5: Customizing WebCenter SDK

You are now ready to customize the region and activate its WebCenter SDK functions.

This section contains the following topics:

Updating and Actioning the $NM WebCenter Parameter Group

After the region has initialized, log on and type /PARMS to access the Customizer : Parameter Groups selection list. Enter U next to the $NM WEBCENTER parameter group in the INTERFACES category to update it. Press Enter to display the following panel.

```
PROD-------------------- Customizer : Parameter Group -------------Page 1 of 2
Command ===>                                                  Function=Update

 WEBCENTER - WebCenter Web Interface

 Web Interface Port .............. 1234_     (NONE, 1 to 65535)
 Access URL ...................... http://101.102.103.104:1234
 Access URL Host Override ........
 User Timeout ..................... 04.00     (hh.mm)
 Enable Public IP Summary Page ... YES       (YES or NO)
 Enable SYSVIEW Interface ........ YES       (YES or NO)
 WebCenter SDK Menu Title ........ User functions
 WebCenter SDK Directory .........
          /iia-prefix/nm/sdk/ws<deliverylevel>/samples

          ____________________________________________________________
          ____________________________________________________________



 Notes
 Specify the WebCenter web interface parameters on this page.

 F1=Help      F2=Split     F3=File      F4=Save      F5=ILog      F6=Action
                           F9=Swap                               F12=Cancel
```

Perform the following steps to complete this panel:

1. Enter a name for your user menu in the WebCenter SDK Menu Title field. This is the submenu label that opens on the WebCenter menu tree. If you do not provide a title, the menu title defaults to User Functions. Alternatively, you could provide the file name for a GIF or JPEG whose image you want to appear on the menu instead of a title.

   If you specified the HFS directory containing the shipped sample web files for your WebCenter SDK Directory, then put /user/WebCenterSDK.jpg in the menu title field.

2. Enter the name of the HFS root directory containing your web files in the WebCenter SDK Directory field (for example, /*myuserid*/mywebfiles).

   When you installed the WebCenter SDK additional feature, the shipped sample web files were saved in the HFS directory *iia-prefix*/nm/sdk/ws<deliverylevel>/samples, where *iia-prefix* is the HFS path name specified during installation. The default WebCenter SDK Directory field value is *iia-prefix*/nm/sdk/ws<deliverylevel>/samples.

   Note: This HFS directory is henceforth referenced by the pseudonym /user/.

3. Press F6 to action the parameter group, and then press F3 to file your changes.

If the user directory you specified contains a valid menu.xml file containing at least one valid menu item definition, you should see your WebCenter SDK user menu the next time you log on to a CA Mainframe Network Management product using the WebCenter interface.

All files in the specified directory can now be accessed using the /user/ prefix instead of their full HFS directory prefix. For example, if the user directory you specified is /*myuserid*/mywebfiles, then the web page /*myuserid*/mywebfiles/mypage.esp can be accessed by specifying /user/mypage.esp. Similarly, any files in a subdirectory can be accessed by specifying the subdirectory after the /user/ prefix. For example, if the user directory you specified is /netmaster/webcenter/userfiles, then the web page /netmaster/webcenter/userfiles/ops/activejobs.esp can be accessed by specifying /user/ops/activejobs.esp.

More information:

Implementation Steps (see page 13)

Chapter 6: Developing ESP Web Pages for WebCenter

The majority of the web pages developed for WebCenter are Executable Server Pages (ESPs) named with the .esp suffix. ESP is the WebCenter proprietary technology that lets a web page interface with CA Mainframe Network Management products. If you want your web page to display useful CA Mainframe Network Management data, or any other data sourced from an NCL procedure, then your web page must be suffixed with .esp.

Important! After developing a new web page and copying it to your WebCenter SDK HFS directory, log on to your CA Mainframe Network Management product and from the Command Entry screen, issue the command WEB RESET CACHE=ALL to ensure that the new copy of the web page will be served up to your web browser.

This section contains the following topics:

Anatomy of an ESP

An ESP may contain a mixture of ESP directives, included HTML fragments, HTML, JavaScript, and Cascading Style Sheet (CSS) definitions. When an ESP is fetched by the web server, it goes through the following processing:

1. All <!--#Include … --> directives are resolved by the include processor, fetching the included HTML fragments as required, resulting in source code where all include directives are converted to their fully-expanded source code.
2. The web page is then parsed with the ESP processor, executing all <% … %> ESP directives that could possibly result in the execution of an NCL or Object Management Language (OML) procedure. The ESP directives are removed from the source at this point and can be replaced with ESP-generated HTML or JavaScript source.
3. The web page, which now only contains HTML, JavaScript, and possibly CSS statements, is then sent to your web browser.
4. The web browser, as it renders the page, processes any in-line JavaScript statements. This could potentially create more HTML to be rendered by the browser.
5. The web page is delivered and the HTTP socket connection is closed.

Object Management Language

This document makes several references to the Object Management Language (OML). OML is an internal development language and is not available for you to write. OML is mentioned in this document because the web server is written in OML and, therefore, your web pages have OML variables available to it. Additionally, the OML programs distributed with your CA Mainframe Network Management product can be called from WebCenter SDK.

Appendix A: Sample Web Files

This section contains the following topics:

Sample Web Files

WebCenter SDK is shipped with the following sample web files. You should copy these files to your development directory, where you can modify them into more useful web pages and rename them accordingly.

blank.esp

This sample web file is a blank page complete with Select, Criteria, and Results sections. The Select section has six action tabs that can be modified as required.

Use this page as the basis for most of the pages that you develop.

You can access this page using the Blank page 1 option on the sample WebCenter SDK menu.

blankDialog.esp

This sample web file is a model for a dialog web page that should be opened in a new window. It consists of a Select section and a Results section, and you can also add a Criteria section. In addition, it has a Close button at the bottom of the Results section to close the window.

You can access this page using the Blank page 5 option on the sample WebCenter SDK menu.

blankNoActTabs.esp

This sample web file is the same as blank.esp except no actions are defined and, therefore, no action tabs are displayed.

You can access this page using the Blank page 3 option on the sample WebCenter SDK menu.

blankNoActTabsOrCriteria.esp

This sample web file is a model Blank page with only a Select section and a Results section. The Criteria section is replaced by the Execute button.

This page is useful if you are designing a web page for an action that has no parameters because action parameters are usually coded as fields in the Criteria section.

You can access this page using the Blank page 4 option on the sample WebCenter SDK menu.

blankNoCriteria.esp

This sample web file is a model Blank page with only a Select section and a Results section. The Select section has six actions defined, therefore, it displays action tabs. The Criteria section is replaced by the Execute button.

This page is useful if you are designing a web page for an action that has no parameters because action parameters are usually coded as fields in the Criteria section.

You can access this page using the Blank page 2 option on the sample WebCenter SDK menu.

fields.esp

This sample web file displays the various data input fields available to be coded in the Select or Criteria section of a web page. This is a fully functional page where users can modify the data in the input fields and click Execute, having the new data being displayed when the page is reloaded. The Results section of this page shows the JavaScript function calls that are required to code the fields displayed in the Criteria section.

You can access this page using the Form fields action tab on the Example web pages option on the sample WebCenter SDK menu.

graphs.esp

This sample web file displays various graphs that can be produced by incorporating the WebCenter SDK On-Line Graphical Applet (OLGA) into a web page. The page displays the lines of code needed to produce each accompanying graph.

You can access this page using the Graphs action in the More Actions drop-down list on the Example web pages option on the sample WebCenter SDK menu.

intergraph.esp

This sample web file is an interactive page allowing you to modify various parameters passed to the OLGA and view the resulting graph along with the code statements that need to be added to your web page to produce the graph. You modify the parameters to create your own graph format. You then copy the code statements to your real web page and populate the OLGA_SET* variables with live data sourced from an NCL procedure.

You can access this page using the Interactive Graph action tab on the Example web pages option on the sample WebCenter SDK menu.

jsvars.esp

This sample web file displays the various JavaScript variables defined by WebCenter SDK and available for use throughout your web page.

You can access this page using the JavaScript variables action tab on the Example web pages option on the sample WebCenter SDK menu.

menu.xml

This sample web file is a sample menu registration file containing all the menu item definitions to add the other sample web pages to the WebCenter menu structure. These menu items appear on the WebCenter menu if you define the HFS directory containing this menu.xml sample file (as well as all the other sample web files) as your WebCenter User Directory in the $NM WEBCENTER parameter group.

select.htmlf

This sample web file is a sample HTML fragment. It defines the Select section along with all its action tabs for the Example web pages. If you view the source for any of these example web pages, such as jsvars.esp, you will see that they all include select.htmlf.

shusers.esp

This sample web file illustrates how a web page can call an NCL procedure, passing to it parameters defined in a Criteria section. This web page displays a list of users retrieved using a SH USERS command. The list can be optionally filtered using the User filter field in the Criteria section. This page calls the sample NCL procedure $W3SDKS2. This page also illustrates how to populate a table with dynamic data using the ESP FOR-ENDFOR directives.

You can access this page using the CA Mainframe Network Management Users sample action in the More Actions drop-down list on the Example web pages option on the sample WebCenter SDK menu.

styles.esp

This sample web file displays the various CSS style classes defined by WebCenter SDK. It allows you to see what each style class looks like. These CSS styles are coded on HTML tags using the class= attribute. For example, if you require the text within a table cell to be displayed using the subelement style class then you would code the HTML table cell tag as <TD CLASS=data>.

You can access this page using the CSS Styles action tab on the Example web pages option on the sample WebCenter SDK menu.

tables.esp

This sample web file illustrates just what can be achieved in terms of data formatting by using the table* JavaScript functions defined by WebCenter SDK. This page calls a sample NCL procedure named $W3SDKS1 that populates a set of variables with the planetary data displayed in the tables. In this sense the page can be used as an example of how to get data from an NCL procedure as well.

You can access this page using the Table examples action tab on the Example web pages option on the sample WebCenter SDK menu.

WebCenterSDK.jpg

This sample web file is a JPEG image file. If you have specified the directory containing these sample web files as your WebCenter SDK Directory and you have specified /user/WebCenterSDK.jpg in place of a text title in the WebCenter SDK Menu Title field of the $NM WEBCENTER parameter group, then this image displays as the title for the user menu on the WebCenter menu tree.

Appendix B: Menu Registration Files

The WebCenter menu tree is an amalgamated structure built from multiple menu registration files depending upon what products and features are licensed in your product region. A menu registration file is usually (but not always) named menu.xml. It is an XML file containing menu item definitions.

