SAP HTML Viewer (BC-CI)

HELP.BCCIHTML

Release 4.6C

Copyright

Icons

Icon	Meaning
⚠	Caution
💬	Example
➡	Note
🧭	Recommendation
‹SYN›	Syntax
💡	Tip

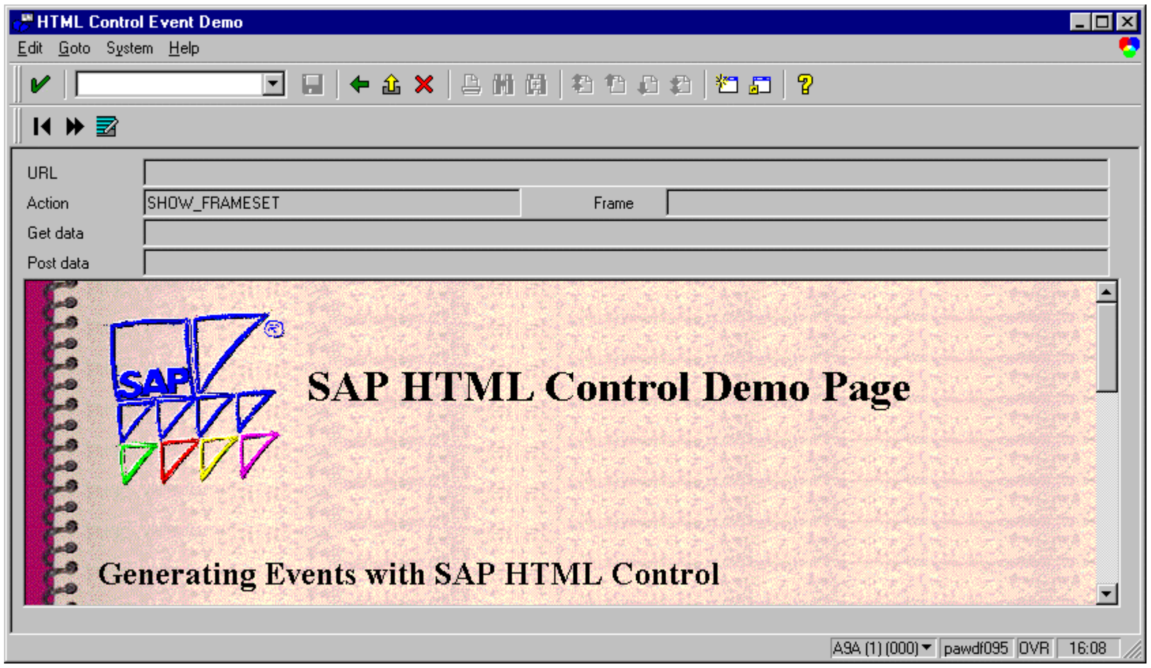
Contents

SAP HTML Viewer (BC-CI)

Purpose

SAP HTML Viewer is a control developed by SAP for use within the SAPgui. SAP has not developed its own Web Browser. Instead, the SAP HTML Viewer uses the Microsoft Internet Explorer Web Browser Control. This is supported by Microsoft Internet Explorer Version 4.0 and above.

The graphic below shows the SAP HTML Viewer in an R/3 screen:



URL			
Action	SHOW_FRAMESET	Frame	
Get data			
Post data			

The screen comes from the example program SAPHTML_EVENTS_DEMO.

Implementation considerations

The interface of the SAP HTML Viewer is the same for all platforms, but its functions will depend on the underlying HTML browser. Under SAPGUI for Windows , it uses Microsoft Internet Explorer 4.0 as an external browser.

Integration

You can use the SAP HTML Viewer with other controls (for example, buttons and list boxes) on a screen in an R/3 transaction. It allows you to display HTML pages, graphics, and pictures in your transactions, either retrieved from the database or generated at runtime.

Features

As well as displaying data, the SAP HTML Viewer can also trigger events in reaction to user input. You can react to these events in your application program (see Registering and Processing Events [Page 11]). Events are triggered when the user chooses a URL or sends an HTML document. The URL or document data is made available to the ABAP program.

