The SAP Web Framework (SWF)

MYSAP.WP_SAPWEBFRAMEWORK

Release 2.11

Copyright

Icons

Icon	Meaning
⚠	Caution
	Example
➡	Note
	Recommendation
	Syntax
💡	Tip

The SAP Web Framework (SWF)

Purpose

The SAP Web Framework (SWF) provides functions that make programming Internet services (MiniApps and Internet Application Components) easier. The SWF is installed automatically when you install the Internet Transaction Server (ITS). However, you can also install the SWF on any other server.

Features

The SWF provides the following functions:

- Origin concept: How can two HTML pages communicate with one another (inter-script)?
- Service concept: How do I talk to elements on different pages of a service?
- Interface concept: How can different services communicate using interfaces?
- Event concept: How can anonymous communication between different services be implemented?

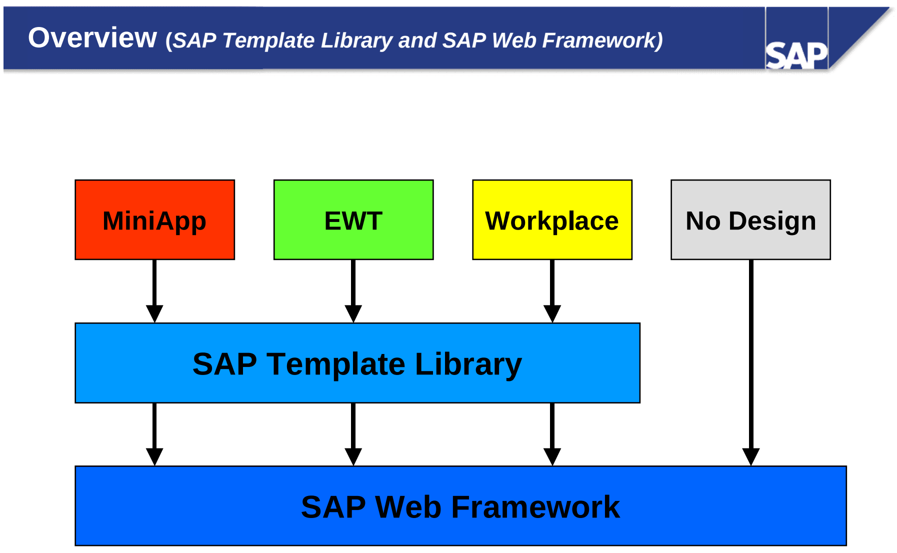
Including the SAP Web Framework (SWF) In Your Service

Purpose

To be able to use the SWF, you must include the SWF definitions in your service. You have two options for doing this:

- Include the SWF together with the SAP Template Library
- Include the SWF without the SAP Template Library

The type that you decide on depends on whether you want to create a service that builds on the SAP design and uses the SAP Business HTML functions or whether you want to implement a service without these design aids.



Process Flow

Including with the SAP Template Library

1. Add the following line at the start of each page of your service:

```
`include( ~service="system", ~theme="dm", ~language="",
                ~name="TemplateLibraryDHTML.html" );`
```

2. Add the following line on each page of your service in the <*head*> HTML tag.

```
`SAP_TemplateJavaScript()`
```

For more information, see also the documentation on the SAP Template Library.

Including Without the SAP Template Library

1. Add the following line on each page of your service:

```
`include( ~service="system", ~theme="dm", ~language="",
                ~name="SAPWebFramework.html" );`
```

2. Add the following line on each page of your service in the <*head*> HTML tag.

```
`SAP_WebFrameworkJavaScript()`
```

Origin Concept in JavaScript

In the JavaScript Origin Concept, only those pages that have the same origin can communicate with one another. In JavaScript, this is controlled using the browser settings. The standard setting in this case is that the origin of a page is the respective Web server on which the page was started. For SAP applications, this would mean that only those services that were created on the same Web server can communicate with one another.

Therefore, this Origin Concept has been made weaker in the SAP Web Framework (SWF). The standard setting of the Web browser is changed so that all services that originate from the same domain can communicate with one another. The prerequisite for this is that the URL that points to a service must always be qualified in full.

You can talk to your service by using *http://webserver.domain:port/path/page.html*, however, you may not use *http://webserver:port/page.html* under any circumstances.