When the Customizer $NM WEBCENTER parameter group is actioned (either at region initialization or when a user presses F6 to action the group), these menu registration files are registered to WebCenter. If you have configured WebCenter SDK, then WebCenter will also try to register a menu.xml file in the user directory you have specified (that is, /user/menu.xml). For your web files to appear in the WebCenter menu tree, they must have a corresponding menuItem entry in this /user/menu.xml file.

Important! If you update your menu.xml after region initialization, you must action the $NM WEBCENTER parameter group and reload the page in your browser for the change to take affect.

This section contains the following topics:

Structure

A menu registration file consists of two XML tags:

- <menuRegistration>…</menuRegistration>
- <menuItem *menu-attributes* />

<menuRegistration>…</menuRegistration> Tag

The <menuRegistration> tag is a containment tag used to contain all the <menuItem *menu-attributes* /> tags within the file. A menu registration file must begin with an open <menuRegistration> tag and end with a closing </menuRegistration> tag.

<menuItem menu-attributes /> Tag

The <menuItem *menu-attributes* /> tag is an empty XML tag that does not contain anything. However, it does have numerous attributes that define a menu item. The following attributes are of interest to users of WebCenter SDK:

desc

Use the desc attribute to specify a short description of the menu item. When the parent menu is clicked, this description opens on the right side of the browser. The description also displays as tooltip text if no other tooltip text is provided on the tooltip attribute.

For example, desc=”Review the messages in the Activity Log”.

enabled

Use the enabled attribute to specify the name of a variable whose value determines if the menu item is to be enabled or disabled. The variable name *must* begin with $W3$ and the leading ampersand (&) is replaced by a tilde (~).

These variables are set in the .ACT-SETVAR subroutine in the $W3MH01X web menu user exit.

If the value of the variable is FALSE, then the menu item is disabled; otherwise, the menu item is enabled. A disabled menu item is displayed in grey and is non-clickable.

For example, enabled=~$W3$SDKOPT1.

id

Use the id attribute to specify a short mnemonic to identify this menu item. The ID you provide must be unique among all your user menu items.

For example, id=”logbrowse”.

label

Use the label attribute to specify the label that opens on the menu for this menu item.

For example, label=”View activity log”.

parentId

If this menu item is to appear within a submenu, then you must specify the ID of the submenu item in the parentId attribute for this item.

For example, parentid=”submenu1”.

seqnum

The seqnum attribute is used internally by the WebCenter menu handler but it must be specified on each menu item. You should set its value to *.

For example, seqnum=”*”.

tooltip

Use the tooltip attribute to specify the text that opens when a user positions their mouse over this menu item.

For example, tooltip=”Select this option to view the messages in the activity log”.

url

Use the url attribute to specify the URL for the web page to be loaded when the user selects this menu item. The URL for all user web pages should begin with /user/. A menu item that represents a submenu (that is, its ID will be referred to by other menu items as their parent ID) should not specify a url attribute.

For example, url=”/user/logbrowse.esp”.

Sample menu.xml

The following sample shows how the attributes are used in the menu.xml file.

```
<menuRegistration>
    <menuItem
      label="Show NetMaster users"
      id="showusers"
      desc="Display a filterable list of NetMaster users"
      tooltip="Select this option to display a list of all NetMaster users"
      url="/user/showUsers.esp"
      enabled=~$W3$SDKUSERS
      seqnum="*"
    />
    <menuItem
      label="My sub-menu"
      id="submenu1"
      desc="A sub-menu to contain related tasks"
      seqnum="*"
    />
    <menuItem
      label="Subtask 1"
      id="subtask1"
      parentid="submenu1"
      desc="Execute subtask 1"
      tooltip="‘Subtask 1’ is positioned under ‘My sub-menu’"
      seqnum="*"
      url="/user/subtask1.esp"
      enabled=~$W3$SUBTASK1
    />
</menuRegistration>
```

Appendix C: $W3MH01X Web Menu User Exit

This section contains the following topics:

$W3MH01X Web Menu User Exit

The WebCenter menu handler provides a user exit that enables you to write code to enable or disable menu options for different users depending upon your security requirements. To disable a menu option, set its corresponding $W3$ variable to FALSE in the ACT-SETVAR subroutine of the *dsnpref*.NMC0.CC2DEXEC($W3MH01X) user exit.

The distributed version of this subroutine documents all the $W3$ variables for the various options on the WebCenter menus. Add any new $W3$ variables you have specified in your menu.xml file to this subroutine. Include the necessary logic to test if the variable is set to TRUE (enabled) or FALSE (disabled).

Note: You must ensure that the name does not clash with any existing CA Mainframe Network Management variables.

Important! If you set the &$W3$* variables without performing an &SECCALL check on an individual user ID, then they are set *globally* for *all* users. In most cases, you set these variables based on a particular user's level of authority.

Important! If modifications are required, we recommend that you create an SMP/E ++USERMOD to record and control the changes. Alternatively, you can copy the distributed member to the region's TESTEXEC data set for modification.

Appendix D: HTML Fragments

This section contains the following topics:

HTML Fragments Overview

One of the components of the WebCenter web server is an include processor. This feature enables you to store a commonly used code fragment as a single file that is then included by other web pages, thereby reducing the coding effort. An HTML code fragment file typically ends with .htmlf (which stands for HTML fragment). These HTML fragments are included in a web page through the use of the #Include directive. For example:

```
<!--#Include Virtual="/sdk/includes/HeadBegin.htmlf"-->
```

Fragments Included with WebCenter SDK

WebCenter SDK comes with a collection of HTML fragments designed to help you develop web pages with the WebCenter look-and-feel. However, you can also develop your own fragments that are then included in multiple web pages. Fragments can include other fragments. For example, in the sample web files supplied with WebCenter SDK, there is a select.htmlf fragment file. This file includes all the fragments necessary to build the Select section for the various example web pages. Each of the example web pages that then include this fragment set SELECTED_ACTION to a different value before including select.htmlf. This results in a different action tab being highlighted on the web page.

The following HTML fragments are available within WebCenter SDK:

- ActionBarBegin.htmlf
- BodyBegin.htmlf
- ClickOkRedo.htmlf
- CloseWindowButton.htmlf
- CriteriaBegin.htmlf
- CriteriaEnd.htmlf
- CriteriaOkRedoOnly.htmlf
- DocumentEnd.htmlf
- HeadBegin.htmlf
- HelpButton.htmlf
- olga.htmlf
- ResultsBegin.htmlf
- ResultsEnd.htmlf
- SelectBegin.htmlf
- SelectEnd.htmlf
- SelectInputBegin.htmlf
- SelectInputEnd.htmlf

Note: Unless explicitly stated, all of the supplied HTML fragments assume JavaScript mode upon entry (that is, the document has an open <SCRIPT> tag) and exit, with the exception of HeadBegin.htmlf. This fragment assumes a blank document on entry because it must be the first statement in a web page that uses the WebCenter look-and-feel.

Fragment Dependencies

Many of the supplied fragments are designed to be dependant on other fragments because one fragment might end an HTML structure begun by a previous fragment. If the document contains one of fragments in the following fragment groups, then the document must also contain the other fragments in the fragment group, but they do not have to necessarily appear consecutively in the document:

- HeadBegin.htmlf
  BodyBegin.htmlf
  DocumentEnd.htmlf
- SelectBegin.htmlf
  SelectInputBegin.htmlf
  SelectInputEnd.htmlf
  SelectEnd.htmlf
- CriteriaBegin.htmlf
  CriteriaEnd.htmlf
  (mutually exclusive with CriteriaOkRedoOnly.htmlf)
- ResultsBegin.htmlf
  ResultsEnd.htmlf

Fragment Syntax, Context, and Related HTML Tags

This topic lists each fragment along with its syntax, its context, the HTML tags it closes, the HTML tags it opens, and a brief description of its function.

- Syntax shows the #Include statement to be coded to use the fragment.
- Context shows what HTML tags the fragment assumes to be open on entry into the fragment.
- Tags Closed illustrates what HTML tags are effectively closed by this fragment.
- Tags Opened shows what HTML tags are effectively opened by this fragment. A fragment may effectively close or open no HTML tags, but internally it may close an HTML </TABLE> tag and then open a new <TABLE>.

ActionBarBegin.htmlf

Function:

Begins the table structure that will contain the action bar tabs.

Syntax:

```
<!--#Include Virtual="/sdk/includes/ActionBarBegin.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

<TABLE><TR>

BodyBegin.htmlf

Function:

Calls the init( ) function that establishes the WebCenter frameset and menu tree status. Begins the body of the document, sets the document style, defines an onUnload function, and opens a form called docform.

Syntax:

```
<!--#Include Virtual="/sdk/includes/BodyBegin.htmlf"-->
```

Context:

<HEAD><SCRIPT>

Tags Closed:

</HEAD>

Tags Opened:

<BODY><FORM>

ClickOkRedo.htmlf

Syntax:

```
<!--#Include Virtual="/sdk/includes/ClickOkRedo.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

None

Function:

Writes the contents of the CLICK_OKREDO_TEXT variable into a table using the subelement class. This fragment is usually included after ResultsBegin.htmlf.

CloseWindowButton.htmlf

Function:

Adds a close window button to a page. This fragment is usually included in a dialog web page immediately before the ResultsEnd.htmlf fragment.

Syntax:

```
<!--#Include Virtual="/sdk/includes/CloseWindowButton.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

None

CriteriaBegin.htmlf

Function:

Opens the table for the Criteria section, sets the criteria title, and adds the expand/collapse button (for Internet Explorer). This fragment is usually paired with a corresponding CriteriaEnd.htmlf fragment.

Syntax:

```
<!--#Include Virtual="/sdk/includes/CriteriaBegin.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

<TABLE><TR>[assign the value for TD in your book]<DIV><TABLE><TR>

CriteriaEnd.htmlf

Function:

Terminates the Criteria section tables and adds the Execute button.

Syntax:

```
<!--#Include Virtual="/sdk/includes/CriteriaEnd.htmlf"-->
```

Context:

<SCRIPT> {<TR> | <TABLE>}

Tags Closed:

</TR></TABLE></DIV></TD></TR></TABLE>

Tags Opened:

None

CriteriaOkRedoOnly.htmlf

Function:

Creates an empty Criteria section with only the Execute button.