The SAP HTML Viewer also supports navigation in HTML pages and MIME documents from sources outside the current R/3 transaction (for example, from a desktop file system or an HTTP server). This is made possible by a close integration of the 4.6 release of SAPgui with your Internet and Intranet environment.

Constraints

SAPGUI for HTML

The SAP HTML Viewer will only work on frontend platforms on which Microsoft Internet Explorer version 4.0 (or higher) is installed.

The SAPgui installation **does not** install Microsoft Internet Explorer. You need a SAPgui installation with Release 4.6 or higher.

Certain features of the SAP HTML Viewer are **not available** (or behave differently) under SAPGUI for HTML. The following components are affected:

Methods	**Events**
go_back [Page 23]	sapevent [Page 14]
go_forward [Page 24]	navigate_complete [Page 13]
go_home [Page 25]	
do_refresh [Page 26]	
stop [Page 22]	
get_current_url [Page 27]	

SAPGUI for Java

Certain functions of this control may not work in the SAPGUI for Java environment. Others may work with restrictions. For further details, refer to the documentation provided with the SAPGUI for Java.

Instance for the SAP HTML Viewer

Definition

You define this instance with reference to the class **`cl_gui_html_viewer`**:

DATA html_viewer TYPE REF TO cl_gui_html_viewer.

Use

An instance of the SAP HTML Viewer administers all of the relevant information for an HTML browser on your screen. You can call the methods of this instance to define and change the attributes of the SAP HTML Viewer.

Integration

The class **`cl_gui_html_viewer`** contains both control-specific methods [Page 19] and methods of the Control Framework [Ext.].

Using the SAP HTML Viewer

This section lists the functions that are specific to the SAP HTML Viewer.

Prerequisites

The process described here is an extension of the general process for using controls [Ext.] that is specific to the SAP HTML Viewer. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

1. Create a reference variable for the SAP HTML Viewer.

```
DATA html_viewer TYPE REF TO cl_gui_html_viewer.
```

2. Create an instance [Page 20] of the SAP HTML Viewer:

```
CREATE OBJECT html_viewer
EXPORTING parent  = container.
```

Register the Events

3. Register the required events of the SAP HTML Viewer. It supports the following events:

Event name	Meaning
SAPEVENT	Click on a special HTML link that triggers an SAP event
NAVIGATE_COMPLETE	HTML page fully loaded

Using the SAP HTML Viewer

Load HTML pages [Page 28] or parts [Page 29] of HTML pages.

4. Display [Page 21] the loaded objects.
5. Navigate forwards [Page 24] and backwards [Page 23] in the stack of your HTML browser from the application program, or return to the homepage [Page 25].
6. Update [Page 26] the HTML page from your application program.
7. Find out the address [Page 27] of the current page.

Destroying the Control

The lifetime management [Ext.] is normally responsible for destroying any controls you use. However, the following two steps allow you to destroy the control yourself:

8. Destroy [Page 44] the SAP HTML Viewer at the frontend. If you no longer need the control container, release it as well:

CALL METHOD html_viewer->free.

9. Delete the reference variable to the SAP HTML Viewer and, if necessary, the reference to the control container as well:

   FREE html_viewer.

   The program SAPHTML_EVENTS_DEMO shows how you can use the SAP HTML Viewer.

Registering and Processing Events

Purpose

The event mechanism of the Control Framework allows you to use handler methods in your programs to react to events triggered by the control (for example, a double-click).

Prerequisites

The following description has been generalized to apply to all custom controls. For more information specific to a particular control, refer to that control's documentation.

Process Flow

1. Assume you are working with a custom control that has the ABAP wrapper **`cl_gui_xyz`**.

```
DATA my_control TYPE REF TO cl_gui_xyz.
```

Registering Events with the Control Framework

2. Define an internal table (type **`cntl_simple_events`**) and a corresponding work area (type **`cntl_simple_event`**).

```
DATA events TYPE cntl_simple_events.
DATA wa_events TYPE  cntl_simple_event.
```

3. Now fill the event table with the relevant events. To do this, you need the event ID (**`event_id`** field). You can find this information in the Class Browser by looking at the attributes of the class **`cl_gui_xyz`**. You must also decide whether the event is to be a system event (**`appl_event = ' '`**) or an application event (**`appl_event = 'X'`**).

```
wa_events-eventid = event_id.
wa_events-appl_event = appl_event.
APPEND wa_events TO events.
```

4. You must now send the event table to the frontend so that it knows which events it has to direct to the backend.

```
CALL METHOD my_control->set_registered_events
        events = events.
```

To react to the events of you custom control, you must now specify a handler method for it. This can be either an instance method or a static method.