Service Concept

Complex applications (such as the mySAP.com Workplace) containing many services that in turn can contain many HTML frames, are particularly difficult to manage for the following reasons:

- Large hierarchies of framesets are built
- There is no demarcation between services
- Different pages in a service are only loosely connected

The SAP Web Framework connects all pages of an application. Therefore, all pages in an application form a unit. Furthermore, a JavaScript function *SAPWF_getObject()* is offered. You can use this function to simply address objects in a service (see also: the *SAPWF_getObject* unit). The prerequisite for this is that the object ID was uniquely assigned in the service.

If you have created an object with the name *myLabel* on one page of your service, you can, for example, change the *setLabel* attribute in the following way:

```
SAPWF_getObject( "myLabel" ).setLabel( "ready" )
```

The service concept that is currently implemented does not support any recursive use of services. In this case, the recursively created services are interpreted as a service from the SAP Web Framework.

Interface Concept

Services can communicate with one another via interfaces, since these control the flow of data between or within services. In part, the interfaces provide implementations that your application can redefine.

For example, the SAP Web Framework provides the following interfaces:

- Terminate [Seite 13]: The application can use this interface to regain control before the user quits the Workplace.
- Refresh [Seite 12]: You can use this interface to control the refresh rate for the Workplace.
- Save/Cancel [Extern]: You can use this interface to call the standard dialog for Save/Cancel.

- External Window [Seite 9]: This interface helps to manage external browser windows that that application called.

The interfaces provided by the SAP Web Framework are standardized, provided that they are public.

You can get a reference for the interfaces using the SAPWF_getInterface(iName) [Seite 24] function.

When calling an interface, your application can react using the logic implemented in the interface. It can also redefine this logic.

In all cases, you have to create a flow file, even if you do not want to store any flow logic.

You implement the SWF interface to refresh your application and set the interval to 12 seconds.

```
<head>

...

'SAP_WebFrameworkUseInterface( id="SAP_RefreshInterface",
scope="public" )'

...

</head>


<body>

...

<SCRIPT>

   function SAPWF_getInterface ("SAP_RefreshInterface"
).setInterval( 12 );

</SCRIPT>

</body>
```

SAP_ExternalWindowInterface

You can use the *SAP_ExternalWindowInterface* interface to manage external windows.

You should not redefine the functions. The following functions are available:

Function	**Description**
open	Opens a new browser window
isAlive	Checks if the window is still open or if it has already been closed
getWindowRef	Supplies the window reference
close	Closes a window
closeAll	Closes all windows
getLength	Number of windows opened

elements	List of windows opened

Use

1. Include the following line in your service's <*head*> HTML tag.

'SAP_WebFrameworkUseInterface(id=" SAP_ExternalWindowInterface", scope="public")'

2. Get a reference for the interface using the following function:

var IExternalWindow = SAPWF_getInterface("SAP_ExternalWindowInterface");

3. The functions listed above allow you to open and close windows, for example. You can find the descriptions of the transfer parameters in the table below.

open

The *open* function opens a new browser window, provided that there is no window with exactly the same name already open. In this case, the function uses the browser window that is already open.

This function expects the following transfer parameters: The transfer parameters correspond to the parameters of the window open method in JavaScript:

Parameter	Optional	Description
url	no	URL that is to be displayed in the additional window
name	no	Technical name of the window
features	yes	Characteristics of the additional window
bReplace	yes	Replaces the existing information

As a result, the function returns a reference for the browser window that is created.

In this example, the page http://mypage.com is to be started in the window "test". In addition, the program should configure some browser window options.

```
var loNewWnd = IExternalWindow.open( "http://mypage.com",
"test"
,"toolbar=yes,directories=yes,status=yes,menubar=yes,scrollb
ars=no,resizable=yes,copyhistory=yes,width=600,height=500","
" );
```

isAlive

This function checks if a browser window is still open.

Interface

The technical name of the browser window is expected as the value transferred.

The function returns a Boolean type:

- True means that the window still exists and has not been closed.
- False means that the window has been either closed or renamed.

If the browser window has already been closed, a new window is to be opened:

```
if ( IExternalWindow.isAlive( "test" ) )
IExternalWindow.open( "http://mypage.com", "test" ,"", "" );
```

getWindowRef

This function allows you to determine the reference for a browser window.

Interface

The technical name of the browser window is expected as the value transferred.

The function returns a reference for the browser window.

You want to receive a reference for the browser window with the technical name *test*.

```
loNewWnd = IExternalWindow.getWindowRef( "test" )
```

close

This function allows you to close a browser window.

