Enterprise Portal Client Framework (EPCF)

Release 646
Copyright

© Copyright 2004 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.
Icons in Body Text

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Caution</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
<tr>
<td>Recommendation</td>
<td></td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
</tbody>
</table>

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see Help on Help → General Information Classes and Information Classes for Business Information Warehouse on the first page of any version of SAP Library.

Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example text</em></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Cross-references to other documentation.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><strong>&lt;Example text&gt;</strong></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>
Enterprise Portal Client Framework (EPCF) ................................................................. 5
EPCF Levels .................................................................................................................. 5
EPCF API ....................................................................................................................... 6
  System API .................................................................................................................. 7
  Event API ................................................................................................................... 9
Client Data Bag API ...................................................................................................... 11
WorkProtect API ......................................................................................................... 13
Navigation API ............................................................................................................ 14
Data Session Management (DSM) API ......................................................................... 19
EPCM Proxy .................................................................................................................. 20
EPCF Configuration ....................................................................................................... 22
  Properties for EP 5.0 ................................................................................................... 24
Session Management .................................................................................................... 27
  SessInfo Object Properties ....................................................................................... 35
  Terminator Component ............................................................................................. 35
  WorkProtect Feature for EP 6.0 ................................................................................ 37
Navigation .................................................................................................................... 38
Glossary ......................................................................................................................... 40
  Automatic Server Session Termination (ASST) ........................................................ 40
Client Data Bag ........................................................................................................... 40
Client Data Channel .................................................................................................... 40
JavaScript Origin Policy ............................................................................................... 40
Namespaces ................................................................................................................... 41
**Enterprise Portal Client Framework (EPCF)**

The **Enterprise Portal Client Framework (EPCF)** provides an infrastructure for scripting used in iViews and by the Portal.

To keep the application design simple and maintain compatibility to different browsers, Web applications usually avoid scripting. However, there are tasks that make it necessary to use scripting. To name a few:

- Increase user acceptance for example with context sensitive entry helpers
- Enhance response time of the application for example through validation of input values

When a business application uses more than one iView, you need the EPCF service to transfer data between the iViews. The EPCF service provides:

- Mechanisms for *eventing* between iViews.
- A Java object, called *client data bag*, that serves as transient data buffer on the browser.
- Distributed Session Management (DSM) to enhance the performance of the server session management.

The EPCF implementation itself is based on JavaScript and Java applets.

**EPCF Levels**

The EPCF level defines which functionality of the EPCF service is available to the Web application. The EPCF level affects the size of the generated HTML data is generated, that is sent to the client. A higher level generates more HTML data. The EPCF has the following levels.

- **0: No EPCF Service**
  This level generates no JavaScript or Java applet framework functions. Communication between iViews is not possible.

- **1: JavaScript**
  This level generates framework functions for JavaScript.

- **2: JavaScript and Applet**
  This level generates framework functions for JavaScript and Java applet.

**Detailed feature list**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit softening the JavaScript Origin Policy [Page 40]</td>
<td>1</td>
<td>Enables scripting between <em>IFrames</em> from the same domain but different hosts.</td>
</tr>
<tr>
<td>System function</td>
<td>1</td>
<td>Retrieves the EPCF version, the current EPCF level and client information.</td>
</tr>
<tr>
<td>Client eventing</td>
<td>1</td>
<td>Client communication service for iViews.</td>
</tr>
</tbody>
</table>
Predefined Client Events

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Frequency</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Server Session Termination</td>
<td>1* / 2</td>
<td>Support of the SAP Internet Session Protocol (SISP) that allows session management with connectionless HTTP.</td>
</tr>
<tr>
<td>Client Data Bag</td>
<td>1* / 2</td>
<td>Buffer for the client that stores JavaScript data as long as the Portal session exists.</td>
</tr>
<tr>
<td>Client Data Channel</td>
<td>2</td>
<td>Synchronous HTTP channel between the Portal page on the Web client and the Portal Server.</td>
</tr>
</tbody>
</table>

* : A JavaScript implementation is available since Enterprise Portal version 5.0.1.3.

EPCF API

The EPCF service defines the Enterprise Portal Client Manager (EPCM) JavaScript object. With the methods of the EPCM object you can access the EPCF service functions as follows:

```javascript
EPCM.[API_method_name]( [Params]* );
```

iViews can access the EPCM object from every Portal page or IFrame. Every iView of the Portal runtime contains the EPCM object. As a result every embedded or isolated iView can use the EPCF service with the method:

```javascript
EPCM.subscribeEvent(, eventName, eventHandler );
```

For details on namespaces, see the glossary chapter Namespaces [Page 41]

Every EPCM object stores the data it receives and delegates them automatically to the registered EPCM objects.

With EPCF level 2 every Portal page and as a result in all isolated iViews, contain a Java applet. The applet serves as a class factory for methods that return references to the intrinsic classes. These classes are implemented as Singletons so every class has only a single instance in a Portal session, even when iViews are reloaded. The applet object is instantiated inside every Portal page frame.

The EPCF API has following parts:

- System API
  The system API [Page 7] provides methods to get the version and the level of the EPCF service and information about the client.

- Event API
  The event API [Page 9], which allows iViews to communicate with each other and with the Portal environment itself on the. This is done by using JavaScript functions on the client which are invoked on client events like onload, onclick and so on.

- Client Data Bag API
  The client data bag API [Page 11] offers a transient data buffer for iViews. The data remains in the buffer are saved even after the iView or the whole Portal page is reloaded. Depending on the EPCF level, following storage mechanisms are used:

  o EPCF level 1 - JavaScript
Values are stored as cookies in the browser.

- EPCF level 2 – Java Applet
  Values are stored as a Java class attribute.

- WorkProtect API
  The WorkProtect API [Page 13] provides the infrastructure for handling unsaved data in a stateful application.

- Navigation API
  The navigation API [Page 14] provides methods to navigate in the Portal.

- Distributed Session Management (DSM) API
  The DSM API [Page 19] is used for session management. The DSM uses this API to access the properties of the SessInfo object.

- EPCM Proxy
  The EPCM proxy API [Page 20] allows the EPCF functionality in Portal applications that are rendered in their own IFrame (for example, ITS-based applications and BSP).

### System API

With the EPCF system API you have access to the settings of the EPCF service.

#### EPCM.getVersion()

This method returns the current framework version as type number.

**Usage**

```javascript
<script language="JavaScript">
    var version = EPCM.getVersion();
</script>
```

#### EPCM.getLevel()

This method returns the current EPCF level as type number.

The EPCF level defines which EPCF services are available. The Portal application has to take care, that it uses services which are available at the current EPCF level.

**Usage**

```javascript
<script language="JavaScript">
    if (EPCM.getLevel() >= 2) {
        EPCM.storeClientData( "urn:com.sap:myObjects",
                                "person", "Albert Borland" );
    }
</script>
```
EPCM.getUAType()
This method returns the client type as type number.
Usually this method will be used together with the method getUAVersion() and getUAPlatform(). The return value can be compared to predefined EPCM-constants:

- EPCM.MSIE
- EPCM.NETSCAPE
- EPCM.MOZILLA
- EPCM.OPERA
- EPCM.NOKIA
- EPCM_UP
- EPCM_ERICSSON
- EPCM_MSPIE
- EPCM_PALM
- EPCM.OTHER

Usage

```javascript
<SCRIPT language ="JavaScript">
    if(EPCM.getUAType() == EPCM.MSIE){/*coding for IE*/ }
    if(EPCM.getUAType() == EPCM.MOZILLA){/*coding for Mozilla*/ }
</SCRIPT>
```

EPCM.getUAVersion()
This method returns version of the client as type number.