Syntax:

```
<!--#Include Virtual="/sdk/includes/CriteriaOkRedoOnly.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

None

DocumentEnd.htmlf

Function:

Closes the docform form and terminates the document with the standard HTML termination tags.

Syntax:

```
<!--#Include Virtual="/sdk/includes/DocumentEnd.htmlf"-->
```

Context:

<HTML><BODY><FORM><SCRIPT>

Tags Closed:

</SCRIPT></FORM></BODY></HTML>

Tags Opened:

None (should be the last statement within a web page)

HeadBegin.htmlf

Function:

Sets the standard document header, links to the sdk.js JavaScript library and the sdk.css Cascading Style Sheet library.

Syntax:

```
<!--#Include Virtual="/sdk/includes/HeadBegin.htmlf"-->
```

Context:

None (should be the first statement within a web page)

Tags Closed:

None (should be the first statement within a web page)

Tags Opened:

<HTML><HEAD><SCRIPT>

HelpButton.htmlf

Function:

Adds the help button icon to the top right corner of a page. This fragment must be included immediately after SelectBegin.htmlf

Syntax:

```
<!--#Include Virtual="/sdk/includes/HelpButton.htmlf"-->
```

Context:

<SCRIPT><TR>

Tags Closed:

None

Tags Opened:

None

olga.htmlf

Function:

Adds the On-Line Graphical Applet to a web page to produce a graph wherever required.

Syntax:

```
<!--#Include Virtual="/sdk/includes/olga.htmlf"-->
```

Context:

<SCRIPT>

Tags Closed:

None

Tags Opened:

None

ResultsBegin.htmlf

Function:

Opens the table for the Results section, sets the white background and sets the results title.

Syntax:

```
<!--#Include Virtual="/sdk/includes/ResultsBegin.htmlf"-->
```

Context:

<SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

<TABLE><TR>[assign the value for TD in your book]

ResultsEnd.htmlf

Function:

Closes the Results section table structure.

Syntax:

```
<!--#Include Virtual="/sdk/includes/ResultsEnd.htmlf"-->
```

Context:

<TABLE><TR>[assign the value for TD in your book]<SCRIPT>

Tags Closed:

</TD></TR></TABLE>

Tags Opened:

None

SelectBegin.htmlf

Function:

Creates the table for the Select section.

Syntax:

```
<!--#Include Virtual="/sdk/includes/SelectBegin.htmlf"-->
```

Context:

< SCRIPT> [[assign the value for TD in your book]]

Tags Closed:

None

Tags Opened:

<TABLE><TR>[assign the value for TD in your book]<TABLE><TR>

SelectEnd.htmlf

Function:

Closes the tables for the Select section.

Syntax:

```
<!--#Include Virtual="/sdk/includes/SelectEnd.htmlf"-->
```

Context:

<TABLE><TR>[assign the value for TD in your book]<TABLE><TR>< SCRIPT>

Tags Closed:

</TR><TABLE></TD></TR></TABLE>

Tags Opened:

None

SelectInputBegin.htmlf

Function:

Creates a tabular structure for input fields in the Select section.

Syntax:

```
<!--#Include Virtual="/sdk/includes/SelectInputBegin.htmlf"-->
```

Context:

<TABLE><TR>< SCRIPT>

Tags Closed:

None

Tags Opened:

[assign the value for TD in your book]<TABLE>

SelectInputEnd.htmlf

Function:

Closes the Select section input table structure.

Syntax:

```
<!--#Include Virtual="/sdk/includes/SelectInputEnd.htmlf"-->
```

Context:

<TABLE><TR>[assign the value for TD in your book]<TABLE><
SCRIPT>

Tags Closed:

</TABLE></TD></TR></TABLE>

Tags Opened:

None

Appendix E: JavaScript Variables

This section contains the following topics:

JavaScript Variables

This topic describes the JavaScript variables that are defined in /sdk/sdk.js and are available for use by your web page if you have included /sdk/includes/HeadBegin.htmlf.

- Some of these variables *must* be modified as required by your web page so that your web page sits appropriately in the WebCenter framework.
- Others, such as the variables containing color definitions, are only meant to be referred to and must not be changed because this may affect the way your page looks and interfaces with WebCenter.

ACTION_PATH

Contains the file path to the document specified by an action URL. It defaults to NULL because most action URLs are sourced from the same directory. However, occasionally the need arises to action documents in one directory from a document in a separate directory.

Change as required.

CLICK_OKREDO_TEXT

Contains the text rendered by the ClickOkRedo.htmlf include. It defaults to Click Execute to proceed, and must be changed before including ClickOkRedo.htmlf.

Change as required.

CRITERIA_PREFIX

Contains the heading prefix for the Criteria section. It defaults to Criteria - , and must be set before including CriteriaBegin.htmlf.

Change as required.

DOACTION_LEVEL

Indicates the document level where the new document, indicated by its associated URL, will be loaded when an action other than the current action is selected. This enables an action that creates a lower-level frameset (such as Alerts for node) to return to a higher frameset level when a new action is selected.

Change as required.

ENABLE_ACTION_BUTTON

Usually set to FALSE. Set to TRUE if you want the currently active action tab (apart from the More actions tab) to be selectable.

Change as required.

FIELD_ERROR

If your web page uses validation functions, then these functions can set this variable to a value other than 0 if they detect a validation error.

Change as required.

IMAGE_PATH

The file path to the directory containing the WebCenter image files. Currently set to /public/images/.

Note: Read only – Do not change.

INCLUDE_PLEASEWAIT

For Microsoft Internet Explorer, set to TRUE if the text Please wait, request in progress... is to be displayed in the Results section of the document prior to the document load completing.

Change as required.

INCLUDE_SELECT_TEXT

Set to TRUE to include the text string for selected options from a form's drop-down lists on the next requested URL.

Change as required.

INCLUDE_SELECT_VALUE

Set to TRUE to include the value string for selected options from a form's drop-down lists on the next requested URL.

Change as required.

INIT_FUNCTION

The name of a function to be executed when a web page has finished loading.

Change as required.

IS_IE

This Boolean variable is set to TRUE if the browser is Microsoft Internet Explorer and is used by the WebCenter infrastructure in building your web page.

Note: Read only – Do not change.

IS_NN

This Boolean variable is set to TRUE if the browser is Mozilla 4.x and is used by the WebCenter infrastructure in building your web page.

Note: Read only – do not change.

LOAD_FRAMESET

Set to TRUE if the loading of this web page will cause the surrounding WebCenter frameset and menu tree to be loaded as well. Only set this to FALSE for those web pages that do not require the WebCenter frameset, such as web dialog boxes that are displayed in new windows.

Change as required.

MAX_ACTIONBUTTONS

The maximum number of action buttons or tabs to be displayed before the More Actions button is displayed. The default is 5.

Change as required.

MENU_URL

The URL to be used for highlighting an option on the WebCenter menu tree. This defaults to the *location.pathname* for the current web page.

Change as required.

OKREDO_HASH

Optional hash text to append to the end of an Execute URL identifying a particular place within a web page to jump to.

Change as required.

OKREDO_LEVEL

The level of the Document Object Model (DOM) where you want to load the page requested by the Execute button.

Change as required.

OKREDO_URL

The URL for the page requested by the Execute button. This defaults to the location.pathname for the current web page.

Change as required.

OKREDO_VALIDATE_FUNCTION_LIST

A list of semicolon-separated functions to be executed when a user clicks the Execute button. These functions typically validate fields in the Criteria section of a web page that are specific to this web page.

Change as required.

ON_ENTER

The name of a function to be executed when the Enter key is pressed.

Change as required.

ON_UNLOAD

The name of a function to be executed when the web page is unloaded (that is, replaced by another URL).

Change as required.

PRODUCT_NAME

The simple name of the software product that is used in the browser's title bar (for example, NetMaster or SOLVE).

Note: Read only – Do not change.

REAL_URL

The URL that represents the real URL for this page. This defaults to the *location.pathname* for the current web page. It is used to reinstate the page if the loading of this page causes the WebCenter frameset to be loaded first.

Change as required.

RESULTS_PREFIX

Contains the heading prefix for the Results section. It defaults to Results and must be modified before including ResultsBegin.htmlf.

Change as required.

SELECTED_ACTION

Set to indicate what action tab ID this document relates to. It is used by the showActionBar() function to determine which action tab should be displayed as active and must be set before this function is called. It defaults to NULL.

Change as required.

SELECT_PREFIX

Contains the heading prefix for the Select section. It defaults to NULL and must be set before including SelectBegin.htmlf.

Change as required.

TITLE_TEXT

Contains the title displayed in the browser's title bar and in the headings of the Select, Criteria, and Results sections. It defaults to NULL and can be changed as needed before the relevant section is included.

Change as required.

VALIDATE_FUNCTION_LIST

Contains a list of semicolon-separated JavaScript function calls that are to be executed to validate the current document when the user selects an action on this page. It defaults to validate( ), a function that returns TRUE. Validation functions should return TRUE if everything is fine, otherwise they should return FALSE. The functions listed in the v_validateFunctionList should only validate the fields contained in the Select section of a document. These fields should be validated no matter what action is selected.

Change as required.

Appendix F: JavaScript Functions

This chapter describes the JavaScript functions that are defined in /sdk/sdk.js and are available for use by your web page if you have included /sdk/includes/HeadBegin.htmlf.

Note: Including /sdk/includes/HeadBegin.htmlf implies that you are developing a web page with the WebCenter look-and-feel. JavaScript is case-sensitive, so you need to code any JavaScript function calls in the same case that they are documented here.

This section contains the following topics:

Action Tab Functions

The following functions assist in building the action tabs on a page and the pop-up action menu as well:

addAction

The addAction function adds an action object to the actions array for future display as an action tab or as an action within the pop-up action menu.

Syntax

```
addAction(actionId, label, url, tooltip)
```

Parameters

- actionId—The action ID mnemonic that identifies this action
- label—The text label displayed for this action
- url—The web page to request when the user selects this action
- tooltip—Text that appears when you move the mouse over the button

Example

```
addAction(tables,Table examples, tables.esp,Show how to create a table);
```

addActionSeparator

The addActionSeparator functions adds a separator line to the list of actions in the pop-up menu.

Syntax

```
addActionSeparator()
```

Parameters

None.

Example

```
addActionSeparator()
```

setAction

The setAction function sets the status of an action to active (that is, the currently selected action).

Syntax

```
setAction(actionId)
```

Parameters

- actionId—The action ID mnemonic that identifies the action

Example

```
setAction(tables);
```

setActionDisabled

The setActionDisabled function sets the status of an action to disabled (that is, not selectable).

Syntax

```
setActionDisabled(actionId)
```

Parameters

- actionId—The action ID mnemonic that identifies the action

Example

```
setActionDisabled(tables);
```

setActionPath

The setActionPath function sets the file path prefix for all subsequent addAction( ) calls to the file path provided.

Syntax

```
setActionPath(path)
```

Parameters

- path—The slash-terminated file path prefix

Example

```
setActionPath(/user/operator/);
```

showActionBar

The showActionBar function renders the action tabs within the Select section of a web page. It is passed to the action ID of the action to be rendered as active and passes this action ID on a call to setAction( ).

Syntax