Interface

The technical name of the browser window is expected as the value transferred.

This function does not return a value.

You want to close the browser window with the technical name *test*.

```
IExternalWindow.close( "test" );
```

closeAll

This function allows you to close all registered browser windows.

Interface

This function does not expect a value to be transferred.

This function does not return a value.

```
IExternalWindow.closeAll();
```

getLength

This function allows you to determine the number of open windows that the user has not yet closed.

Interface

This function does not expect a value to be transferred.

This function returns the number of open windows.

You want to determine the number of open windows.

```
Anzahl = IExternalWindow.getLength( );
```

elements

This function provides a list of the open windows.

Interface

This function does not expect a value to be transferred.

This function returns a reference for the windows.

SAP_RefreshInterface

You can use the *SAP_RefreshInterface* interface to define the time interval after which the system should refresh your application. You can also define the *refresh* function yourself, if you want to implement your own refresh routine.

The refresh routine implemented by the SAP Web Framework triggers the *onLoad* event.

In the case of MiniApps, you can also specify in the Web Application Builder whether the MiniApp is to be refreshed automatically. With this method, the MiniApp URL is then called again after this time interval.

Use

1. Include the following line in your service's <head> HTML tag:
   `'SAP_WebFrameworkUseInterface( id="SAP_RefreshInterface", scope="public" )'`
2. Get a reference for the interface using the following function:
   `var IRefresh = SAPWF_getInterface("SAP_RefreshInterface" );`
3. You can use the functions listed above to set and read attributes, for example. You should not redefine these functions in your application.

Method	**Description**	**Parameter**
refresh	Calls the refresh routine	None
setInterval	Sets the refresh interval	Transfer of the interval in seconds
getInterval	Queries the refresh interval	Return of the interval in seconds
getAutomation	Queries whether automatic refresh is active	Return of the following values: true: active false: not active
setAutomation	Sets the attribute to activate automatic refresh	Transfer of the following values: true: active false: not active

You set the refresh interval for your application to 600 seconds and activate the automatic refresh function.

```
var IRefresh = SAPWF_getInterface("SAP_RefreshInterface" );
     IRefresh.setInterval( 600 );
     IRefresh.setAutomation( true );
```

SAP_TerminateInterface

This interface allows you to react in your application to the user closing the application. To do this, you must implement the *SAP_TerminateInterface()* function in your application. This is then called by the SAP Web Framework if the user wants to close the application.

Use

1. Include the following line in your service's <head> HTML tag:
   `'SAP_WebFrameworkUseInterface( id="SAP_TerminateInterface", scope="public" )'`
2. Get a reference for the interface using the following function:
   `var IRefresh = SAPWF_getInterface("SAP_TerminateInterface" );`
3. The *isOK* function is always called if the user wants to end the application using a Workplace function. You can determine whether the application can be ended (return true) or whether it may not be ended (return false). In the second case, the ending of the application is prevented.
   To do this, define the *isOK* function for the interface using:
   ```
   function myA.isOK( ) {
      if <condition>
            myA.isOK = true;
      else
             myA.isOK = false;
      endif }
   ITerminate.isOK = myA_isOK;
   ```
4. Define the *SAP_TerminateInterface()* function: In this function, you can implement which activities the system is to perform before ending the service.
   ```
   function myA.terminate() {
   do something};
   Iterminate.terminate = myA.terminate;
   ```

SAP_WorkplaceInterface

The SAP_WorkplaceInterface interface provides you with functions in the Workplace. At present there are functions for the following topics:

Function	**Description**
Notification to application whether the Onload event was triggered	
applNotifOnload	If you redefine this method, your application will be called when the Onload event is triggered in the Workplace.
Showing and hiding the LaunchPad	
launchpadShow	Show the LaunchPad
launchpadHide	Hide the LaunchPad
Data storage in the Workplace. This allows your applications to store data in the Workplace. This data is then available even when the application starts a new page.	
dataBagExists	Check whether the system already created an array
dataBagSet	Set new values in the data bag
dataBagGet	Read values from the data bag
dataBagValues	Read all key-value pairs from the data bag

dataBagKeys	Read all keys from the data bag
dataBagDelete	Delete a key in the data bag
dataBagDeleteAll	Delete all keys in the data bag

Use

1. Get a reference to the WorkplaceInterface using the SAP_getWorkplaceInterface() function:
   ```
   var IWorkplace = SAP_getWorkplaceInterface();
   ```
2. Use the functions listed above to access the functions of the WorkplaceInterface.
3. If you wish to use the applNotifOnload function, you must implement the appropriate logic in your application:
   ```
   function myA.applNotifOnload( ) {
      do something }
   IWorkplace.applNotifOnload = myA.applNotifOnload;
   ```

Showing and Hiding the LaunchPad

The functions *launchpadHide()* and *launchpadShow()* do not require parameters and do not return a value.