Usage

```javascript
<SCRIPT language ="JavaScript">
    if(EPCM.getUAType() == EPCM.MSIE){
        if(EPCM.getUAVersion() == 5.0){/*coding for MSIE 5.0 */
    }
    if(EPCM.getUAVersion() > 5.5){/*coding for MSIE 5.0+*/ }
}
</SCRIPT>
```

EPCM.getUAPlatform()
This method returns the platform on which the client is running as type number.
The return value can be compared to predefined EPCM constants:

- EPCM.NT_PLATFORM
- EPCM.WIN_PLATFORM
- EPCM.MAC_PLATFORM
- EPCM.LINUX_PLATFORM
- EPCM.WAP_PLATFORM
- EPCM.PDA_PLATFORM
- EPCM.OTHER_PLATFORM

Usage

```javascript
<SCRIPT language ="JavaScript">
    if(EPCM.getUAPlatform() == EPCM.LINUX_PLATFORM){
```
getInstanceId() Method
This method returns an unique EPCF instance as type String.
The method is used by the EPCF core to distinguish the pages after a page refresh.

Usage

```<SCRIPT language ="JavaScript">document.write("EPCMInstanceId = " + EPCM.getInstanceId() );</SCRIPT>```

EPCM.getUniqueWindowId()
This method returns an unique identifier of the IFrame as type String.
You can use this method to append the returned IFrame identifier string to the name you use to define a client data bag. This creates a client data bag that can only be accessed by a specific IFrame.

⚠️
The method returns null when the object returned by window.top is not accessible because of security reasons (JavaScript origin policy [Page 40]).

Usage

```<SCRIPT language ="JavaScript">
document.write("WindowId = " + EPCM.getUniqueWindowId() );
...
ECPM.storeClientData(
"com.sap.portal:test",EPCM.getUniqueWindowId()+"Item",myItem);
...
</SCRIPT>```

Event API
The EPCF event API provides methods for the event handling on the client.
EPCM.subscribeEvent( nameSpace, eventName, eventHandler )

This method assigns an event handler to the specified event.

The method sets the event handler to the subscription list for the event defined by the `nameSpace` and the `eventName`. The combination of `nameSpace`, `eventName` and `eventHandler` must be unique. It is not possible to register the same event handler to several events. See chapter Namespaces [Page 41] for more details.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameSpace</td>
<td>String</td>
<td>URN [Page 41] of the event namespace.</td>
</tr>
<tr>
<td>eventName</td>
<td>String</td>
<td>The event name you subscribe to. You can use an asterisk (*) to subscribe for all events of this name-space.</td>
</tr>
<tr>
<td>eventHandler</td>
<td>Function</td>
<td>Reference to the event handler.</td>
</tr>
</tbody>
</table>

Usage

```javascript
function onWakeup( eventObj ) {
    alert( "got a wakeup call from " +
            eventObj.sourceId + ": " + eventObj.dataObject );
}
...
EPCM.subscribeEvent( "urn:com.sap:alarmClock",
                    "morningCall", onWakeup );
...
EPCM.subscribeEvent( "urn:com.sap:alarmClock", ",", onWakeup );
```

EPCM.raiseEvent( nameSpace, eventName, dataObject [, sourceld])

This method raises the event defined by `nameSpace` and `eventName`. The EPCF service calls all event handlers which are registered for this event and passes the event object on to the event handler.

The `event object` is created by the EPCF service whenever an event is raised. It combines the `dataObject`, the `eventName` and the `sourceld` (which may be null) to a single argument for the event handler.

Parameter Description
### Usage

```
<SCRIPT language ="JavaScript">
...
EPCM.raiseEvent( "urn:com.sap:alarmClock", "morningCall",
       "Good morning ladies and gentlemen", "iView_0815" );
...
<SCRIPT>
```

### Client Data Bag API

The EPCF **client data bag** API provide methods to store data in a transient data buffer on the client.

**EPCM.storeClientData(nameSpace, name, value)**

This method saves data in `value` under a key. The key is generated by combining the parameters `nameSpace` and `name`. If the key already exists, the stored data will be overwritten.

#### Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameSpace</td>
<td>String</td>
<td>URN [Page 41] for the first part of the key under which the data is stored. nameSpace will be combined with name.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>This name, combined with namespace, creates the key under which the data is stored.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>Data to be stored.</td>
</tr>
</tbody>
</table>

⚠️ The parameter value must be of type `String`. Primitive data types will be converted to `String`, complex data types and references are not supported. This restriction is necessary to guarantee that the **client data bag** functionality is working in a JavaScript environment using the browser cookies for clients that have no Java support.
### Usage

```javascript
<SCRIPT language ="JavaScript">
    var selectedPerson = "Tim Taylor"
    EPCM.storeClientData( "urn:com.sap.myObjects", "person",
    selectedPerson );
</SCRIPT>
```

### EPCM.loadClientData(nameSpace, name)

This method returns the data stored under the specified key as `String`. The key is generated by a combination of the parameters `nameSpace` and `name`. If the key does not exist, the method returns `null`.

**Parameter Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameSpace</td>
<td>String</td>
<td>URN [Page 41] for the first part of the key from which the data is reloaded. nameSpace will be combined with name.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>This name, combined with namespace, is the key from which the data is reloaded.</td>
</tr>
</tbody>
</table>

```javascript
<SCRIPT language ="JavaScript">
    var person=EPCM.loadClientData("urn:com.sap.myObjects", "person");
    if ( person != null ){
        /* process person */
    }
</SCRIPT>
```

### deleteClientData(nameSpace, name)

This method deletes the data stored under the specified key and the key itself. The key is generated by a combination of the parameters `nameSpace` and `name`.

**Parameter Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameSpace</td>
<td>String</td>
<td>URN [Page 41] for the first part of the key which is. nameSpace will be combined with name.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>This name, combined with namespace, is the key from which is deleted.</td>
</tr>
</tbody>
</table>

```javascript
<SCRIPT language ="JavaScript">
    // Code for deleteClientData
</SCRIPT>
```
EPCM.deleteClientData( "urn:com.sap.myObjects", "person" );
</SCRIPT>

**WorkProtect API**

The EPCF *WorkProtect* API provides methods to get the status about unsaved data on the page.

**EPCM.setDirty(indicator)**

This method sets the status of the *dirty indicator* to *true* or *false*.

**Parameter Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>indicator</td>
<td>boolean</td>
<td>Status of the dirty indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>true</em>: Page contains unsaved data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>false</em>: Page is clean – no unsaved data.</td>
</tr>
</tbody>
</table>

**Usage**

```javascript
<SCRIPT language="JavaScript">
if  (storedValue != enteredValue){
    changedData["DataKey"] = enteredValue;
    EPCM.setDirty( true );
}

// do other actions ...

storeArrayToDataBase(changedData);
;EPCM.setDirty( true );

</SCRIPT>
```

**EPCM.getDirty()**

This method returns the current setting of the *dirty indicator* as type *boolean*. The *WorkProtect* feature uses this method to get the *dirty indicator* for the entire Portal page.

**Usage**

```javascript
<SCRIPT language="JavaScript">
var isDirty = EPCM.getDirty( );
alert("Component " + (isDirty) ? "clean" : "dirty" );
</SCRIPT>
```
**EPCM.getGlobalDirty()**

This method returns the current setting of the dirty indicator as type boolean. The difference to the getDirty() method is, that the getGlobalDirty() method checks the dirty flag of all iViews on the page and returns a true value if at least one of the iViews had a dirty flag set to true, and false otherwise.

**Usage**

```javascript
<SCRIPT language ="JavaScript">
var isDirty = EPCM.getGlobalDirty();
alert("One component " + (isDirty) ? "clean" : "dirty");
</SCRIPT>
```

**Navigation API**

The Navigation API provides the methods to navigate inside the Portal. Refer to chapter Enterprise Portal Navigation [External] for more details about the navigation service.

**Absolute Navigation**

For an absolute navigation you have to know the full path name of the component. The full path name starts at the navigation hierarchy root node all the way to the navigation target.

**EPCM.doNavigate(String target, [int mode, String winFeat, String winName, int history, String targetTitle, String context])**

This method triggers the absolute navigation on the client. By default, when the parameter mode is not specified, the dirty indicator of the component is checked by the WorkProtect feature and the target is opened in a new window or on the current Portal page depending on the result of the check.

The optional parameters are new in EP 6.0 and can be used when the target is displayed in the new window.