Processing an Event Using an Instance Method

5. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**). There is also a default event parameter **`sender`**, which is passed by all events. This contains the reference to the control that triggered the event.

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
METHODS Event_Handler
  FOR EVENT event_name OF cl_gui_xyz
  IMPORTING event_parameter
          sender.
ENDCLASS.
```

6. Register the handler methods with the ABAP Objects Control Framework for the events.

```
DATA event_receiver TYPE REF TO lcl_event_receiver.
CREATE OBJECT event_receiver.
SET HANDLER event_receiver->Event_Handler
         FOR my_control.
```

Processing an Event Using a Static Method

7. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**).

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
CLASS-METHODS Event_Handler
      FOR EVENT event_name OF cl_gui_xyz
      IMPORTING event_parameter
              sender.
ENDCLASS.
```

8. Register the handler methods with the ABAP Objects Control Framework for the events.

```
SET HANDLER lcl_event_receiver=>Event_Handler
         FOR my_control.
```

Processing Control Events

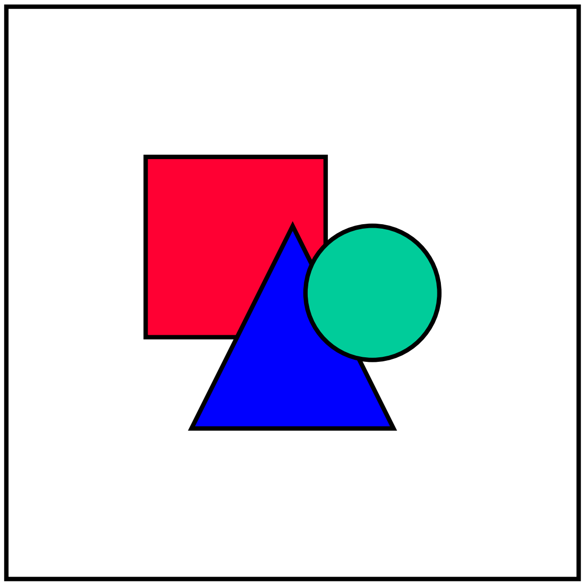
9. You define how you want the system to react to an event in the implementation of the handler method.

```
CLASS lcl_event_receiver IMPLEMENTATION.
METHOD Event_Handler.
* Event processing
ENDMETHOD
ENDCLASS.
```

10. If you registered your event as an application event, you need to process it using the method **`CL_GUI_CFW=>DISPATCH`**. For further information, refer to Event Handling [Ext.].

navigate_complete

Use



Due to the Control Framework architecture, this event may not always be transmitted from the frontend back to the application server. Consequently, **you should never make your program flow dependent on receiving the event**.

The event **`navigate_complete`** is triggered when a page has finished loading.

Event	**Event ID** html_viewer->	**Meaning**
navigate_complete	m_id_navigate_complete	HTML page fully loaded

Features

The event returns the following parameters:

Parameter	**Meaning**
url	URL of the page loaded in the browser.

Activities

Read the general process [Page 11] for working with events in the Control Framework.

sapevent

Use

This event is triggered whenever an event defined on the HTML page with type **`SAPEVENT`** is triggered by the user.

Event	**Event ID** html_viewer->	**Meaning**
sapevent	m_id_sapevent	The user clicked an HTML link with type **`SAPEVENT`**.