The following program shows how to hide the LaunchPad:

```
var IWorkplace = SAP_getWorkplaceInterface();
IWorkplace.launchpadHide();
```

Data Storage in the Workplace.

If the application starts a new page, its data is no longer available. Therefore the Workplace provides a data bag in which the application can store data that it still requires after starting a new page.

You can store data of types integer, string and Boolean in the data bag. If you store objects or arrays, the system only stores the references to the object or array. These references are no longer valid after the application starts a new page, for example.

dataBagExists(dbId)

This function checks whether the system has already created a data bag called **dbId**. It returns a Boolean type.

The following program shows how to check if the system has already created a data bag called *My_Data_Bag*:

```
var IWorkplace = SAP_getWorkplaceInterface();
IWorkplace.dataBagExists( "My_Data_Bag" );
```

dataBagSet(dbId, key, value, idx)

This function allows you to create a new data bag, provided that you have not already created one called **dbID**. It is recommended that you create a separate data bag for each instance of an application to avoid undesired side effects. As of ITS 4.6D you can use the ITS context variable *~serviceunique* for this purpose.

In this function you also create new entries in the data bag or overwrite existing ones. The two parameters *dbId* and *key* must uniquely identify the entries. You pass the value using the *value* parameter.

The *idx* parameter is optional and is only required for arrays.

If you wish to put an array in the data bag, provide values for the parameters *value* and *idx*, where *value* corresponds to one of the values in the array and *idx* specifies the index of the entry.

The following program shows how to create a value *X* with *My_Value* as its key and how to create one line in an array called *My_Array*:

```
IWorkplace.dataBagSet( "My_Data_Bag", "My_Value", "X" );

IWorkplace.dataBagSet( "My_Data_Bag", "My_Array", "First",
"1" );
```

dataBagGet(dbId, key, idx)

This function allows you to read an entry from the data bag. You must always specify *dbID*, that is the data bag ID, and the key *key* of the entry. In the case of an array, you must also specify which line of the array you want to read.

The following program shows how to read data from a new data bag. It reads the value for the *My_Value* key and the first entry in the *My_Array* array:

```
My_Value = IWorkplace.dataBagGet( "My_Data_Bag", "My_Value"
);

My_Array[1] = IWorkplace.dataBagGet( "My_Data_Bag",
"My_Array", "1" );
```

dataBagValues(dbId)

This function returns an array containing the values in the current data bag.

The following program shows how to read all the values in the data bag.

```
My_Value = IWorkplace.dataBagGet( "My_Data_Bag", "My_Value"
);
```

dataBagKeys(dbId)

This function returns an array containing the keys in the current data bag.

The following program shows how to read all the keys in the data bag.

```
Key_Array = IWorkplace.dataBagKeys( "My_Data_Bag" );
```

dataBagDelete(dbId, key)

This function allows you to delete a certain entry in the data bag. You must specify the key of the entry and the data bag ID.

The following program shows how to delete the *My_Value* entry.

```
IWorkplace.dataBagDelete( "My_Data_Bag", "My_Value" );
```

dataBagDeleteAll(dbId, key)

This function allows you to delete the data bag. You must specify the ID of the data bag.

You wish to delete the data bag called *My_Data_Bag*.

```
IWorkplace.dataBagDeleteAll( "My_Data_Bag" );
```

Implementing Your Own Interfaces

Use

If you are implementing one-way services, it does not make sense to define interfaces. However, if you are building extensive services that consist of many pages, you can use interfaces to define functions once and call them easily from all pages via the interface.

Prerequisites

If you want to create your own interfaces, you must declare them in a JavaScript files (*.js). You must store the public interfaces in a file in the ITS path /mimes/system/99/script/. You store private interfaces in the ITS path /mimes/<service>/99/script/, whereby <service> is the name of your application.