The method always returns false, for easier use with event handlers like onClick.

**Parameter Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>String</td>
<td>Navigation target that corresponds to the location in PCD or another structure (see details below)</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0 or not specified: Depending on the setting of the WorkProtect feature the target is opened in a new window or on the current desktop. 1: Open target in a new window, with no a Portal header and navigation bar.</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2:
Open target in a new window, with a Portal Header and navigation bar.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>winFeat (optional)</td>
<td>String</td>
<td>Window feature string when the target is to be opened in the new window. This is a comma separated list of features with no blanks that has the same syntax as the JavaScript method <code>window.open</code>. Example: &quot;width=400,height=500&quot;.</td>
</tr>
<tr>
<td>winName (optional)</td>
<td>String</td>
<td>Window title for when the target is opened in a new window.</td>
</tr>
<tr>
<td>history (optional)</td>
<td>int</td>
<td>history mode 0: Track history entries and allow duplicates. 1: Track history entries and do not allow duplicates. 2: Do not track history entries.</td>
</tr>
<tr>
<td>targetTitle</td>
<td>String</td>
<td>Title for the page title bar. In case the navigation target is sent through an integrator, the title will be the integrator title. You can specify a specific title for this navigation and optional for the history entry.</td>
</tr>
<tr>
<td>Context</td>
<td>String</td>
<td>Navigation context URL.</td>
</tr>
</tbody>
</table>

**Usage**

```javascript
<SCRIPT language ="JavaScript">
// navigate.
EPCM.doNavigate(
    "ROLES://portal_content/folder1/role1/workset1/iView111"
)
</SCRIPT>

<A HREF="myLink"
   onclick="return EPCM.doNavigate
              ('ROLES://portal_content/folder1/role1/workset1/iView111')">
This is an HTML Link
</A>
```

**Result**
This starts the navigation to the iView 111 under role 1.
Relative Navigation

For a relative navigation you specify the relative location of the navigation target to the current navigation node.

**EPCM.doRelativeNavigate(String basenodename, int level, List pnamesList, int mode, String winFeat, String winName, int history, String targetTitle, String context)**

This method triggers the relative navigation on the client. You have to know the location of the navigation target relative to the current node. That the navigation model can create an absolute path, you have to provide at least the `level` or the `pnamesList` parameters in addition to the `basenodeName` parameter.

The optional parameters are new in EP 6.0 and can be used when the target is displayed in the new window.

The method always returns `false`, for easier use with event handlers like `onClick`.

**Parameter Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basenodeName</td>
<td>String</td>
<td>Current presented URL – current node.</td>
</tr>
<tr>
<td>level</td>
<td>int</td>
<td>Number of hierarchy levels to step up.</td>
</tr>
<tr>
<td>pnameslist</td>
<td>List</td>
<td>A list with all the atomic names of the children nodes,</td>
</tr>
</tbody>
</table>
relative to the node that has been reached by stepping up the number of level.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| mode   | int  | 0 or not specified:  
Depending on the setting of the WorkProtect feature the target is opened in a new window or on the current desktop.  
1: Open target in a new window, with no a Portal header and navigation bar.  
2: Open target in a new window, with a Portal Header and navigation bar. |

<table>
<thead>
<tr>
<th>Window Feature (optional)</th>
<th>Description</th>
</tr>
</thead>
</table>
| winFeat | String | Window feature string when the target is to be opened in the new window. This is a comma separated list of features with no blanks that has the same syntax as the JavaScript method window.open.  
Example: "width=400,height=500". |

<table>
<thead>
<tr>
<th>Window Name (optional)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>winName</td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>History (optional)</th>
<th>Description</th>
</tr>
</thead>
</table>
| history | int | history mode  
0: Track history entries and allow duplicates.  
1: Track history entries and do not allow duplicates.  
2: Do not track history entries. |

<table>
<thead>
<tr>
<th>Target Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetTitle</td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>String</td>
</tr>
</tbody>
</table>

Usage

```javascript
<SCRIPT language="JavaScript">
// navigate.
EPCM.doRelativeNavigate(
    "ROLES:// portal_content/role3/Folder32/Folder33", 2,
    {"page3"}, ..., ..., ...);
</SCRIPT>

<A HREF="myLink"
onclick="return EPCM.doRelativeNavigate
    ('ROLES:// portal_content/role3/Folder32/Folder33', 2,
    {"page3"}, ..., ..., ...)">
This is an HTML Link
```
Result

This starts the navigation from folder 33 under role 3 to page 3 under role 3. See figure 1.

If you do not provide the first parameter, the current navigation node, the navigation model will find the current navigation node itself and add it to the path automatically.

Object Based Navigation

The object based navigation allows navigation based on actual business objects from productive back end systems. The object based navigation is based on the concept of business objects, that perform certain operations and iViews that can be declared as implementors of these operations. Every operation has a priority. When choosing the link of the object based navigation, the operation with the highest priority, that has an implementing iView in the user role (the default), will be executed.

EPCM.doObjBasedNavigate(String systemAlias, String businessObjName, String objValue, String operation)

This method allows navigating in a context environment, without a specific URL for a navigation target. For more details, see the Object Base Navigation description.

The method always returns false, for easier use with event handlers like onClick.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemAlias</td>
<td>String</td>
<td>The system alias of the business object.</td>
</tr>
<tr>
<td>businessObjName</td>
<td>String</td>
<td>Business object name for which the operation was defined.</td>
</tr>
<tr>
<td>objValue</td>
<td>String</td>
<td>Any data that has to be transferred to the navigation target when the visualization iView represents relative data. The objValue can be any string that is added to the URL of the navigation target (after the &quot;?&quot; separator) and the target iView can access the objValue via the iView request object.</td>
</tr>
<tr>
<td>operation</td>
<td>String</td>
<td>Operation that should be performed when the business object has more than one operation.</td>
</tr>
</tbody>
</table>

The parameters businessObjName and systemAlias define the MetaMatrix name of the object based navigation business object.

If the object based navigation uses relation resolving, the objValue parameter is used to transfer the HRNPLink.
Data Session Management (DSM) API

The DSM uses this API to access the properties of the SessInfo object. See chapter Session Management [Page 27] for more details.

EPCM.DSM.init (String url)

This method Initializes the DSM and registers the URL at the Terminator component.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>URL that is registered at the Terminator component.</td>
</tr>
</tbody>
</table>

EPCM.DSM.processSession (sessInfo)

This method is the entry point for the session management (handler for SAPWP_receiveSessInfo).

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessInfo</td>
<td>Object</td>
<td>SessInfo object.</td>
</tr>
</tbody>
</table>

EPCM.DSM.notifyMonitor (sessInfo)

This method notifies the DSM when the new SessInfo object is available.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessInfo</td>
<td>Object</td>
<td>SessInfo object.</td>
</tr>
</tbody>
</table>

EPCM.DSM.registerSession (sessInfo)

This method registers the SessInfo object.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessInfo</td>
<td>Object</td>
<td>SessInfo object.</td>
</tr>
</tbody>
</table>

EPCM.DSM.removeSessionByGUSID (gusid)

This method removes the SessInfo object associated with the Global Unique Session ID (GUSID) from the register.

Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gusid</td>
<td>String</td>
<td>GUSID.</td>
</tr>
</tbody>
</table>

EPCM.DSM.terminateByGUSID (gusid)

This method terminates the sessions associated with GUSID and removes them from the register.
### Parameter Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gusid</td>
<td>String</td>
<td>GUSID.</td>
</tr>
</tbody>
</table>

### EPCM.DSM.terminateAll ()

This method terminates all registered sessions and clears the register.

### EPCM.DSM.getAllToArray ()

This method makes a copy of all active SessInfo objects in a JavaScript Array object. The SessInfo objects can be used and processed by the client application.

### EPCM.DSM.getSize ()

This method returns the number of registered SessInfo objects in the DSM.

---

### EPCM Proxy

The EPCM proxy allows the EPCF functionality in Portal applications that are rendered in their own IFrame (for example, ITS-based applications and BSP).