```
showActionBar(actionId)
```

Parameters

- actionId—The action ID mnemonic that identifies the active action

Example

```
showActionBar(tables);
```

Button Functions

The following functions are useful when you need to add a button to a web page:

addButton

The addButton function writes the HTML to render a WebCenter-style button in a web page.

Syntax

```
addButton(text,url,tooltip)
```

Parameters

- text—Text to place next to the button image.
- url—URL to be actioned when the button is clicked
- tooltip—Text that appears when you move the mouse over the button

Example

```
addButton(Previous message, prevMsg.esp,Get the message immediately prior to
this one);
```

getButton

Returns a string containing the HTML for a standard button, but does not render the button.

Syntax

```
getButton(url, text, tooltip)
```

Parameters

- url—URL to be actioned when the button is clicked
- text—Text to place next to the button image
- tooltip—Text that appears when you move the mouse over the button

Example

```
document.write("[assign the value for TD in your
book]" + getButton(MY_URL, "History graph") + "</td>");
```

Form Field Functions

The following functions assist in adding data input fields to a form, whether in the Select section or the Criteria section:

addCheckBoxField

The addCheckBoxField function writes an <INPUT> tag of type=CHECKBOX to a form within a document. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addCheckBoxField(fieldName,label,initValue,default)
```

Parameters

- fieldName—The name to assign to the check box, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- label—The label to associate with this check box
- initValue—The initial value to assign to this check box (YES | NO)
- default—The default value to assign to this check box if the initial value is null, either YES or NO

Example

```
addCheckBoxField("linenum", "Include line numbers", "YES", "YES");
```

addField

The addField function writes an <INPUT> tag of type=TEXT to a form within a document. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addField(fieldName,label,maxLength,size,initValue)
```

Parameters

- fieldName—The name to assign to the field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- label—The label to associate with this field
- maxLength—The maximum number of characters that may be typed in the field.
- size—The visible width of the text input field
- initValue—The initial value to assign to the text in this input field

Example

```
addField("cmd", "Command text", 255, 30, sh users);
```

addHiddenField

The addHiddenField function writes an <INPUT> tag of type=HIDDEN to a form within a document. Hidden fields are useful for setting and maintaining state information.

Syntax

```
addHiddenField(fieldName,initValue)
```

Parameters

- fieldName—The name to assign to the field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable *&Request.QueryString.Named.fieldName*
- initValue—The initial value to assign to the hidden field

Example

```
addHiddenField("mode","search");
```

addPasswordField

The addPasswordField function writes an <INPUT> tag of type=PASSWORD to a form within a document. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addPasswordField(fieldName,label,maxLength,size,initValue)
```

Parameters

- fieldName—The name to assign to the field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- label—The label to associate with this field
- maxLength—The maximum number of characters that may be typed in the field
- size—The visible width of the input field
- initValue—The initial value to assign to the text in this input field

Example

```
addPasswordField("opw","Old password",8,8,"");
```

addRadioButtonField

The addRadioButtonField function writes an <INPUT> tag of type=RADIO to a form within a document. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addRadioButtonField(fieldName,label,value,initValue)
```

Parameters

- fieldName—The name to assign to the field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- label—The label to associate with this field
- value—The value to be assigned to the field when the radio button is checked
- initValue—The value to specify if the radio button should be checked: CHECKED or null

Example

```
addRadioButtonField("searchdir", "Forward", "FWD", "CHECKED");
```

addTextArea

The addTextArea function writes a <TEXTAREA> tag to a form within a document. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addTextArea(fieldName,label,rows,cols,initvalue)
```

Parameters

- fieldName—The name to assign to the text area field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- label—The label to associate with this text area
- rows—The number of editable lines to display
- cols—The number of monofont characters to display in the width
- initValue—The initial value to assign to the text in this text area field

Example

```
addTextArea("address","Enter your address","4","","");
```

buildList

The buildList function writes a <SELECT> tag to a form within a document. A <SELECT> tag displays a drop-down single-select list or a scrollable multi-select list. It begins by writing a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
buildList(itemList, selList, fieldName, mode, noentry, label, valList,
events, size)
```

Parameters

- itemList—The full list of items to display in the selection list
- selList—A list of the selected items (upon entry into the page).
- fieldName—The name to assign to the text area field, subsequently used in the query portion of request URLs and may be referenced using the ESP variable &Request.QueryString.Named.*fieldName*
- mode—Multi-select or single-select: multiple | single
- noentry—What to display if the list contains no entries:
- blank: display a blank line
- noblank: do not display anything
- string: display this string
- label—The label to associate with this list
- valList—A list of values to correspond with the itemList if the value for the items displayed in the list is different than their display text
- events—A string of onEvent-type statements for event processing
- size—For a multi-select list, the number of items to display

Example

```
buildList("Pizza,Chinese,Hamburgers,Sandwich,Roast
Lamb,Seafood","Pizza","food","single","","My favorite food is");
```

newColumn

Creates a new column within the Criteria section for the positioning of form fields and sets the alignment of the column. It begins by writing a [assign the value for TD in your book] tag into the web page so it must be called within the context of an HTML <TR> tag and *must* be called before adding any fields into the Criteria section.

Syntax

```
newColumn(align)
```

Parameters

- align—The alignment for data within the column: left | center | right

Example

```
newColumn(left);
```

Text Functions

Use the following functions to write text to the body of a document:

addTableText

The addTableText function writes a <TABLE> tag into the web page so it must be called within the context of an HTML [assign the value for TD in your book] tag (that is, within the Results section of a web page) or outside of *all* table structures within the web page.

Syntax

```
addTableText(label)
```

Parameters

- label—The text to be written to the document

Example

```
addTableText("10 users displayed");
```

addText

The addText function writes a <TR> tag into the web page so it must be called within the context of an HTML <TABLE> tag (that is, within the Criteria section of a web page).

Syntax

```
addText(label[,colspan])
```

Parameters

- label—The text to be written to the document
- colspan—The number of table columns this text needs to span

Example

```
addText("Enter your name here");
```

Data Formatting Functions

The following linePrint* and table* functions simplify data formatting:

linePrintBegin

The linePrintBegin function creates a table into which line-by-line output will be written. This is useful when displaying 3270-style fixed width command output.

Syntax

```
linePrintBegin([format])
```

Parameters

- format—An optional format indicator that, if provided, must be one of the following values:
  - NOBOX: Do not put a box around the output.
  - FULLWIDTH: Use the available width of the window.

Example

```
linePrintBegin("FULLWIDTH");
```

linePrint

The linePrint function writes a line of text into a line-by-line output table previously established by the linePrintBegin() function.

Syntax

```
linePrint(text [,format])
```

Parameters

- text—The text to be rendered
- format—An optional format indicator that, if provided, must be one of the following values:
  - MSGID: Hyperlink the first word to message help as a message ID.
  - CMDID: Hyperlink this text to command help as a command ID.
  - NUM: Insert line numbers at the beginning of each row.
  - TRANSPARENT: Use container's background color.
  - HTML: The text contains HTML so substitute problematic characters.

Example

```
linePrint("N13511 19 USERS LOGGED ON",MSGID);
```

linePrintEnd

The linePrintEnd function closes the line-by-line output table created by linePrintBegin( ).

Syntax

```
linePrintEnd()
```

Parameters

None.

Example

```
linePrintEnd();
```

tableBegin

The tableBegin function creates the containing table structure and must be the first table statement issued.

Syntax

```
tableBegin([title])
```

Parameters

- title—The text to place in the title bar of this table

Example

```
tableBegin("User list");
```

tableAlign

The tableAlign function sets the column alignment for each column in a table and must be issued before a tableHeading or tableRow statement. The value must be either left, center, or right, and defaults to left. All values are positional.

Syntax

```
tableAlign(["left"|"center"|"right"|""][,…])
```

Parameters

Each parameter represents a table column and must contain one of the following values: left, center, right, or empty string.

Example

```
tableAlign("","","right");
```

tableWrap

The tableWrap function sets the column wrapping for each column in a table and must be issued before a tableHeading or tableRow statement. The value must be either wrap, nowrap, or empty string, which defaults to wrap. All values are positional.

Syntax

```
tableWrap(["wrap"|"nowrap"|""][,…])
```

Parameters

Each parameter represents a table column and must contain one of the following values: wrap, nowrap, or empty string.

Example

```
tableWrap("","nowrap");
```

tableSorted

The tableSorted function sets the sorted indicator in a column heading indicating whether the column is sorted in ascending or descending order. The function must be issued before a tableHeading or tableRow statement. The value must be either A (ascending), D (descending), or empty string – not sorted. All values are positional.

Syntax

```
tableSorted(["A"|"D"|""][,…])
```

Parameters

Each parameter represents a table column and must contain one of the following values: A, D, or empty string.

Example

```
tableSorted ("","A");
```

tableUrl

The tableUrl function defines URLs for sort functions. If a column has a sort function URL, then the column heading has a button-like appearance and is a link. All values are positional.

Syntax

```
tableUrl(url[,…])
```

Parameters

Each parameter represents a table column and must contain a valid URL for a JavaScript sort function, or empty string.

Example

```
tableUrl("sort('name')","sort('date')");
```

tableHeading

The tableHeading function sets the headings for each column of a table and must be issued before a tableRow statement. The value must be a string. All values are positional. To set the column headings for a 3-column table to tic, tac, and toe, code tableHeading(“tic”,”tac”,”toe”).

Syntax

```
tableHeading([string][,…])
```

Parameters

Each parameter represents a table column and must contain a string value.

Example

```
tableHeading("Date","Time","User","Message");
```

tableHeadingComplex

The tableHeadingComplex function sets the headings for each column of a table and must be issued before a tableRow statement. It allows you to specify how many columns a table heading will span. The value must be a string. All values are positional.

Syntax

```
tableHeadingComplex([string, colspan][,…])
```

Parameters

Each parameter pair represents a table column heading and the number of columns it will span. The column heading is a string value and the column span value is numeric.

Examples

```
tableHeadingComplex("Date",1,"Time",1,"Values",3);
tableHeadingComplex("1st",1,"2nd",1,"3rd",1);
```

tableRow

The tableRow function sets the data for each column. The values provided must be strings and all values are optional and positional.

Syntax

```
tableRow([string][,…])
```

Parameters

Each parameter represents a table column and must contain a string value.

Example

```
tableRow("19-Mar-2003","12:34:09","usr1","Hello world");
```

tableRowSpan

The tableRowSpan function sets the data for each column and allows the data to span multiple columns. The values provided must be string-number pairs and all values are optional and positional.

Syntax