The interfaces must satisfy all of the SAP_<Name>Interface naming conventions. You can define both methods and attributes for your interfaces. You only have to implement the interface once for all pages in the application.

An implementation could look like the following example:

```
`SAP_WebFrameworkUseInterface( “SAP_XInterface”, “public” )`
<SCRIPT>
      function SAP_XInterface.m1( a, b, c ) {...}
      function SAP_XInterface.m2( a ) {...}
      SAP_Xinterface.p1 = 42;
      SAP_Xinterface.p2 = “hello”;
</SCRIPT>
```

Event Concept

The event concept is used for communication between services. However, when triggering these events, the system does not know which application will react to the event.

An example of this is the LaunchEvent. You can use this event to trigger an event in your application that is interpreted from the mySAP.com Workplace application in the form that the transaction specified in the parameters is started by the Workplace.

In the SAP Web Framework (SWF), events are passed up from the service in the service hierarchy (Bubble-up Events). This means that, for example, when a MiniApp triggers the LaunchEvent in the Workplace, this event is first passed to the mySAP.com Workplace application.

There is a naming convention for the events. The names are put together in the following way: SAP_<Name>Event.

The event is called via the SWF function *SAPWF_raiseEvent* (evtName, data).

A service can register for this event and handle this event if an event handler is implemented in the service. This is specified using the name of a function that must be implemented in the service: **`SAP_<Name>Event_handler( evtObj )`**

Provided that an application successfully processed the event, the service receives a reference to the event object:

```
result = SAPWF_raiseEvent(evtName, data);
if ( result == zero ) {Event was not processed}
else {Event was processed}
```

SAP_LaunchEvent

The *SAP_LaunchEvent* allows you to start a service from your service.

To do this, you trigger the event using the *SAPWF_raiseEvent ( )* function:

result = SAPWF_raiseEvent("SAP_LaunchEvent", "data");

If the result is <> zero, the Workplace successfully handled the event.

The *data* information depends on the type of service that you want to call.

Part of this information is always the *target* parameter, with which you specify where the service is to be started.

Parameter value	Description
inplace	The service is started in the WorkSpace (that is, in one of the Workplace channels).
external	The service is started in a separate browser window.
hidden	The service is started but is not visible to the user.

At the moment, there is a differentiation between the following types of services:

Transactions

You call a transaction using the *SAP_LaunchEvent* by specifying the *SAP_TRANSACTION* service type the name of the *logSys* logical system (for example, WPPCLNT800), the *tCode* transaction code (for example, FK01), and the *target* target (for example, *external* to open the transaction in a window outside the Workplace). If you want to start the transaction with parameters, you can supply a parameter string that you can create with the SAP Web Framework function SAPWF_objectToQueryString() [Seite 27], for example.

You call transaction FK01 in logical system WPPCLNT800 with parameters so that the transaction starts in the Workplace.

```
result = SAPWF_raiseEvent( "SAP_LaunchEvent",
                   { serviceType:        "SAP_TRANSACTION",
                     logSys:             "WPPCLNT800",
                     tCode:              "FK01",
                     params:             SAPWF_objectToQueryString(
paramObj ),
                     target:             "inplace" }
                     );
```

Alternatively, you can also call the *SAP_launchR3Transaction* function. This requires the same information, apart from the service type:

```
result = SAP_launchR3Transaction( logSys , Tcode , Params , target );
```

Documentation

You call up the documentation using the *SAP_LaunchEvent* by specifying the service type *SAP_SIMPLE_DOCU_URL*, the URL *url*, and the target *target* (for example, internally, to open the documentation in place of the LaunchPad).

You call the documentation so that the transaction is started in the Workplace.

```
result = SAPWF_raiseEvent( "SAP_LaunchEvent",
         { serviceType:    "SAP_SIMPLE_DOCU_URL",
           url: http://aiokeh.wdf.sap-
ag.de:1080/SAPIrExtHelp/IWB_EXTHLP.asp?_LOIO=CCD8B6E5531011D4B50E0060
941931ED&LANGUAGE=DE&RELEASE=2.10&IWB_INDUSTRY=WORKPLACE,
           target:         "intern"
          }
          );
```

Alternatively. you can also call the *SAP_launchSimpleSAPDocumentation* function. This requires the same information, apart from the service type:

```
result = SAP_launchSimpleSAPDocumentation( url , target );
```

URL

You call up a URL using the *SAP_LaunchEvent* by specifying the service type *SAP__URL*, the URL *url*, and the target *target* (for example, internally, to open the page in the WorkSpace).