#### “Reload” Function and Event-Subscription

The EPCF event methods are used by function reference. Since the references are kept across the IFrame borders, the references become invalid whenever the IFrame content is reloaded. To solve this problem, you have to use the second signature of the EPCM.subscribeEvent method that references to the current window object.

```javascript
EPCM.subscribeEvent(nameSpace, name, window_reference, method_name)
```

External applications (for example, BSP, BW-Reports) are rendered in their own IFrame. The EPCM Object therefore can convert the event handler registration from [window_reference,method_name] to [iframe_name,method_name]. With this conversion the method keeps the name and not the object/method reference. When the IFrame content is reloaded now, the iframe_name and method_name are still valid and the event handler, that is located inside the IFrame, can be called from the EPCM event manager outside the IFrame using the following call:

```javascript
window.frames[iframe_name][method_name](event_data)
```

### IncludeProxy

To simplify the implementation, the EPCMPROXY object is provided that serves as the proxy. The EPCF calls within the IFrame are delegated by the proxy layer to the upper EPCF layer. So instead of EPCM calls you use EPCMPROXY. THE EPCMPROXY object is in the JavaScript file epcfproxy.js that comes with the Portal and has to be included into your Portal application.

### Usage

Usage of the JavaScript file epcfproxy.js in a HTML page:

```html
<HTML>
<HEAD>
<TITLE>EPCMProxy test example</TITLE>
<!--
This is a general proxy to delegate all EPCM method calls to the upper frame -->
```
<SCRIPT src="epcfproxy.js"></SCRIPT>
<SCRIPT>
  var lnDotPos = document.domain.indexOf( "." );
  if (lnDotPos == 0) document.domain = document.domain.substr( lnDotPos + 1 );

  function run() {
    // call EPCF method via proxy (transparent for End-User
    EPCMPROXY.subscribeEvent( "urn:com.sapportals.portal.epcmdemo.animals",
                          "onAnimalSelect", window, "handleEvent");
    showCurrentAnimal();
  }

  function showCurrentAnimal() {
    var lsAnimal =
      EPCMPROXY.loadClientData("urn:com.sapportals.portal.epcmdemo.animals",
                              "animalstored" );
    if (lsAnimal == null){ lsAnimal = "unknown"; }
    document.getElementById( "infoBox" ).innerHTML = lsAnimal;
  }

  function handleEvent( evt ) {
    showCurrentAnimal();
  }

  function showEPCMdata(){
    var data = ""
    data += "\n EPCMPROXY.getUAType = " + EPCMPROXY.getUAType()+"\n EPCMPROXY.getUAVersion = " + EPCMPROXY.getUAVersion()+"\n EPCMPROXY.getUAPlatform = " + EPCMPROXY.getUAPlatform()+"\n EPCMPROXY.getUAVersion = " + EPCMPROXY.getVersion()+"\n EPCMPROXY.getInstanceId = " + EPCMPROXY.getInstanceId()+"\n EPCMPROXY.getUniqueWindowId = " + EPCMPROXY.getUniqueWindowId()+"");
    alert(data);
  }
</SCRIPT>
</HEAD>

<BODY onLoad="run()">
<DIV class="header"> EPCMProxy test component </DIV>
<P>
<BUTTON onClick="location.reload()">reload</BUTTON>
<BUTTON onClick="showEPCMdata()">show EPCM Data</BUTTON>
<DIV id="infoBox"></DIV>
</BODY>
</HTML>
There must be *document - domain* alignment so that the parent EPCM object can be accessed across the *IFrame* border. See chapter *JavaScript Origin Policy* [Page 40] for more details.

Use the *object-call-signature*, with the reference to the window object for the EPCMPROXY.subscribeEvent() method.

**Restrictions**

Restrictions for the *EPCFPROXY* object.

- Following APIs are available for the *EPCMPROXY* object:
  - System API [Page 7]
  - Event API [Page 9]
  - Client data Bag API [Page 11]
  - WorkProtect and Cross Navigation API [Page 13]

- The EPCMPROXY object delegates the calls up one level. If your Portal application uses additional *Framesets or IFrames*, the calls inside the *subframes* are not processed.

- To avoid the JavaScript errors, encapsulate the EPCFPROXY calls with JavaScript try/catch statements.

- The JavaScript file *epcfproxy.js* is not a part of the Portal core. The file must be stored in the application code repository and delivered with the Portal application.

**EPCF Configuration**

To configure the EPCF service have to be logged in as administrator.

Configuration steps:
1. Choose the command *System Administration* in the top level navigation.
2. Choose the commands *System Configuration → Service Configuration*
3. Choose the *Browse* tag and open the node *Applications*.
4. Open the node *com.sap.portal.epcf.loader*.
5. Open the subnode *Services* and you will see the entry *epcfloader*.
6. Click the *epcfloader* entry with the right mouse key and select *Edit*.

The property page is displayed and you can modify the values. To save the changes, choose the *Save* button.

If you are working in a cluster environment, you have to restart the EPCF service so that the changes take immediate effect on all cluster nodes.

If the EPCF property values are not set correctly, the EPCF service uses the default settings at runtime.

**EPCF Properties**
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applet.archive = &lt; on</td>
<td>off &gt;</td>
</tr>
<tr>
<td></td>
<td>on: All classes are loaded from the server to the client in one JAR file.</td>
</tr>
<tr>
<td></td>
<td>off: Every class is loaded from the server to the client individually.</td>
</tr>
<tr>
<td></td>
<td>For a productive system we recommend the value on.</td>
</tr>
<tr>
<td>applet.trace.level = &lt; 0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>The applet.trace.level controls the level of error messages displayed. A higher applet.trace.level reduces the Portal performance.</td>
</tr>
<tr>
<td></td>
<td>0: Display errors only.</td>
</tr>
<tr>
<td></td>
<td>1: Display errors and warnings.</td>
</tr>
<tr>
<td></td>
<td>2: Display errors, warnings and information.</td>
</tr>
<tr>
<td></td>
<td>For a productive system we recommend the value 0.</td>
</tr>
<tr>
<td>framework.level = &lt; 0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>The default setting is 2.</td>
</tr>
</tbody>
</table>

**Properties for the Session Release Agent**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsm.term.custcomp</td>
<td>Defines the name of a user defined Terminator component [Page 35].</td>
</tr>
<tr>
<td>dsm.term.custurl</td>
<td>Defines the URL of a user defined Terminator component [Page 35]. The URL can point to any server based component (for example, servlet, JSP, Perl, ASP and so on) that can process the string parameter TermString. Example: <a href="http://myserver/MyTerminator.asp">http://myserver/MyTerminator.asp</a></td>
</tr>
</tbody>
</table>
dsm.term.mode

Defines which Terminator component [Page 35] is used:

**internal**: The default **Terminator** component **DSMTerminator** is used

**custcomp**: The component specified in the property **dsm.term.custcomp** is used as **Terminator** component.

**custurl**: The component specified in property **dsm.term.custurl** is used as **Terminator** component.

---

**Related Chapters**

Work Protect Mode for EP 6.0 [Page 37]

Properties for EP 5.0 [Page 37]

Terminator Component [Page 35]

---

**Properties for EP 5.0**

The EPCF service for EP 5.0 is configured with the property file **config.properties**. You can find the file in the following directory:

```
<TOMCAT_HOME>/cluster/server/services/servlet_jsp/work/jspTemp/irj/root/WEB-INF/portal/portalapps/com.sap.portal.epcf.loader/config/config.properties
```

The EPCF service reads the property file at the Portal start up when the services are initialized. Therefore you have to restart the Portal so that the changes take affect, after you have changed the properties. If a property has an invalid value, the EPCF service uses the default value.

**General Settings for EP 5.0**

**framework.level**

Sets the EPCF level for the Portal. See chapter EPCF levels [Page 5] for more details.

```
# ---------------------------------------------------------------------
# Framework level
# framework.level=0 -> disabled EPCF
# framework.level=1 -> enable JavaScript
# framework.level=2 -> enable JavaScript & Java
# ---------------------------------------------------------------------

framework.level=2
```

Beginning with EP version 5.0.1.3 the **client data bag** [Page 40] and the **session management** [Page 27] has been implemented in JavaScript. If the Web client has no Java support, the EP switches automatically to this implementation.