```
tableRowSpan([string, colspan][,…])
```

Parameters

Each parameter pair represents a table column and must contain a string value followed by a numeric.

Example

```
tableRowSpan("19-Mar-2003",1,"12:34:09",1,"usr1",1,"Hello world",4);
```

tableEnd

The tableEnd function closes the containing table structure and must be the last table statement issued. There are no parameters.

Syntax

```
tableEnd()
```

Parameters

None.

Example

```
tableEnd();
```

Appendix G: CSS Style Classes

This section contains the following topics:

CSS Style Classes

Include /sdk/includes/HeadBegin.htmlf if you are developing a web page with the WebCenter look-and-feel.

The following CSS style classes are defined in /public/castyles.css and are available for use by your web page if you have included /sdk/includes/HeadBegin.htmlf.

data

Normal black text used for the majority of text on a page.

datalink

Blue text used to display hyperlinks.

error

Bold red text used to highlight an error message.

DATA_fixed_width

Fixed width text used to display formatted text such as activity logs and command output.

label

Bold black text used to display labels.

Appendix H: ESP Directives

This section contains the following topics:

ESP Directives Overview

Executable Server Page (ESP) directives are directives that allow the building of dynamic pages at the server rather than having only dynamic HTML to control the web page construction. The main benefit of ESP directives is that they allow you to tie the data and the visual elements (such as HTML, text, JavaScript, and images) on the page together without embedding these elements into your programs. The programs only need to retrieve data.

ESP directives are coded into a WebCenter web page so that the web page can interact with CA Mainframe Network Management products. The web page must be named with the .esp suffix. ESP directives enable your web page to call an NCL procedure and share variables back into your web page. ESP includes three directive types with any other data (such as HTML and JavaScript) interspersed in between. All directives begin with the directive start delimiter <% and end with the directive end delimiter %>.

ESP Directive Types

The very first character after the start delimiter is the directive indicator, which identifies the type of directive. The valid directives types are:

<%= (Output Directive)

Use this directive to specify data to be placed into the page. The directive indicator is an = character after the start delimiter.

<%@ (Processing Directive)

Use this directive to specify the general processing controls (such as the CONTROL statement). The directive indicator is an @ character after the start delimiter.

<% (Scripting Directive)

Use this directive if you want to use language statements that control logical processing of the page. The directive indicator is a blank character after the start delimiter.

Notes:

- Directives can be specified across multiple lines and extraneous blanks between keywords are ignored. Keyword values must be quoted in order to keep the text as is.
- The directive start and end delimiters *must not* appear in the unsubstituted data of a verb; you must use a built-in variable for this.
- All error and trace messages are inserted into the resulting page.

Output Directive

The output directive allows for the substitution of a variable or the result of an OML function into a page.

Syntax

The output directive uses the following syntax:

```
<%= {   &variable | { package:function | built-in }  [ share=variableList ]
[ escape=char ] [ parm1=parm1 [ parm2=parm2 … [ parm25= parm25] ] ] } %>
```

Keywords

The output directive contains the following keywords:

&variable

This is the first of two forms for the output directive. Specifying any OML variable name will result in the value of this variable being substituted into the page.

{ package:function | built-in }

This is the second form for the output directive. Specifying either your own OML function to invoke or one of the ESP built-in functions (see page 90) will result in the value returned by the function being substituted into the page.

share=variableList

Identifies the variables to be shared to the called procedure. By default, no variables are shared. The format must be a list of names separated by commas. If a generic set of variables is to be shared, then specify a prefix followed by an asterisk. To share just a stem or MDO, follow the name with a period. Do not specify ampersands before the names.

escape=char

Specifies a variable substitution escape character. The end result is to parse each parameter on the escape character, do variable substitution on each chunk, and then concatenate them together. This allows unintentionally invalid variable names to be processed. For example, escape=\ parm1=hello&fred\world results in parm1=hello whole world (assuming that &fred contains whole). This processing is only valid for the PARM parameters.

parm1= to parm25=

These keywords identify the parameters to be passed to the function. The parameters that you identify here must be parameters that you can use with the SETVARS command.

If a variable is specified, the current value for that variable is returned and inserted into the page. If an OML function is specified, the function is called, and the data returned by the function is inserted into the page.

Examples

```
<%= &ipaddr %>
<%= &Request.QueryString.ipaddr %>
<%= $IPPKG:SNMPGetSingleAttribute parm1='&ipaddr' parm2='oid=1.2.3.4.56.1.45.1.0'
 %>
```

Processing Directive

The processing directive controls general aspects of ESP execution.

Syntax

The processing directive uses the following syntax:

```
<%@    [ subchar={ & | c } ]  [ trace={on | on-html | on-js | off } ] %>
```

Keywords

The processing directive contains the following keywords:

subchar= { & | c }

Specifies the default substitution character to be used in all scripting statements.

Trace={ on | on-html | on-js |off }

If set to on, then trace messages are inserted into the output page. These messages contain the following information:

- What statements looked like in the source.
- What statements looked like after substitution.
- In some cases, other relevant information about a particular statement or directive.

If set to on-html, then WebCenter SDK assumes that the page is processing HTML and the trace messages are generated assuming the HTML context. If on-js is specified, then WebCenter SDK assumes the context is JavaScript.

Examples

```
<%@ subchar=% security=03A5F001C %>
<h1>Managing IP Nodes from &lt%nmid&gt</h1>
```

Scripting Directive

The scripting directive encapsulates all the logical processing to be performed in the page. The SUBSTITUTE, IF, MODEL, WHILE, and FOR statements all support nesting. The general statement format is:

Syntax

The scripting directive uses the following syntax:

```
<%  verb [ keyword1=value1 [ keyword2=value2 … ] ] %>
```

There can be only one statement within the directive start and end delimiters. All verbs are a single word indicating the action to perform and the keywords control variable aspects of the performed action. All the keywords must be keywords that you can use with the SETVARS command.

Examples

```
<%  set var=ipaddr data=&Request.QueryString.ipaddr %>
```

CALL Statement

Use the CALL statement to call an NCL or OML procedure to perform some processing and optionally to return variables into the page context. The identified NCL or OML procedure is called, sharing in the variables identified. When the procedure returns, the variables and MDOs that are returned are available within the scope of the page.

The called procedure is invoked with INTSAVE to allow it to perform any command processing required.

The following variables are set for use in other scripting statements.

&callRetcode

Contains the return code set by the procedure invoked.

&callSysmsg

Contains the value of &SYSMSG as returned by the procedure invoked.

Syntax

This statement uses the following syntax:

```
<%  call  proc=[package:]procedure  [ share=variableList ] [ escape=char ]
[ parm1=parm1 [ parm2=parm2 … [ parm25= parm25] ] ] %>
```

Keywords

This statement contains the following keywords:

proc=[package:]procedure

Identifies the name of the OML or NCL procedure to call.

share=variableList

Identifies the variables to be shared to the called procedure. By default, no variables are shared. The format must be a list of names separated by commas. If a generic set of variables is to be shared, then specify a prefix followed by an asterisk. To share just a stem or MDO, then follow the name with a period. Do not specify ampersands before the names.

Note: Be careful when sharing variables to a called NCL procedure because the ESP processor and the other components of the web server are written in OML and use long variable names as well as stem variables. Because NCL only supports variable names of up to 12 characters, you may experience NCL errors if you are not careful. For example, do not code share=*.

escape=char

Specifies a variable substitution escape character. The end result is to parse each parameter on the escape character, do variable substitution on each chunk, and then concatenate them together. This allows unintentionally invalid variable names to be processed. For example, escape=\ parm1=hello&fred\world results in parm1=hello whole world (assuming that &fred contains whole). This processing is only valid for the PARM parameters.

Parm1= to Parm25=

These keywords identify parameters to be passed to the procedure that is to be invoked. The parameters that you identify here must be parameters that you can use with the SETVARS command.

Examples

```
<h1>Node Address: &ipaddr</h1>
<% call proc=$IPCALL share='$IPDATA.,$IPGH*'
   parm1='ACTION=GET' parm2='CLASS=IPHOST' parm3='ADDRESS=&ipaddr' %>
<% if cond='&callRetcode = 0' %>
   <% set var=stack data=&$IPDATA.HOSTID.STACK %>
   <p><%=&stack %> in use</p>
<% endif %>
<h1>Managing IP Node &ipaddr from &nmid</h1>
```

FLUSH Statement

This statement tells the ESP processor to release all HTML data generated so far. All buffers are cleared and the data is sent to the browser.

Syntax

This statement uses the following syntax:

```
<%  flush %>
```

Keywords

The FLUSH statement contains no keywords.

Examples

```
<h1>Traceroute Results List for &ipaddr</h1>
<% set var=qid data='TRACEROUTE.&ipaddr.&ZUNIQUE' %>
<% call proc=DOTRCRT share='$IPDATA.'
   parm1='ADDRESS=&ipaddr QUEUE=&qid' %>
<% model queue=&qid prefix=$ip%> <p><%=&$iptrResHost %>  <%=&$iptrResTime %>
   </p>
   <% flush %>
<% endmodel %>
```

FOR – ENDFOR Statements

This statement provides a fixed iterative loop. The page data is processed exactly the number of times specified. An iteration variable must be specified and is automatically set to the starting value and incremented by the given value.

Syntax

This statement uses the following syntax:

```
<%  for  var=variable   [ start=n ]  count=m  [ by=a ] %>
   page data
<% endfor %>
```

Keywords

This statement contains the following keywords:

var=variable

Identifies the name of the iteration variable used when processing the page data. For each iteration of the FOR statement, &variable is available for substitution. Its value will be controlled by the start= and by= parameters.

start=n

Indicates that the iterator's initial value is to be *n*. The default is 1.

count=m

Indicates that the iterations are to continue *m* times.

by=a

Indicates that, for each iteration, the iterator variable is to be incremented by *a*. The default is 1.

Examples

```
<h1>Results List for &ipaddr</h1>
<% for var=i start=&resPos count=5 %>
   <% set var=triptime data=&$IPDATA.results.{&i}.tripTime %>
   <p><%=&triptime %></p>
<% endfor %>
```

IF - ELSE - ENDIF Statements

This statement allows conditional processing. If the condition evaluates to TRUE, then the data after the IF statement (that is, page data1) is processed, otherwise the data after the ELSE statement (that is, page data2) is processed, provided an ELSE statement is present.

Syntax

This statement uses the following syntax:

```
<%  if  cond=boolexprCondition  %>
page data1
   [  <% else %>    page data2  ]
   <% endif %>
```

Keywords

This statement contains the following keywords:

cond=boolexprCondition

This is the standard input to the BOOLEXPR function. It may contain simple variables previously set by the SET statement, system variables, or any variables returned by a called procedure.

Examples