You call the transaction URL http://www.mysap.com so that the transaction is started in the Workplace.

```
result = SAPWF_raiseEvent( "SAP_LaunchEvent",
                             { serviceType:    "SAP_URL",
                               url:            "http://www.mysap.com
",
                               target:         "intern"
                             }
                           );
```

```
Alternatively. you can also call the SAP_launchUrl function. This
requires the same information, apart from the service type:
result = SAP_launchUrl( url , target );
```

Functions of the SAP Web Framework

As already described in the introduction, the SAP Web Framework (SWF) provides functions that make programming Web applications easier.

For the most part, the functions are implemented independently of the ITS infrastructure. However, additional functions are available for Web application within an ITS structure. These build on the core functions of the SWF. These functions are also listed here. They are marked accordingly.

SAPWF_getNowSecs()

This function provides you with the current browser time in seconds. It specifies the number of seconds that have passed since 01/01/1970.

You can use this information, for example, if you want the system to perform actions at regular intervals.

Interface

This function does not require any parameters.

It returns an integer.

```
var oldtime = SAPWF_getNowSecs();
...
if (SAPWF_getNowSecs() > oldtime) {
do something};
```

SAPWF_getVersion()

This function tells you which version of the SAP Web Framework (SWF) is installed on the Web server on which your application is running.

Interface

This function does not require any parameters.

The function returns the current SWF version in the following format: *4.6d/02*. This example indicates ITS release 4.6D and SWF at Support Package level 02.

```
if (SAPWF_getVersion() == "4.6d/00" ) {
do something};
```

SAPWF_getUserAgentVersion()

This function provides you with the type and the version of the browser being used.

Interface

This function does not require any parameters.

The function returns an object with the following attributes:

Attribute	Description
agentType	Browser type: IE: Microsoft Internet Explorer NN: Netscape Navigator
agentVersion	Browser release

```
if   (SAPWF_getUserAgentVersion().agentType   =   'IE')   AND
(SAPWF_getUserAgentVersion().agentVersion) = '5.0' {

do something};
```

SAPWF_isInternetExplorer()

This function verifies whether the browser in which your application is running is the Microsoft Internet Explorer.

If you also want to know the version of the browser, you should use the SAPWF_getUserAgentVersion() [Seite 19] function instead.

Interface

This function does not require any parameters.

The function returns a Boolean type. It returns **`true`** if the browser is a Microsoft Internet Explorer. Otherwise it returns **`false`**.

```
if ( SAPWF_isInternetExplorer() ){

do something};
```

SAPWF_isNetscapeNavigator()

This function verifies whether the browser in which your application is running is the Netscape Navigator.

If you also want to know the version of the browser, you should use the SAPWF_getUserAgentVersion() [Seite 19] function instead.

Interface

This function does not require any parameters.

The function returns a Boolean type. It returns **`true`** if the browser is a Netscape Navigator. Otherwise it returns **`false`**.

```
if ( SAPWF_isNetscapeNavigator() ){
do something};
```

SAPWF_isPageScriptable()

This function checks if the current page can access a certain HTML page with JavaScript functions. In this way you can prevent the system from accessing pages in other domains, for example.

Interface

The function requires a reference to the page to be checked as a parameter.

The function returns a Boolean type. It returns **`true`** if the page can be accessed using JavaScript. Otherwise it returns **`false`**.

```
if ( SAPWF_ isPageScriptable( ref ) ) {
do something};
```

SAPWF_isSWFPage()

This function is an extension of the SAPWF_isPageScriptable() [Seite 21] function. It checks whether the current page can access a specified HTML page using JavaScript functions and whether the page also supports SAP Web Framework (SWF) functions.

Interface

The function requires a reference to the page to be checked as a parameter.

The function returns a Boolean type. It returns **`true`** if the page supports the SWF. Otherwise it returns **`false`**.

```
if ( SAPWF_ isSWFPage( ref ) ) {
do something};
```

SAPWF_getServiceName()

This function tells you in which service the current page was called. This may be useful if a page is used in many different services.

This function is mainly used in the mySAP.com Workplace service.

Interface

This function does not require any parameters.

The function returns the name of the application as a string.

```
var actual_service = SAPWF_getServiceName();
```

SAPWF_getServiceInstance()

This function provides you with the ID of the current instance of the service in which the current page is used.