---

# Beginning with EP version 5.0.1.3 the **client data bag** [Page 40] and the **session management** [Page 27] has been implemented in JavaScript. If the Web client has no Java support, the EP switches automatically to this implementation.

```bash
# framework.level=2
```
script.set

If the EPCF service is running on the level 1 or 2, the appropriate JavaScript will be included automatically in every Portal page that is generated by the Portal page builder. The JavaScript code can be generated in two different ways:

1. Standard
   
The default value. The JavaScript code is generated in a user friendly form. The code contains comments and is formatted in a way, that it can easily be read, when you select the command View source in your browser.

2. Optimize
   
   With this setting, the JavaScript code is generated without which discards all the comments and unnecessary white-space characters. The HTML code generated with the optimize option is about 30% smaller then generated with the standard version. The optimize option is recommended for a productive system.

```bash
#----------------------------------------------------------------------
# JavaScript settings. It has only effect with framework.level=1,2
#----------------------------------------------------------------------
# script.set=standard -> use standard JavaScript file set
# script.set=optimize -> use size-optimized JavaScript file set
#----------------------------------------------------------------------
script.set=optimize

applet.trace.level

The ECPF service level 2 uses a Java applet that provides the client data bag [Page 40], the data channel [Page 40] and the session management [Page 27] functionality. Log entries are written on the Java console of the Web client Java console to inform about errors. A high trace level generates many messages on the Java console which will slow down the Portal. The Portal administrator can disable messages by lowering the trace level. For a productive system we recommend the value 0.

```bash
#----------------------------------------------------------------------
# Applet specific settings. It has affect with framework.level=2 only
#----------------------------------------------------------------------
# applet.trace.level=0 -> display error messages
# applet.trace.level=1 -> display error and warning messages
# applet.trace.level=2 -> display error, warning and information messages
#----------------------------------------------------------------------
applet.trace.level=0

applet.archive

The Java applet for the EPCF service uses several Java classes. The classes can be loaded from the server in as one file, packed in a Java archive (JAR) file or as single class files. For a productive system, we recommend to set the applet.archive option to on.

```bash
#----------------------------------------------------------------------
# Applet specific settings. It has affect with framework.level=2 only
#----------------------------------------------------------------------
# applet.archive=on -> applet classes accessed via jar-file
```
applet.archive=off -> applet classes accessed as singlefiles
applet.archive=on

DSM Settings

Distributed session management [Page 27] requires the Terminator component [Page 35] to terminate the server sessions. The Terminator component processes the EPCF service request and sends the termination commands to the servers. To customize the Terminator component you have following properties:

**dsm.term.mode**

This property defines which Terminator component will be used. Following values are possible:

- **internal**
  
  Uses the default Portal component DSMTerminator.default, supplied by the Portal.

- **custcomp**
  
  Uses a Portal component specified with the property dsm.term.custcomp.

- **custurl**
  
  Uses the component specified with the property dsm.term.custurl.

**dsm.term.custcomp**

The specified Portal component, for example, myTerminator.default, will be used as Terminator component. The parameter TermString is passed on to that component.

**dsm.term.custurl**

With this property you can specify any server side component (for example, servlet, JSP, Perl, ASP and so on), as Terminator component.

Example:

http://myserver/MyTerminator.asp

The component is started with the parameter TermString.
Session Management

Business applications use a stateful protocol with a dedicated client (GUI). This allows server sessions to be terminated correctly and all server resources (for example, memory) are released when the client closes the session, for example, by logging off from the Portal.

If an application runs on the Web, the connectionless HTTP protocol works in request/response cycles and does not check if the client has already terminated the session. In this scenario the server sessions and resources of the business application are usually released after a predefined timeout (about 5-10 minutes). This delay can cause the following situations:

- Servers can become overloaded and run out of resources by sessions that have already been terminated.
- Locks for the application are held until timeout. In some cases an application can be deadlocked even if there are many servers available.

The SAP Internet Session Protocol (SISP) included in the SAP Workplace 2.10 overcomes this problem.

How does Session Management Work?

For the explanation we consider a service based on the Internet Transaction Server (ITS). The service is started with a LaunchURL containing the ITS location, name of the service and a set of parameters (user, password, language and so on).

Example:

~transaction=sm04&sapwp_active=1&~client=050&~login=wpdev&~language=de&~passwd=blue

When we enter the LaunchURL directly on the client, the content page, that is generated by the ITS, is displayed. Because of the stateless connection, the responsible ITS is not informed whether the running ITS session is still valid or should be terminated. That is not a reliable session management. If the user closes the browser, navigates to another location or chooses the Back button in the browser, the ITS session is kept alive on the server until the predefined timeout occurs.

To overcome this problem, a main page is created that consists of an IFrame (displaying the content from the ITS) and a special JavaScript object called Distributed Session Manager (DSM). The DSM is responsible for session management handling in the page. Every content page includes a JavaScript code that is processed on the client after the page has been loaded. This JavaScript code creates a new JavaScript object called SessInfo with the unique identification of the ITS session SessInfo.GUSID and the callback URL SessInfo.SessURL. The callback URL is the address where the SISP commands will be sent to. See chapter SessInfo Object Properties [Page 35] for more details.

After the content page in the IFrame has been loaded, the SessInfo object is transferred from the content page to the main page and is saved by the DSM. If the user exits the browser, navigates to another location or chooses the Back button, the DSM is activated by receiving the browser event onunload. DSM sends the termination commands to all registered callback URLs and terminates the ITS sessions on the server.

Following restrictions apply:

- JavaScript origin policy

  The SessInfo object and script can only be transfer over IFrame borders, if both pages (main page and content page) use the same document domain. The main page comes from the Enterprise Portal server, for example, http://epserver.mycomp.com, and the
content page comes from the ITS server, for example, http://itsserver.mycomp.com. Both domains must be aligned to the same denominator, such as mycomp.com, to allow scripting. See JavaScript Origin Policy [Page 40] for more details.

- **Browser event onunload**

  The transmission of a termination command from the DSM back to the ITS server must be triggered by a client event. If the DSM simply registers the onunload event and tries to send one or more HTTP requests from the JavaScript, there is no guarantee that all requests are transmitted. If the connection is slow, the browser can be terminated before all requests have been sent.

  The browser event onbeforeunload solves this problem, but only the Microsoft Internet Explorer supports this event.

  A browser independent solution is, to send the termination commands from a Java applet. Received SessInfo objects are collected directly by the applet and the termination commands are sent to the servers in the destroy() method of the applet. The applet runs in its own JVM and the destroy() method of the applet is executed independently of the browser JVM, so the applet is still executed, even if the browser is already terminated.

  When the browser has no Java support, the commands can be sent from the External Window.

- **Applet origin policy**

  A termination command is HTTP request from the client to the ITS. Since the applet can only connect to the server from which it is loaded, you must have the same applet on every ITS. To overcome this problem the applet code should be loaded from the Workplace Middleware Server. Instead of sending the termination commands from the DSM directly to the ITS, the commands are sent to the dedicated Portal component Terminator. The Terminator component finally distributes the commands to the ITS.

### Components

#### Server: ITS server/ ITS services

No modifications are necessary for the ITS; the customer can use the existing ITS transactions in Workplace 2.10/Workplace 2.11 and integrate them into the new Portal by customizing the Launcher component.

#### Server: Workplace server components/services

**EPCM Object** (Portal service)

The EPCM object is involved in the page assembly process. It generates page stubs into predefined page locations. It is responsible to place DSM interfaces, DSM applets and the EPCF infrastructure into the header and body of the page.

**Launcher** (Portal component)

The Launcher includes an IFrame, with the size of 100% by 100%, on the Portal page with the LaunchURL.