```
<h1>Node Address: &ipaddr</h1>
<% if cond='&ipaddr = "&sys.tcpip.host.address"' %>
   <p>Errors occurred on local host</p>
<% else %>
   <p>Errors occurred for remote host <%= &ipaddr %></p>
<%  endif %>
```

MODEL Statement

The MODEL statement repeatedly processes the page data (delimited by the ENDMODEL statement), making all item variables available for substitution (that is, substitution is ON within the model statement). The items are defined by the iteration method specified, that is, mdo, stem, slproc, or queue.

Syntax

This statement uses the following syntax:

```
<%  model  [ var=variable ]
{ mdo=mdoComponent prefix=variablePrefix |
   stem=omlStem prefix=variablePrefix |
   slproc=selectionListServiceProc  [ escape=char ]
        [ data=serviceProcData ] |
   queue=queueName prefix=variablePrefix [wait={60|nnnn}]}
[ start=n ]  [ count=m ] [ direction={FWD | BWD} ] %>
page data
         <% endmodel %>
```

Keywords

This statement contains the following keywords:

var=*variable*

Identifies the name of the iteration variable to use when processing the elements in the data structure. Therefore, for each iteration of the MODEL statement, &variable is available for substitution.

prefix=variablePrefix

For the MDO, STEM, and QUEUE types, identifies the prefix that is appended to the start of all attributes made available for substitution within the model scope. Therefore, for each iteration of the MODEL statement, all variables with prefix &variablePrefix are deleted, and a new set is created and made available for substitution. This keyword cannot be specified when SLPROC is used.

Note: Ensure that the prefix specified does not include variables that you do not want deleted either after or during the execution of the model statement. After the model statement has completed, all variables with the specified prefix are deleted.

mdo=mdoComponent

Indicates that the variable source is an MDO. The MDO is assumed to have the structure:

- mdoComponent.{*}.attribute1
- mdoComponent.{*}.attribute2 …

For each iteration of the MODEL statement, the attributes for mdoComponent.{n}.* are available as variables for which substitution can take place. The variable names are &variablePrefixAttribute1 ….

Note: The VARSUB command determines the value of this keyword using NCL-style substitution. Therefore, OML-style variables cannot be used to specify the name of the MDO component to use.

stem=omlStem

Indicates that the variable source is an OML stem. The stem is assumed to have the following structure:

omlStem.0 contains the number of items

omlStem.1.attribute1, omlStem.1.attribute2 …

omlStem.2.attribute1, omlStem.2.attribute2 …

…

For each iteration of the MODEL statement, the attributes for omlStem.*n*.* are available as variables for which substitution can take place. The variable names are &variablePrefixAttribute1 ….

Note: The VARSUB command determines the value of this keyword using NCL-style substitution. Therefore, OML-style variables cannot be used to specify the name of the stem variable to use.

slproc=selectionListServiceProc

Indicates that the specified CAS selection list service procedure is to be invoked with CALLTYPE=GETALL. The data returned in the &$SL* variables is then available for substitution. If all the &$SL* variables and the iteration variable is set to the number of the current iteration, the MODEL statement iterates &$SLDTOTRECS times.

escape=*char*

Specifies a variable substitution escape character. The result is to parse each parameter on the escape character, do variable substitution on each chunk, and then concatenate them together. This allows unintentionally invalid variable names to be processed. For example, escape=\ parm1=hello&fred\world results in parm1=hello whole world (assuming that &fred contains whole). This processing is only valid for the data parameter.

data=serviceProcData

Allows you to specify any data to the invoked selection list service procedure. The data is passed to the procedure as &$SLDATA.

queue=queueName

This option indicates that the variable data to use in the MODEL statement will arrive in the INTCMD environment. This option allows a procedure that was called using the CALL statement to start an asynchronous procedure that will queue data to the ESP processor.

A queue must have been defined using the ESPQueueStart function before the MODEL statement is specified. An asynchronous process can then queue data using the ESPQueueWrite function. The queue is ended using the ESPQueueEnd or ESPQueueStop functions.

The variable names contained in the $NCL MDO that is written to the queue are extracted with the prefix variablePrefix. If the MDO queued is *not* mapped by $NCL, then the MDO is available within the model block as is.

start=*n*

Indicates that the iterations are to start at the *n*th item. This option is not valid when the data source is a queue.

Default: start=1.

count=*m*

Indicates that the iterations are to continue *m* times from the start. This option not valid when the data source is a queue.

Default: Iterate through all items.

direction={FWD|BWD}

Indicates the direction that the loop is to proceed in. BWD means to start at the end of the list and move toward the start, decrementing the counter variable.

wait={60|*nnnn*}

Specify the maximum wait time for a message to arrive on the queue. If no messages arrive within this time, then the queue is ended as if there is no more data.

Examples

```
<h1>Results List for &ipaddr</h1>
<% model mdo=$IPDATA.results from=&resStart count=20 prefix=@ip %>
   <p>&@ipcount    &@iptripTime</P>
<% endmodel %>

<% set var=qid function=$W3ESP00:ESPQueueStart %>
<% call proc=DOTRCRT share='$IPDATA.'
   parm1='ADDRESS=&ipaddr QUEUE=&qid' %>
<% model queue=&qid prefix=@ip %>
   <% flush %>
   <p><%=&@iptrResHost %>  <%=&@iptrResTime %></p>
<% endmodel %>
```

SET Statement

The SET statement allows you to create variables within the page context. Variables created using the SET statement can be used in substitution in other statements, used in substitution in the page itself (if substitution is on), or specified on the output directive.

Syntax

This statement uses the following syntax:

```
<%  set  {var=variable | mdo=mdo  map=mapname }
   { data=data   [ escape=char ] |
       function={package:function | built-in}
       [ share=variableList ] [ escape=char ]
       [ parm1=parm1 [ parm2=parm2 … [ parm25= parm25] ] ]
  }  %>
```

Keywords

This statement contains the following keywords:

var=variable

Identifies the name of the variable to create in the ESP context.

Note: The value of this keyword is determined by the VARSUB command using NCL-style substitution. Therefore, OML-style variables cannot be used to specify the name of the variable being set.

mdo=mdo

Identifies the name of the MDO to create in the ESP context. It must only specify the root stem of the name.

Note: The value of this keyword is determined by the VARSUB command using NCL-style substitution. Therefore, OML-style variables cannot be used to specify the name of the variable being set.

map=mapname

Identifies the map to use for the new MDO. This operand *must* be specified if MDO= is specified and *must not* be specified if VAR= is specified.

data=data

Specifies the data to be set into the variable. The data is passed through substitution using the substitution character specified or defaulted.

function=package:function|built-in

Specifies that the variable is to be assigned the data returned by the function identified. You can specify either your own package plus function or use one of the built-in functions (see page 90) supplied with ESP.

share=variableList

Identifies the variables to be shared to the called procedure. By default, no variables are shared. The format must be a list of names separated by commas. If a generic set of variables is to be shared, then specify a prefix followed by an asterisk. To share just a stem or MDO, then follow the name with a period. Do not specify ampersands before the names.

escape=char

Specifies a variable substitution escape character. The end result is to parse each parameter on the escape character, do variable substitution on each chunk, and then concatenate them together. This allows unintentionally invalid variable names to be processed. For example, escape=\ parm1=hello&fred\world results in parm1=hello whole world (assuming that &fred contains whole). This processing is only valid for the data and PARM parameters.

parm1= to parm25=

These keywords identify parameters to be passed to the function that is to be invoked. The parameters that you identify here must be parameters that you can use with the SETVARS command.

Examples

```
<%  set var=title data='&zprodnam : NetMaster for TCP/IP - IP Node=&ipaddr' %>
<%  set var=ipaddr data=&Request.QueryString.ipaddr %>
<%  set var=qid1 function=$W3ESP00:ESPQueueStart %>
<%  set var=resCnt data=&$ipdata.results.{&index}.count %>
<%  set var=errorCnt function=$IPPKG:SNMPGetSingleAttribute parm1=&ipaddr
   parm2='oid=1.2.3.4.56.1.45.1.0' %>
```

START Statement

The START statement allows you to start an OML or NCL procedure to perform asynchronous processing. The identified NCL or OML procedure is started, sharing the variables identified.

The following variables are set for use in other scripting statements:

&startRetcode

Indicates if the start was successful. 0 means the process was successfully started; otherwise it will be 8.

&startSysmsg

Contains a message indicating the error if startRetcode is not 0 or an informational message if startRetcode is 0.

&startNCLID

Contains the NCL ID of the process that was started.

Syntax

This statement uses the following syntax:

```
<%  start  proc=[package:]procedure  [ share=variableList ] [ escape=char ]
[ parm1=parm1 [ parm2=parm2 … [ parm25= parm25] ] ] %>
```

Keywords

This statement contains the following keywords:

proc=[package:]procedure

Identifies the name of the OML or NCL procedure to start.

share=variableList

Identifies the variables to be shared to the started procedure. By default, no variables are shared. The format must be a list of names separated by commas. If a generic set of variables is to be shared, then specify a prefix followed by an asterisk. To share just an MDO, follow the name with a period. Do not specify ampersands before the names.

Note: You can *only* share NCL type variables; *no* stem variables or variables with long names are allowed.

escape=char

Specifies a variable substitution escape character. The end result is to parse each parameter on the escape character, do variable substitution on each chunk, and then concatenate them together. This allows unintentionally invalid variable names to be processed. For example, escape=\ parm1=hello&fred\world results in parm1=hello whole world (assuming that &fred contains whole). This processing is only valid for the PARM parameters.

Parm1= to Parm25=

These keywords identify parameters to be passed to the procedure that is to be started. The parameters that you identify here must be parameters that you can use with the SETVARS command.

Examples

```
<h1>Managing IP Node &ipaddr from &nmid</h1>
<h1>Node Address: &ipaddr</h1>
<% set var=queue1 function=$W3ESP00:ESPQueueStart %>
<% start proc=$IPCALL share='$IPDATA.,$IPGH*'
   parm1='ACTION=PINGQ'      parm2='CLASS=IPNODE'
   parm3='ADDRESS=&ipaddr'   parm4='QUEUE=&queue1' %>
<% if cond='&startRetcode ^= 0' %>
   <p>System error!<%=&startSysmsg %></p>
<% else %>
   <% model var=i prefix=@ip queue=&queue1 %>
      <p>&@iprttime
   <% endmodel %>
<% endif %>
```

SUBSTITUTE - ENDSUBSTITUTE Statements

These statements indicate whether substitution is to be specifically on or off for **all** page data between the SUBSTITUTE and ENDSUBSTITUTE statements. If substitution is on, the *all* variable references are substituted for the variable values. The variable references *must* conform to OML variable specifications. In addition, the substitution character can be specified and is effective for all data within substitutions bounds.

The substitution is stacked if multiple embedded SUBSTITUTE statements are used.

If the substitution character is changed by a processing directive, then that value is *only* valid up to the ENDSUBSTITUTION statement, after which it reverts to the character set outside the current SUBSTITUTE block.

Syntax

This statement uses the following syntax:

```
<%  substitute  [ type={off|on} ]  [ subchar=c ]  %>
page data
<% endsubstitute %>
```

Keywords

This statement contains the following keywords:

type={off|on}

Indicates whether the substitution block is forcing substitution on or off. The default is on.

subchar=c

Specifies the substitution character to use for substitution within the page data. The default is to use the character set by the last processing directive. The substitution character applies for all page data and scripting statements up to the corresponding end substitution statement.

Examples

```
<h1>Results List for &ipaddr</h1>
<% substitute subchar=? %>
   <% for var=i start=?resPos count=5 %>
      <p>Trip time ?i is ?$IPData.Results.{?i}.TripTime</p>
   <% endfor %>
<% endsubstitute %>
```

WHILE - ENDWHILE Statements

These statements provide conditional looping. The page data is repeatedly processed as long as the specified condition is true.

Syntax

This statement uses the following syntax:

```
<%  while  cond=boolexprCondition    %>
page data
<% endwhile %>
```

Keywords

This statement contains the following keywords:

cond=boolexprCondition

This is the standard input to the BOOLEXPR function. It may contain simple variables previously set by the SET statement, system variables or any variables returned by a called procedure.

Examples

```
<h1>Results List for &ipaddr</h1>
<% call proc=GETLINE share='line*' %>
<% while cond='&callRetcode = 0' %>
   <p><%=&line%></p>
   <% call proc=GETLINE share='line*' %>
<% endwhile %>
```

Variables

During execution of the ESP program, the following variables are available for use:

- Most OML system variables.
- NCL and OML global variables.
- Request.QueryString.Named.keyword—Contains the value for the last occurrence of the page parameter keyword.
- Request.QueryString.Named.keyword.0—Contains number of occurrences of the page parameter keyword.
- Request.QueryString.Named.keyword.*n*—Contains the value for the *n*th occurrences of the page parameter keyword.
- Request.QueryString.Indexed.0—Contains the total number of page parameters.
- Request.QueryString.Indexed.*n*.Name—Contains the name of the *n*th page parameter.
- Request.QueryString.Indexed.*n*.Value—Contains the value of the *n*th page parameter.

Note: All variables created in the page context must have a name that does not begin with #. All other variable names are allowed.

The following variables are built into ESP:

- &DirectiveStart = <%
- &DirectiveEnd = %>

Built-In Functions

This section describes the built-in functions of the scripting directive.

ESP#Items

This function returns the number of items in an MDO component.

Syntax

This function uses the following syntax:

```
<%=ESP#Items parm1=mdocomponent %>
<% set var=variable function=ESP#Items  parm1=mdocomponent %>
```

The first line of this syntax shows how to code the function as an output directive statement (i.e. <%=); the second line of this syntax shows how to code the function to set the value into another ESP variable.

Parameters

This function contains the following parameter:

mdocomponent

The component to be counted for items that it contains.

Returns

Numeric value.

Examples

```
<% set var=cnt function=ESP#Items parm1='$IPDATA.DEVICE.{*}' %>
```

ESPConcat

This function returns all the parameters concatenated together.

Syntax

This function uses the following syntax:

```
<%=ESPConcat [ parm1=string1 ] [ parm2=string2 ] ... [ parm25=string25 ] %>
<% set var=variable function= ESPConcat
[ parm1=string1 ] [ parm2=string2 ] ... [ parm25=string25 ] %>
```

Parameters

This function contains the following Parameter:

string1-string25

The strings to be concatenated.

Returns

String1 to 25 concatenated together.

Examples

```
<%=ESPConcat parm1=& parm2='system=&SYS.NMID'%>
```

ESPCondO

This function conditionally returns one of the values passed to it.

Syntax

This function uses the following syntax:

```
<%=ESPCond0 parm1=condition [ parm2=truevalue ] [ parm3=falsevalue ] %>
<% set var=variable function= ESPCond0
parm1=condition [ parm2=truevalue ] [ parm3=falsevalue ]%>
```

Parameters

This function contains the following parameters:

condition

The condition to be tested.

truevalue

If the condition is true, then this is the value returned.

falsevalue

If the condition is either invalid or false, then this value is returned.

Returns

Value of parm2 or parm3, depending on condition.

Examples

```
<%=ESPCond0 parm1='"&lastDevice" = "&$IPDATA.DEVICE.{&IDX}.NAME"' parm2='ditto'
parm3=&$IPDATA.DEVICE.{&IDX}.NAME %>
```

ESPEval

This function performs arithmetic evaluation of the passed expression. This allows pages to perform simple arithmetic.

Syntax

This function uses the following syntax:

```
<%=ESPEval parm1=expression parm2=realexpr %>
<% set var=variable function=ESPEval { parm1=expression |  parm2=realexpr } %>
```

Parameters

This function contains the following parameters:

expression

Arithmetic expression that will be evaluated with CONTROL INTEGER.

realexpr

Arithmetic expression to be evaluated with CONTROL REAL.

Returns

Numeric result of expression.

Examples

```
<% set var=cnt function=ESPEval parm1='&cnt + 1' %>
```

ESPFormat

This function performs formatting on a number. This allows pages to format the results of arithmetic expressions into displayable formats.

Syntax

This function uses the following syntax:

```
<%=ESPFormat parm1=number
[ parm2=before ] [ parm3=after ] [ parm4=exppos ] [ parm5=exptrig]
[ parm6={R|T} ] [ parm7={S|E} ] %>
<% set var=variable function=ESPFormat parm1=number
[ parm2=before ] [ parm3=after ] [ parm4=exppos ] [ parm5=exptrig]
[ parm6={R|T} ] [ parm7={S|E} ] %>
```

Parameters

This function contains the following parameters:

number

Number to be formatted.

before

Number of digits before decimal point.

after

Number of digits after decimal point.

exppos

Number of exponent digits.

exptrig

Number of digits to trigger exponent notation.

{R|T}

Rounding or truncation.

{S|E}

Exponent format: Scientific or Engineering.

Returns

Numeric formatted number.

Examples

```
<% set var=Height function=ESPFormat parm1=&Height parm3=2%>
```

ESPHTMLQuote

This function allows text to be included into an HTML document that will not be interpreted for HTML tags.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPHTMLQuote  parm1=string %>
<%=ESPHTMLQuote  parm1=string %>
&resultVar = $W3ESP00:ESPHTMLQuote(string)
```

Parameters

This function contains the following parameters:

string

String to be HTML quoted.

Returns

HTML quoted string.

Examples

```
<%=ESPHTMLQuote parm1=&textData %>
```

ESPJSQuote

This function allows text to be included as a quoted JavaScript string that does not cause a syntax error.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPJSQuote  parm1=string %>
<%=ESPJSQuote  parm1=string %>
&resultVar = $W3ESP00:ESPJSQuote(string)
```

Parameters

This function contains the following parameters:

string

String to be quoted.

Returns

JavaScript quoted string.

Examples

```
writeLogMessage(<%=ESPJSQuote parm1=&@msgtext&i%>);
```

ESPList

This function strings together the identified elements of an array into a single string with the specified string between each element.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPList
parm1=separator
{ parm2=mdoPrefix.{*}.mdoSuffix |
   parm3=stemPrefix.*.stemSuffix |
   parm4=varPrefix }
[ parm5=fromRange ] [ parm6=toRange ]
[parm7=quoting  [parm8=quoteScope] ] [ parm9=REVERSE ]  %>
```

Parameters

This function contains the following parameters:

parm1=separator

String to place between elements of the list.

parm2=mdoPrefix.{*}.mdoSuffix

Indicates that the variable source is an MDO. The MDO is assumed to have the structure: mdoPrefix.{*}.mdoSuffix.

Each item (mdoSuffix) of the list mdoPrefix.{*} is strung together with the separator between each item.

parm3=stemPrefix.*.stemSuffix

Indicates that the variable source is an OML stem. The stem is assumed to have the following structure:

- stemPrefix.0—number of items in structure
- stemPrefix.1.stemSuffix—first item in list
- stemPrefix.2.stemSuffix—second item …

Each item of the list is strung together with the separator between each item.

parm4=varPrefix

Indicates that the variable source is a variable prefix. The array structure is assumed to be of the form:

- varPrefix1—first item
- varPrefix2—second item …

Each item of the array is strung together with the separator between each item. The list is assumed to terminate at the first null variable reference.

parm5=fromRange

Starting item of the list. Default is 1.

parm6=toRange

Ending item of the list. If not specified, then the whole list is assumed. Null elements of the list result in consecutive separators.

parm7=quoting

Allows the specification of particular quoting for each item of the list. If not specified, then no quoting is done. The valid quoting options are:

- HTMLQuote— Quote the item for HTML.
- JSQuote—Quote the item for JavaScript.

parm8=quoteScope

This parameter is only valid when quoting is specified. It indicates the scope of the quoting. Specify one of the following:

- ITEM—Quoting is only done for the list items.
- SEP—Quoting is only done for the item separator.
- ALL—Quoting is done for both item and separator.

parm9=REVERSE

If specified, then tells ESPList to process the items in reverse order.

Returns

String.