This function is mainly used in the mySAP.com Workplace service.

Interface

This function does not require any parameters.

The function returns the address of the current instance of the application as a string.

```
var actual_instance = SAPWF_getServiceInstance() {
do something};
```

SAPWF_checkPageRelation()

This function checks whether the current application and another page belong to the same application.

This function is mainly used in the mySAP.com Workplace service.

Interface

The function requires a reference to the page to be checked as a parameter.

The function returns a Boolean type. It returns **`true`** if the page belongs to the current service.

```
if ( SAPWF_checkPageRelation( ref ) ) {
do something};
```

SAPWF_isServiceRootPage()

This function checks whether there is a higher-level page (parent) for the current page or whether the current page is higher than all the other objects in the service (root).

Interface

This function does not require any parameters.

The function returns a Boolean type. It returns **`true`** if there are no higher-level pages. Otherwise it returns **`false`**.

```
if ( SAPWF_ isServiceRootPage() ) {
do something};
```

SAPWF_getServiceRootPage()

This function finds the highest page (root page) in the hierarchy for the current service.

Interface

This function does not require any parameters.

It returns a reference to the window object of the root page.

```
if ( SAPWF_getServiceRootPage == mypage ) {
do something};
```

SAPWF_getRelatedPages()

This function provides you with an array of all the pages belonging to a service.

Interface

This function does not require any parameters.

It returns an array that lists references to the window objects of all the pages in the current service.

```
for (i in SAPWF_getRelatedPages() ) {
do something};
```

SAPWF_getChildServices()

This function provides you with a list of all services that are directly dependent on the current service (that is, that appear directly under the current service in the hierarchy).

Interface

This function does not require any parameters.

It returns an array that lists references to the window objects of all the services that are dependent on the current service. The array contains a reference to the topmost page of each dependent service.

```
for (i in SAPWF_getChildServices() ) {
do something};
```

SAPWF_getAncestorService()

This function provides you with the service directly above the current service in the object hierarchy. For example, this function would return the Workplace for a MiniApp in the mySAP.com Workplace.

Interface

This function does not require any parameters.

It returns a reference to the window object of the root page of the parent application. If there is no parent page, the function returns a reference to the current service.

You want to determine the parent service and then display the URL:

```
var ancestor = SAPWF_getAncestorService();
document.write( ancestor.location.href );
```

SAPWF_getObject()

This function provides you with a reference to an object in your application. It assumes that you have registered all your objects. If you use the HTML Business Template Library, the objects are registered automatically. You must define any other objects using the SAPWF_registerObject function.

The application must also define a unique ID for the objects you create. If the object IDs are not unique, the function returns a reference to the first object found that matches the specified ID.

Interface

The function requires the object ID as a parameter.

The function returns a reference to the object.

```
SAPWF_getObject( "myLabel" ).setLabel ( "ready" );
```

SAPWF_getInterface()

This function provides you with a reference to an interface (see also: Interface Concept [Seite 8]).

In the section on interfaces you can find all the interfaces that are available with the SAP Web Framework. You can use these interfaces for your application.

Prerequisites

Before you can use an interface, you must declare it in the application. To do so, you must add a line to your application:

`SAP_WebFrameworkUseInterface("Interfacename" "public")´ for a public interface called *Interfacename*.

`SAP_WebFrameworkUseInterface("Interfacename" "private")´ for a private interface called *Interfacename*.

Interface

The function requires the interface ID as a parameter.

The function returns a reference to the interface.

```
SAPWF_getInterface( "SAP_RefreshInterface").setInterval( 12
);
```

SAPWF_parseUrl()

This function splits up a URL into its component parts.

Interface

The function requires the URL as a string.

It returns an object with the following parts:

Parameter	Description
protocol	Contains the Web protocol (for example, http).
server	Contains the name of the Web server.
path	Contains the path that is used to communicate with the service.
query	Contains the parameters with which the service is started.
fragment	Contains the bookmark in the called page that is to be accessed.