**Terminator** (Portal component)

The Terminator is invisible. It receives a list of termination commands from the client and sends them back to the ITS. Every page assembled by the EPCM Service includes one URL of the Terminator component as the parameter of the applet. See also chapter Terminator Component [Page 35] for more details.
**DSM** (Portal component)

The DSM is used for testing. If this component is included in the same page as the **Launcher**, the processed **SessInfo** object can be displayed easily and terminated manually.

**Client: Scripts & Applet**

**DSM Interface**

The DSM interface is JavaScript code that is fully integrated into the Workplace Client Manager EPCM and therefore automatically included in every Portal page generated by the **EPCM Object**. It provides methods to communicate with the **DSM Applet** and a definition of the function `SAPWP.receiveSessInfo()`, which collects the **SessInfo** objects coming from the content *IFrame*. See chapter [DSM Interface](#) [Page 19] for more details.

**DSM Applet**

The DSM applet is integrated with the applet `com.sap.portal.epcf.EPCMfactory`. It implements the saving of the **SessInfo** object in a hash table and a communication channel to the server using **Client Data Channel**. The methods can be accessed using the **DSM Interface**.

**SISP Code** generated by the ITS in the content page

The ITS server generates JavaScript code to create the **SessInfo** object and transfers it to the main page.

**External Window**

The **External Window** is an additional window to the regular Portal page that is used for browsers with no Java support.

**Session Management Step by Step**

**Clients with Java Support**

1. The **EPCM Object** includes the **DSM Interface** and **DSM Applet** in the Portal page.
2. **The Launcher** creates the *LaunchURL* for the ITS service and includes the *IFrame* in the Portal page. There is an instance of the **Launcher** in the Portal page for every *IFrame* that points to a different ITS service.

3. The page is processed in the browser. Every *IFrame* generated loads its contents from the ITS. The ITS returns the content page with the generated **SISP Code** as the response to *LaunchURL*. The **SISP Code** creates the **SessInfo** object and transfers it to the main page.

4. **SessInfo** objects are collected by the **DSM Interface**, transferred to the **DSM Applet** and stored in the applet.

5. If the user navigates to another location, closes the browser or chooses the *Back* button in the browser, the registered ITS sessions become invalid and must be terminated. The browser triggers the *unload* event and the **DSM Applet** starts to process its own **destroy()** method.

6. The **destroy()** method of the **DSM Applet** computes the URL to terminate the sessions on the ITS for every **SessInfo** object stored. All URLs are collected and placed into one single HTTP *post* request (parameter **TermString**) and sent to the **Terminator** component.

7. The **Terminator** component splits the **TermString** parameter from the HTTP request into single URLs and sends the URLs to the different ITS servers with a new URLConnection. The server sessions on the ITS are then terminated.

8. Every **SessInfo** object is reported by the **EPCF** to the **DSM**.

**Clients without Java Support**

1. **The EPCM Object** includes the **DSM Interface** on the Portal page. The event handler in the **DSM Interface** is attached to the *onunload* event.

2. **The Launcher** creates the *LaunchURL* for the ITS service and includes the *IFrame* into the Portal page. There is an instance of the **Launcher** in the Portal page for every *IFrame* that points to a different ITS service.
3. The page is processed in the browser. Every IFrame generated loads its contents from the ITS. The ITS returns the content page with the generated SISP Code as the response to the LaunchURL. The SISP Code creates the SessInfo object and transfers it to the main page.

4. SessInfo objects are collected by the DSM Interface and stored in a JavaScript array.

5. If the user navigates to another location, closes the browser or chooses the Back button in browser, the registered ITS sessions become invalid and must be terminated. The browser fires the unload event.

6. The onunload event handler creates an External Window, in addition to the existing Portal page, with the URL of the Terminator component. The browser sends a HTTP get request (parameter TermString) to the Terminator. After the request has been sent the External Window can be closed automatically using the JavaScript timeout function.

7. The Terminator component splits the TermString parameter from the HTTP request into single URLs and sends the URLs to the different ITS servers with a new URLConnection. This terminates the server sessions on the ITS.

8. Every SessInfo object is reported by the EPCF to DSM.

Scripts

Here are some scripts that are used for the session management. The different examples refer to the steps, for example Step 1, shown in the previous chapter Session Management Step by Step.

Client scripts

DSM Interface (Step 1)

```
<SCRIPT src="/irj/portalapps/com.sap.portal.epcf.loader/script/standard/js13_epcf.js"> </SCRIPT>

<SCRIPT>

<!--
EPCM.init( 2, 2, '8a50298ad8156b33d0445ae7e4f9062e', 1, 5.5, 1 );
EPCM.DSM.init("http://p45462.wdf.sap-ag.de:8080/irj/servlet/prt/portal/prtroot/DSMTerminator.default")
; function SAPWP_receiveSessInfo( sessInfo, frameRef ){
EPCM.DSM.processSession( sessInfo );
EPCM.DSM.notifyMonitor( sessInfo );
}
//-->

</SCRIPT>
```

DSM Applet as the Part of EPCM Object (Step 1)

```
<APPLET CODEBASE="/irj/services/epcfloader/applet"
CODE="com.sap.portal.epcf.EPCMfactory"
ID="_EPCMfactory_
NAME="_EPCMfactory_"
STYLE="POSITION: absolute;"
```
Function **SAPWP_receiveSessInfo** is the entry point for every **SessInfo** Object. The **SessInfo** object is transferred to the **DSM Applet** using the method **EPCM.DSM.registerSession()** from the **DSM Interface**. The applet parameter **DSM.serverUrl** specifies the URL of the **Terminator** component. The collected termination commands will be sent to this URL (Step 6).

**SISP Code Example Generated by the ITS (Step 3)**

```javascript
// ----------------------------------------------------------------
// -----------------------------
function SAPWF_sendSessInfo( cltSessInfo ) {
    var lsDomain = "";
    var liBehindFirstDot = location.hostname.indexOf( "." ) + 1;
    if (liBehindFirstDot > 0) {
        document.domain = location.hostname.substr( liBehindFirstDot );
    }
    var loCF = window; // current frame
    var loPF = (loCF.opener != null)? loCF.opener: loCF.parent; // parent frame
    while (loCF != loPF) { // while top frame not reached
        if ("object" == typeof( loPF.document ) ) { // is parent frame scriptable?
            if (loPF.SAPWP_receiveSessInfo != null) {
                // workplace frame found, deliver sessinfo
                loPF.SAPWP_receiveSessInfo( cltSessInfo, loCF_ );
                return true;
            } else {
                // try upper frame
                loCF = loPF;
            }
        } else {

```
Testing the Session Management

The Portal has two components to test the session management:

- DSM TestLaunch.
  Creates the docking IFrame for the ITS response.
- DSM Monitor
  Allows you to check the Public API functionality and displays the SessInfo object list and other information.

Both components must be included on the same Portal page for the test. The DSMTestLaunch component properties must be configured to start an ITS service. As already shown in the chapter Session Management Step by Step the transfer of the SessInfo objects is processed in several steps. The DSM component can test Step 4, Step 5 and Step 6 even if the ITS is not available. The DSM component can be started in single component mode with the following URL:

```
```

The test results of the session termination are displayed on the Java console of the browser (browsers with Java support) or on the External Window after the resize of the window is done (browsers with no Java support).