Examples

```
[assign the value for TD in your book]<%=ESPList parm1='</td>[assign the value for TD in your book]'  parm4=$SLRID %> </td>
```

results in:

```
[assign the value for TD in your book]first item</td>[assign the value for TD in your book]second item</td>third item</td>
```

ESPQueueStart

This function defines and activates an ESP queue. Once started, any process can queue messages to the queue by using the ESPQueueWrite function. This function must be called to start queues that will be used in the MODEL statement.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPQueueStart %>
```

Parameters

ESPQueueStart contains no parameters.

Returns

Queue ID that can be used in other ESP statements or functions.

Examples

```
<% set var=qid function=ESPQueueStart %>
```

ESPQueueStop

This function deletes an ESP queue. Any data currently on the queue will *not* be processed. If a MODEL statement specifies the queue, then it will be bypassed as if the queue were empty.

Syntax

This function uses the following syntax:

```
&resultVar = $W3ESP00:ESPQueueStop(queueId)
```

Parameters

This function contains the following parameters:

queueId

Queue ID of existing queue.

Returns

0 if queue was not previously defined, or 1 if it was stopped.

Examples

```
   <% start proc=$CAPXV2Z parm1='QUEUE=&queue1' %>
   <% if cond='&startRetcode ^= 0' %>
      <% set var=x function=ESPQueueStop parm1=&queue1 %>
   <% endif %>
or
Call GetPingData
If &SYS.Retcode ^= 0 Then
   Do
      &tmp = $W3ESP00:ESPQueueStop(&#IP#QID)
      Retsub 8
   End
```

ESPQueueEnd

This function ends an existing queue. All messages currently on the queue are processed by a model statement that specifies the identified queue.

Syntax

This function uses the following syntax:

```
&resultVar = $W3ESP00: ESPQueueEnd(queueId)
```

Parameters

This function contains the following parameters:

queueId

Queue ID of existing queue.

Returns

0 if queue was not previously defined or 1 if it was ended.,

Examples

```
         Call GetPingData
If &SYS.Retcode = 4 Then
   Do
      &tmp = $W3ESP00:ESPQueueEnd(&#IP#QID)
      Retsub 0
   End
```

ESPQueueWrite

This function writes a message to an existing queue. All messages currently on the queue will be processed by a model statement that specifies the identified queue.

Syntax

This function uses the following syntax:

```
resultVar = $W3ESP00:ESPQueueWrite(queueId {,mdoName | ,,prefix | ,,,keywords })
```

Parameters

This function contains the following parameters:

queueId

Queue ID of existing queue.

mdoName

Name of MDO that is to be queued. The MDO can be mapped by any map and if specified, then the remaining parameters are ignored. The MDO must be shared into the function. If the map is $NCL when the MDO is processed in the ESP, then the attributes are extracted using the prefix that was specified on the ESP verb. If the map is NOT $NCL, then the MDO is available as is and no special processing is done.

prefix

Variable prefix that identifies all the variables that are to be queued. If specified, then the remaining parameters are ignored. The variables must be shared into the function.

keywords

A list of keywords and values that are to be queued. These keywords must be keywords that you can use with the SETVARS command. If specified, then the remaining parameters are ignored. No variables need to be shared into the function.

Returns

0 if queue is no longer defined, or 1 if the data was successfully written to the queue.

Examples

```
      Control ShrVars=(&$IPDATA.)
      Call GetPingData
Do While &SYS.Retcode = 0 and $W3ESP00:ESPQueueWrite(&#IP#QID,'$IPDATA')
   Call GetPingData
End
&tmp = $W3ESP00:ESPQueueEnd(&#IP#QID)
```

ESPSpace

This function simulates the OML space function.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPSpace
parm1=string [parm2=n] [parm3=pad] %>
```

Parameters

This function contains the following parameters:

parm1=string

String to space.

parm2=n

Number of spaces between words.

parm3=pad

Pad character to use between words.

Returns

Spaced out string.

Examples

```
<% set var=tmpid function=ESPSpace
   parm1='&REQUEST.QueryString.Named.ipaddr : &REQUEST.QueryString.Named.ipport'
 %>
```

ESPSubstr

This function simulates the OML substr function.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPSubstr
parm1=string parm2=n
[parm3=length] [parm4=pad] [parm5=length2] %>
```

Parameters

This function contains the following parameters:

parm1=string

String that is the source of the requested substring.

parm2=n

Start position.

parm3=length

Length of substring.

parm4=pad

Pad if string too small.

parm5=length2

Maximum length (ignores length and pad).

Returns

Substring requested.

Examples

```
<% set var=tport function=ESPSubstr
parm1='&REQUEST.QueryString.Named.ipaddr'
parm2=17 %>
```

ESPSysparm

This function simulates the OML sysparm function.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPSysparm
parm1=sysparm %>
```

Parameters

This function contains the following parameters:

parm1=sysparm

The name of the system parameter whose value is to be returned.

Returns

Sysparm value requested.

Examples

```
<% set var=pwr function=ESPSysparm
parm1='PWRETRY' %>
```

ESPTranslate

This function simulates the OML translate function.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPTranslate
parm1=string [parm2=tableo] [parm3=tablei] [parm4=pad] %>
```

Parameters

This function contains the following parameters:

parm1=string

String to be translated.

parm2=tableo

Output table of characters to use for the translation. This can be specified in hexadecimal as x'hhhh'. The x and quotes must be left as is and in lower case, and the hhhh string must be valid hex characters.

parm3=tablei

Input table of characters to use for the translation. This can be specified in hexadecimal as x'hhhh'. The x and quotes must be left as is and in lower case, and the hhhh string must be valid hex characters.

parm4=pad

Specifies what tableo should be padded to if it is smaller than tablei. This can be specified in hexadecimal as x'hhhh'. The x and quotes must be left as is and in lower case, and the hhhh string must be valid hex characters.

Returns

Translated string requested.

Examples

```
<% set var=tmp function=ESPTranslate
parm1='&REQUEST.QueryString.Named.hostname'
parm2=_ parm3='~!@#$%^&*()=+[]{}\|/?><`' parm4=_ %>
```

ESPVConcat

This function simulates the OML vconcat function.

Syntax

This function uses the following syntax:

```
<% set var=variable function=ESPVConcat
parm1=prefix parm2=fromRange parm3=toRange [parm4=padString] %>
```

Parameters

This function contains the following parameters:

parm1=prefix

Prefix of variables to be concatenated.

parm2=fromRange

Starting variable range.

parm3=toRange

Ending variable range

parm4=padString

Specifies what string to place between the data of each variable. If omitted, then nothing is placed between the variable values.

Returns

Concatenated string.

Examples

```
<% set var=tmp function=ESPVConcat parm1='$SLRDATA' parm2=1 parm3=10 %>
```

Appendix I: HFS Considerations

This section contains the following topics:

HFS Considerations

WebCenter SDK includes sample code distributed and installed by SMP/E as hierarchical file system (HFS) files.

Note: You can also use ZFS files.

Installation and implementation of WebCenter SDK is an optional step that may be added after your product region has been set up and you are familiar with its operation. You should install the WebCenter SDK HFS code when you install WebCenter SDK.

Note: You should have UNIX superuser authority level to perform the steps in this task.

If you are unfamiliar with z/OS HFS files, see the chapter "An Introduction to the Hierarchical File System" in the IBM *UNIX System Services User's Guide* for your version of z/OS.

More information:

Provide an HFS directory for WebCenter SDK Use (see page 16)

Using an Existing File System

You may choose to install the WebCenter SDK sample code into a new or existing directory in an existing file system, for example:

```
/u/users/ca
/cai
/usr/lpp
```

If the directory does not exist, you will need to create it.

Using a New File System

If you do not want to install the WebCenter SDK sample code into one of your existing file systems, you will need to allocate a new physical file system and mount it permanently.

Your UNIX Systems Services administrator can tell you what your HFS data set naming standards are and what mount point to use for the new file system.

For detailed information about how to allocate and mount a file system, see the IBM *UNIX System Services Planning* manual.

Allocate the File System

You can use the following sample JCL code as a guide to allocate a file system:

```
//*-------------------------------------------------------------------*
//*         ALLOCATE THE HFS                                          *
//*-------------------------------------------------------------------*
//ALLOC    EXEC PGM=IEFBR14
//HFS      DD DSN=?HFSQUAL.NETMASTR.HFS,
//            DISP=(NEW,CATLG,DELETE),
//            DSNTYPE=HFS,
//            ?VOL=SER=XXXXXX or STORCLAS=X,MGMTCLAS=X
//            SPACE=(CYL,(100,50,5)),DCB=DSORG=PO
```

Mount the File System

You can use the following sample JCL code as a guide to temporarily mount a file system. The file system must be mounted read/write on the system where your generated installation JCL will be run. You may also issue the mount command from OMVS. See the IBM UNIX System Services documentation for details.

```
//*-------------------------------------------------------------------*
//*         MOUNT THE HFS TO MOUNTPOINT                               *
//*                                                                   *
//* Instructions: 1) Change ?HFSDSN to the name of the HFS file       *
//*                  that was allocated.                              *
//*                                                                   *
//*               2) Change /?cai to the mount point for your         *
//*                  installation                                     *
//*                                                                   *
//*               3) Update your BPXPARM to include the new mount     *
//*                  point.                                           *
//*                                                                   *
//*-------------------------------------------------------------------*
//MOUNTHFS EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=50
//SYSTSPRT DD SYSOUT=*
//SYSTSOUT DD SYSOUT=*
//SYSTSIN  DD *
 MOUNT FILESYSTEM('?HFSDSN') +
       TYPE(HFS) MODE(RDWR) +
       MOUNTPOINT('/?cai')
/*
```

Mount the File System Permanently

This file system will only remain mounted until the next IPL. To permanently mount this file, you must also update your PARMLIB BPXPRM*xx* to include the mount point, so subsequent IPLs will mount the HFS automatically.

Preparing to Install the HFS Code

When you have a new or existing file system to use, and you know what directory path name to use for the WebCenter SDK HFS files, you should review these common considerations with your UNIX Systems Services administrator.

Authorities Required

The SMP/E user who will be submitting the generated installation JCL for WebCenter SDK must have write access to the path name.

In addition, the **SMP/E program name** (not the name of the submitting user) must be defined to the security class BPX.SUPERUSER before any SMP/E installation or maintenance can be done. This ensures that SMP/E always has sufficient file system access, irrespective of who runs the SMP jobs.

See the *SMP/E for z/OS and OS/390 User's Guide* for more information.

More information:

Verify UNIX System Services Security Requirements (see page 16)

Shared Mount Points

The file system must be mounted read/write on the same system where your generated installation JCL will be run. This is so SMP/E can create the SMP target library.

A WebCenter SDK control region needs at least read and execute access to the sample code in this file system. If such a region is on a different system, this file access may be accomplished by methods such as mounting the file system as readable from multiple systems, or copying the target library files to a run time directory. This depends on your specific configuration and code management standards.

WebCenter SDK data regions do not access the WebCenter SDK HFS files.

Directories and Files Created

The generated installation JCL for WebCenter SDK creates and runs a shell script named WS<deliverylevel>MKDIR. This creates a subdirectory named nm/sdk/ws<deliverylevel>/samples under your *iia-prefix* path name. The access permissions for these subdirectories are set at 755.

For example, if your *iia-prefix* path name is /u/users/cai, the JCL creates a directory named:

```
/u/users/cai/nm/sdk/ws<deliverylevel>/samples
```

During the installation of WebCenter SDK, the access permissions for the individual files are set at 775 for executable files and 664 for non-executable files. You do not need to change these access permissions.

Note: For more information about setting and changing HFS file permissions, see IBM's *UNIX System Services User's Guide* and *Command Reference*.

Index

H

I

J

M

O

P

R

S

T

U

V

W