You want to split the following URL into its component parts and display these as text: *http://igwtt.wdf.sap-ag.de:1080/scripts/wgate/sapwp/!?~login=abc&~password=de#second*

```
var parseUrl = SAPWF_parseUrl( http://igwtt.wdf.sap-
ag.de:1080/scripts/wgate/sapwp/!?~login=abc&~password=de#sec
ond );

document.write( parseUrl.protocol );

document.write( "<br>" );

document.write( parseUrl.server );

document.write( "<br>" );

document.write( parseUrl.path );

document.write( "<br>" );

document.write( parseUrl.query );

document.write( "<br>" );
document.write( parseUrl.fragment );
document.write( "<br>" );
```

SAPWF_parseServer()

This function allows you to split up the server-dependent part of a URL. To determine the server-dependent part of a URL, you can use the SAPWF_parseUrl() [Seite 25] function, for example.

Interface

The function requires the server-dependent part of the URL as a string.

It returns an object with the following parts:

Parameter	Description
userInfo	If there is any user information specified in the URL, this information is provided in this parameter.
host	Host name of the Web server.
port	Port of the Web server.

You now want to split up the information on the Web server that you found using the SAPWF_parseUrl() [Seite 25] function into its component parts.

```
var parseServer = SAPWF_parseServer( parseUrl.server );
document.write( parseServer.userInfo );
document.write( "<br>" );
document.write( parseServer.host );
document.write( "<br>" );
document.write( parseServer.port );
document.write( "<br>" );
```

SAPWF_parseQuery()

This function allows you to convert the parameter part of a URL to a table of value pairs (parameter name, parameter value). To determine the parameter part of a URL, you can use the SAPWF_parseUrl() [Seite 25] function, for example.

Interface

The function requires the parameter part of the URL as a string.

It returns an array with the following parts:

Parameter	Description
index	Name of the parameter.
value	Value of the parameter.

SAPWF_objectToQueryString()

This function converts an object to a query string that can be used in a URL. The attribute name is used as the parameter name and the attribute value is used as the parameter value.

Interface

This function requires an object.

It returns a string that can be used as the parameter part of a URL.

SAPWF_createUrl()

This function allows you to create a URL.

Interface

This function requires an object with the following attributes:

Parameter	**Description**
protocol	Contains the Web protocol (for example, http)
server	Contains the name of the Web server
path	Contains the path that is used to communicate with the service
query	Contains the parameters with which the service is started
fragment	Contains the bookmark in the called page that is to be accessed.

It returns a string containing the new URL.

You want to create a URL from its individual components:

```
var myurl = SAPWF_createUrl( {protocol: "http",
                             server:   "www.mysap.com",
                             path:
"/scripts/wgate/myservice/!",
                             query:    "~client=050&~languag
e=EN",
                             fragment: "test" });
```

SAPWF_addQueryToUrl()

This function appends the parameter part to a URL. To create the parameter part of a URL, you can use the SAPWF_objectToQueryString() [Seite 27] function, for example.

The parameter part is appended to the URL using *?* or *&*, depending on whether the URL provided already has a parameter part or not.

Interface

This function requires an object with the following attributes:

Parameter	Description
url	URL in string format
query	The parameter part of the URL in string format

It returns a string containing the new URL.

SAPWF_setCookie()

This function allows you to create a cookie.

Interface

This function requires the following parameters:

Parameter	Optional	Description
name	no	Name of the cookie
value	no	Value of the cookie
expires	yes	Expiry date
path	yes	Filter for sending the cookie. The cookie is only sent to those services that can be accessed using this path.
domain	yes	Filter for the domains to which the cookie is to be sent.
secure	yes	Boolean: True: https; False: http

This function does not return a value.

SAPWF_getCookieArray()

This function returns a list of all cookies that are currently active in the browser and that belong to the domain of the current page.

Interface

This function does not require any parameters.

It returns an array containing the names of the cookies.

SAPWF_getCookie()

This function provides you with the value of a cookie.

Interface

The function requires a string containing the name of the cookie.

It returns a string containing the value of the cookie.

SAPWF_deleteCookie()

This function allows you to delete a cookie.

⚠

The browser does not delete the cookie immediately. Therefore use this function with caution.

Interface

This function requires the following parameters:

Parameter	Optional	Description
name	no	Name of the cookie.
path	yes	Filter for sending the cookie. Only services that can be accessed by this path are deleted.
domain	yes	Domain to which the cookie that is to be deleted is sent.

This function does not return a value.

SAPWF_encode()

This function allows you to convert an object to a string. You can use it, for example, to create a string that you can then use as the value of a cookie.

Interface

This function requires an object.

It returns a string.

SAPWF_decode()

This function allows you to convert a string to an object. You can use it, for example, to convert the value of a cookie to an object.

Interface

This function requires a string.

It returns an object.