**Test 1: DSM Displays SessInfo Objects**

If the session management works correctly, all received SessInfo objects are displayed in the DSM and you can display all properties of the SessInfo objects. If there are no SessInfo objects displayed in the DSM following problems are possible:

1. ITS did not include the SISP coding stub into the response

```javascript
loPF = (loPF.opener != null) ? loPF.opener : loPF.parent;
}
} else {
  // access to foreign frame denied, stop here
  return false;
}
} // top frame reached, but no workplace frame found
return false;
}

function SAPCLT_SessInfo() {
  var prot = ("off" == "on" ? "https://" : "http://" );
  this.protocolVersion = "1.0";
  this.sessUrl = prot + "pgtlsp4c.wdf.sap-ag.de:1080" + "/scripts/wgate" + "/" + "fielddump" + "/~session=+" + "QW2-A:pgwdf062:0000.0040.afa108fd";
  this.GUSID = "QW2-A:pgwdf062:0000.0040.afa108fd";
  this.lastSessCmd = "USR_OPEN";
  this.redirectURL = "";
  this.dTimeout = "120";
}
SAPWF_sendSessInfo( new SAPCLT_SessInfo());

</SCRIPT>
```
Check if the service parameter `SAPWP_ACTIVE` is set in ITS. You can add to the LaunchURL following string: `SAPWP_ACTIVE=1`

2. The ITS server and Workplace server use different domains

The typical local test installations at SAP do not have a specific IP address. The Workplace Server can be accessed using `localhost` instead of a specific server name or IP address (for example, http://localhost:8080/irj/servlet/prt/portal) and the ITS server uses a server name like `itsserver.wdf.sap-ag.de`. This does not comply with the JavaScript origin policy [Page 40] and therefore the DSM can not display a SessInfo object.

To solve this problem, use a specific IP-address for the computer or define a synonym in the hosts file of the computer (for example, Windows NT keeps that file at location C:\WINNT\system32\drivers\etc\hosts). The entries for the synonym look like that:

```
127.0.0.1 localhost
127.0.0.1 myhost.wdf.sap-ag.de
```

Now you can access your Portal runtime from your computer locally using the address:

```
http://myhost.mycomp.com/irj/servlet/prt/portal
```

The name `myhost.wdf.sap-ag.de` is only known on your local computer and can not be accessed from another computer.

**Test 2: DSM Interface to DSMTerminator Component Connection**

SessInfo objects will be processed during the page `unload` event. The termination command is sent by the Java applet to DSMTerminator component. The transfer protocol can be displayed on the Java console of your browser. The Java console of the MS Internet Explorer can be enabled as follows:

```
Tools → Internet Options → Advanced → Microsoft VM
```

If there is an error message in the Java console, you should first check if the proxy settings `http.proxyHost`, `http.proxyPort` and `http.nonProxyHosts` in your servlet environment are correct.

**Test 3: Destroying of ITS Sessions**

Every SessInfo object represents a session on the ITS. You can directly start the R/3 system to which the ITS server is connected and check with the transaction SU04 which sessions have been destroyed during the page `unload` event.

**Related Chapters**

[SessInfo Object Properties][Page 35]
[DSM API][Page 19]
[Terminator Component][Page 35]
SessInfo Object Properties

SessInfo properties defined by the SISP protocol version 1.0.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUSID</td>
<td>Global Unique Session ID.</td>
</tr>
<tr>
<td>sessUrl</td>
<td>URL where the session commands and termination command is sent to.</td>
</tr>
<tr>
<td>lastSessCmd</td>
<td>Last session command.</td>
</tr>
<tr>
<td>redirectURL</td>
<td>URL which can be used for chaining the application (close the current one and start new one with given redirectURL).</td>
</tr>
<tr>
<td>dTimeout</td>
<td>Timeout value for session response.</td>
</tr>
<tr>
<td>Label</td>
<td>Description of session.</td>
</tr>
<tr>
<td>protocolVersion</td>
<td>Provides information of the version of the protocol.</td>
</tr>
<tr>
<td></td>
<td>The current version released is 1.0.</td>
</tr>
</tbody>
</table>

Terminator Component

The Terminator component sends HTTP request to servers, to terminate the registered sessions.

Basic functionality

The Terminator component receives a HTTP request (either get or post), with the parameter TermString, TermString contains escaped URLs concatenated with an ampersand (&). The special characters must be escaped (see JavaScript functions escape/unescape) so that the URLs are transmitted correctly.

Format of the parameter TermString:

escaped(escaped(URL1)+"&"+escaped(URL2)+"&"+...escaped(URLn))

The Terminator component splits und unescapes the TermString parameter into single URLs and sends for every of them one single HTTP post request to the target system (usually ITS).

The Portal uses by default the Portal component DSMTerminator as Terminator component. You can customize this component or define another component or Portal component as Terminator component.

Configuring the DSMTerminator component

The default Terminator Portal component DSMTerminator can be customized by changing the properties of the iView.

DSMTerminator Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
validationEnabled=<true|false>  The **DSM Terminator** component sends requests to the servers it is told to. With this property you can define, if the URL is validated before it is sent. If the property is set to true, every URL will be validated against the `trustedHost` and `restrictedHost` list.

trustedHosts=<host_name_list>  This semicolon separated list specifies the hosts to which the **DSM Terminator** component is allowed to connect. A host entry can also be specified as pattern with one wildcard (*).
Example: *.myfirm.com;*yourfirm.com

restrictedHosts=<host_name_list>  This semicolon separated list specifies the hosts to which the **DSM Terminator** component is not allowed to connect. A host entry can also be specified as pattern with one wildcard (*).
Example: *.badfirm.com;*anotherbadfirm.com

cookieThroughEnabled=<true|false>  Allows that cookies from the incoming requests are passed on to the all requested destinations. This is required when you integrate components based on the BSP (Business Server Pages).

The **DSM Terminator** component can also be used *standalone* by specifying the URL of the **DSM Terminator** component (<Portal_URL>/irj/servlet /prt/portal/prtroot/DSMterminator.default) with following parameters:

### DSM Terminator Parameter

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TermUrl=&lt;String&gt;</td>
<td>Optional parameter specifies all URLs to which the requests should be distributed. If the parameter not specified or empty, no request is distributed.</td>
</tr>
<tr>
<td>Autoclose=&lt;positive_Number&gt;</td>
<td>Optional parameter that specifies the delay before the External Window (used for browsers with no Java support) is closed. If not specified, the External Window will not be closed.</td>
</tr>
</tbody>
</table>
### Filter=<NOCOPY | ESCAPE | COPY>

Optional parameter that specifies how the single responses returned after the request distribution should be handled by the DSMTerminator component, when the final (collective) response is assembled.

- **COPY**
  The single responses are copied into the collective response.

- **ESCAPE**
  Is similar to COPY but all special characters will be converted (escaped) so that the string is properly displayed on the HTML page.

- **NOCOPY**
  The single responses are not copied into the collective response.

If the parameter is not specified, the NOCOPY option is selected.

---

### Using Another Terminator Component

You can define another Portal component or an external script or process specified by full qualified URL (for example, servlet or ASP) as Terminator component. The specified component has to implement the basic functions of a Terminator component. See chapter [EPCF Configuration](Page 22) for more details.

---

### WorkProtect Feature for EP 6.0

To match the concept of the WorkProtect feature, a Portal application must meet the following requirements:

- Maintain the dirty indicator
- Adjust Portal links (This function is currently only supported by CRM).

#### Maintaining the Dirty Indicator

The dirty indicator status of a Portal application informs the Portal that there is unsaved data.

The Portal application sets the dirty indicator when the user enters a new value into an input field. The Portal application resets the dirty indicator when the user saves the value (for example when the user chooses the Save button).

See chapter [WorkProtect and Cross Navigation API](Page 13) for more details.

#### Adjusting Portal Links

The Portal can only check the current dirty indicator and perform the navigation without losing data, if the Portal application replaces all the links that can destroy the contents of the content area with links having the following syntax (analogous to New Navigation Model / WorkProtect Mode, section Cross Navigation):

```html
<A HREF=myLink onclick=return EPCM.doNavigate('any_PCD_URL')>
```

---

Enterprise Portal Client Framework (EPCF) 646
The parameter `<any_PCD_URL>` specifies the location of a page or an external service in the user role. Constants must be enclosed in quotation marks.

You can find the correct value for the page in the Role Editor.

Make sure that you update the corresponding parameter values for the `<PCD_URL>` in the secondary links and navigation targets when you change the role structure.

**Configuration Test**

Test tool: `com.sap.portal.epcf.loader.Dirty`

The test tool supports the tracing and solving of problems related to the _dirty indicator_. You can find the tool under

System Administration → Support → Support Desk → Client Framework

**Navigation**

The Portal navigation model supports the _navigation_ and _WorkProtect_ feature. These features allow tight integration of _stateful_ applications in the Portal environment and improve the usability of the applications running in the Portal.

**Related Topics:**

- Navigation Service [External]
- Navigation API [Page 14]
- WorkProtect API [Page 13]

**Availability**

The JavaScript based API has been introduced in EP 5.0.4.1 (see SAP Note 543274) and is compatible to the API in EP 6.0.

**Business Case for Navigation**

Portal implementations usually have separate areas for handling navigation and displaying content. The navigation area typically visualizes a navigation tree and highlights the selected node. The content area visualizes this selected node, for example, a Portal page, document or Portal application). This model resembles a simple file system browser and works well with _stateless_ Portal applications.

However, this approach does not meet all requirements for an enterprise solution, which should be able to handle complex business processes in parallel and switch from one context to another.

**Navigation Target**

The navigation target specifies the location of an iView or a page in the current user role. The target can be obtained from the Portal catalog as a value that is concatenated by folder ids, roles or other objects.

The navigation target has to be prefixed with the corresponding navigation connector name that is used for retrieving the navigation structure. When accessing iViews and pages in the role from the Portal Content Directory (PCD), you have to add the prefix `ROLES://` to the URL.

**Example:**

We have created a custom role (`MyRole`) and assigned an iView (`MyIView`) to it:
The corresponding navigation target is:

ROLES://portal_content/MyRole/MyRole/MyTest/MyIView

Compatibility to EP 5.0

In the Enterprise Portal 5.0 the navigation target to the Portal pages or external services are specified without the prefix for example, /roles/MyEP50Role/MyEP50Folder/MyEPApp. These navigation targets are also valid in EP 6.0.

When you have migrated the content from EP5.0 to EP6.0, the migrated content is in folder portal_content/com.sap.portal.migrated/ep_5.0. The Portal navigation will check for incomplete navigation targets and it to the new schema automatically.

ROLES://portal_content/com.sap.portal.migrated/ep_5.0/roles/MyEP50Role/MyEP50Folder/MyEPApp

Navigation Features for Navigation Targets

The Enterprise Portal offers the following features for navigation targets:

- Start with Navigation Target
  This feature lets you start the Portal and automatically navigate to any Portal page or iView inside the role.

- Navigation
  This feature allows seamless navigation from Portal applications. The primary links in the navigation as well as the secondary links in the content area can be used in a Portal application. The navigation updates the content area correctly and highlights the corresponding node in the navigation tree - for a primary link or another instance.

Start with Navigation Target

The Portal can be started in the browser with an URL that starts the first page assigned in the user role. The URL has the following structure:

<your_portal_server>/<portal_alias>

This URL can be extended by the navigation target that will be called automatically after the start. You have to specify a valid page (by default, index.htm). The extended URL has the following structure:

<your_portal_server>/<portal_alias>/<initial_page>?
NavigationTarget=<escaped_NavigationTarget>

The <escaped_NavigationTarget> parameter represents the escaped location of the page or iView in the role. Escape is necessary to avoid conflicts when using special characters. See the JavaScript function escape for more details.

Example:

NavigationTarget=ROLES%3A//portal_content/MyRole/MyRole/MyTest/MyIView
Navigation

The solution implemented for secondary links in the Portal component uses a combination of a HTML hyperlink and a call of an EPCF service method EPCM.doNavigate() [Page 13].

\[\text{<A HREF="myLink" onclick="return EPCM.doNavigate('target')">This is HTML Link</A>}\]

The String parameter target represents the location of an iView or a page in the role. The value is available from the Portal content catalog.

Constant String values must be enclosed in JavaScript in single quotes.

When the link is activate, the EPCM.doNavigate() method informs the Top Level Navigation (TLN) about the required navigation target. The TLN will handle this request it in the same way as any other navigation using primary links.

Glossary

Terms used in the EPCF service description.

Automatic Server Session Termination (ASST)

The Internet protocol HTTP is connectionless. However typical business applications provide information about the user session.

The Enterprise Portal session management controls the session state in an application with the SAP Internet Session Protocol (SISP). It avoids locks and also releases objects when the user leaves an application, closes the browser or logs off from the computer. This functionality uses the SISP that has been introduced with Workplace 2.10 and 2.11.

Client Data Bag

The client data bag is a transient data buffer for the Web client (browser) which is active as long as the session of the browser. For the Portal that is as long as the user is logged on to the Portal. An iView can access the client data bag with the Enterprise Portal Client Manager (EPCM) object.

Client Data Channel

An additional channel to the main communication channel (the browser HTTP connections), that allows independent communication.

JavaScript Origin Policy

The JavaScript Origin Policy controls the access to the Document Object Model (DOM) from different frames. Scripting between two frames is permitted only if both frame sources come from the same top level domain.
Example

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Scripting permitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>site1.page2.mydomain.com</td>
<td>site2.page3.mydomain.com</td>
<td>Yes</td>
</tr>
<tr>
<td>site1.mydomain.com</td>
<td>site2.yourdomain.com</td>
<td>No</td>
</tr>
</tbody>
</table>

The EPCF service automatically sets the document domains to top level.

All browsers support the JavaScript Origin Policy, so foreign web sites are unable to retrieve data from the Portal page or the iViews.

An similar origin policy also applies for the Java Virtual Machine (JVM). Classes/objects can only interact with classes/objects which are loaded from the same location. Therefore it is impossible for a foreign applet to access the data inside the Client Data Bag or use the Client Data Channel.

**Namespaces**

The World Wide Web Consortium (W3C) ([http://www.w3c.org](http://www.w3c.org)) defined the naming and addressing standards for Web development. The Enterprise Portal uses these standards in the EPCF service for the events and [Client Data Bag](Page 40).

This chapter refers to Request for Comments (RFC). The comment for the specified RFC number can be found at [http://www.ietf.org/rfc/](http://www.ietf.org/rfc/).

**Name Syntax**

Some methods, for example, `EPCM.raiseEvent(...)` expect a `Name` as argument. `Name` is a String variable with restricted characters.

**Valid characters for Name are:**

<table>
<thead>
<tr>
<th>Range</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase characters</td>
<td>a to z</td>
</tr>
<tr>
<td>Uppercase characters</td>
<td>A to Z</td>
</tr>
<tr>
<td>Numerical characters</td>
<td>0 to 9</td>
</tr>
<tr>
<td>Additional characters</td>
<td>Underscore(_), Dash (-)</td>
</tr>
</tbody>
</table>

**Namespace Syntax**

The namespace definition is compliant with the Unified Resource Name (URN) specification, which is available from the World Wide Web Consortium. Namespaces used in JavaScript functions calls must be compliant to this specification.

**Namespaces reserved by the Portal**

<table>
<thead>
<tr>
<th>Reserved name-space</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.sapportals.portal.*</td>
<td>Portal core development</td>
</tr>
<tr>
<td>com.sapportals.*</td>
<td>Portal core development</td>
</tr>
</tbody>
</table>

The namespace must start with the string "urn:" followed by the structure (in Backus-Naur form)

```
<URN>::="urn:" <Namespace_identifier> ":" <Namespace_Specific_String>
```
The tokens <namespace_identifier> and <Namespace_Specific_String> must be compliant with the recommendation RFC 2141 and RFC 1630. We recommend that you use only lowercase, uppercase and numerical characters.

**Uniform Resource Identifier (URI)**

It addresses a resource in the Internet in the following way:

- By name
  - This is called *Uniform Resource Name* (URN).
- By location,
  - This is called *Uniform Resource Locator* (URL).

**Uniform Resource Locator (URL)**

It addresses a resource in the Internet. The URL is the address you enter into the address field of your browser. The URL syntax describes a subset of the *Uniform Resource Identifier* syntax.

**Uniform Resource Name (URN)**

It addresses a resource in the Internet, regardless of its location. The URN syntax follows the rules of the URI. A URN can also be used to define distinct entities without being associated to an existing resource. The name spaces in the Portal make use of this feature. A URN has the prefix: urn:/. For further information about the syntax, see the description for *RFC 2141* at www.ietf.org.