Prerequisites

When you define the HTML page that you want to display, define special HTML events (**`SAPEVENT`**). HTML events that are to trigger this control event must have the following format: **SAPEVENT:<action>?data.**

Using SAPEVENT in the SAPGUI for HTML

- To ensure that this event is triggered properly in the HTML GUI, you may only code it in one of the following forms:
  - **`<FORM name=myform action="SAPEVENT:xxx">`**
  - **`<A HREF=SAPEVENT:xxx>`**

    If you code the event in any other way (for example, within a script), **it will not be recognized** and therefore **will not be passed back to the application server**.
- You must also **never** use **`SAPEVENT`** in an **`OnLoad`** event, since this causes an endless loop. After an event, the Internet Transaction Server reloads the relevant page (triggering a new **`OnLoad`** event). Hence each **`SAPEVENT`** would implicitly cause a new **`OnLoad`** which, in turn, triggers another **`SAPEVENT`**.

Features

The event returns the following parameters:

Parameters	**Meaning**
action	Action parameter of the **`SAPEVENT`**.
frame	Name of the HTML frame in which the event was triggered.
getdata	Data in the event string
postdata	Data table for post data
query_table	Event table in the form '**`Name`** | **`Value`**'.

HTML events can send data as well as actions. If you use the 'POST' method in the HTML source code, the data is not sent with the event string. Instead, it is sent in a parallel datastream. This data is available in the table **`postdata`**. The table `query_table` is a version of table `post_data` in pairs of values. The data must have the following format: **Param1=Value1&Param2=Value2.**

Activities

Read the general process [Page 11] for working with events in the Control Framework.

Using Controls in a WAN

When you use controls in your programs, you place an extra load on the communication channel between the frontend and backend. In a LAN, and particularly in a WAN environment, this can be a critical factor.

The problem is alleviated somewhat by buffering mechanisms (see also Automation Queue [Ext.]). Use these points as a guideline to using controls in a WAN.

The documentation for the individual controls also contains more specific notes about using that control in a WAN.

Using CL_GUI_CFW=>FLUSH

The method CL_GUI_CFW=>FLUSH [Page 36] synchronizes the automation queue and the ABAP variables in it. Calling it often generates a synchronous RFC call from the application server to the frontend. To optimize the performance of your application, you should call this method as little as possible.

It is often a good idea to read all control attributes in a single automation queue (for example, at the beginning of the PAI) and retrieve them in a single synchronization. You should, in particular, do this when you read attributes that are not necessary in your event handlers or the PAI/PBO cycle.

You do not need to include a "safety flush" at the end of the PBO to ensure that all method calls are transported to the frontend. A flush at the end of the PBO is guaranteed. Consequently, you cannot construct an automation queue spread over several screens.

There is no guarantee that an automation queue will be sent when you call `CL_GUI_CFW=>FLUSH`. The queue recognizes whether it contains any return values. If this is not the case, it is not sent.

If you have a queue with no return values, and want to ensure that it is synchronized, you can use the Control Framework method CL_GUI_CFW=>UPDATE_VIEW [Page 39]. You should only use this method if you absolutely need to update the GUI. For example, you might have a long-running application in which you want to provide the user with regular updates on the status of an action.

After you have read the attributes of a control, the contents of the corresponding ABAP variables are not guaranteed until after the next flush. The contents of the ABAP variables remain undefined until this call occurs. In the future, there will be cases in which this flush is unnecessary. They will be recognized by the automation queue and the corresponding flush call will be ignored.

Creating Controls and Passing Data

Creating controls and passing data to them is normally a one-off procedure, which in comparison to using normal screen elements can be very runtime-intensive. You should therefore not use any unnecessary controls, or pass unnecessary data to the controls that you are using.

A typical example is a tabstrip control with several tab pages. If the pages contain controls, you should consider using application server scrolling instead of local scrolling, and not loading the controls until the corresponding page is activated by the user. The same applies to passing data to the controls on tab pages.

If you want to differentiate between LAN and WAN environments when you pass data to a control, you can use the function module **`SAPGUI_GET_WANFLAG`**. In some applications, you may need to pass different amounts of data or use a complete fallback in a WAN application. The environment affects, for example, the number of same-level nodes that you can transfer to a tree control without having to introduce artificial intermediate levels.

Unlike screen elements, controls only have to be created and filled with data once. From a performance point of view, this means that they become more profitable the longer they exist. In applications that are called repeatedly, and therefore initialized repeatedly, controls can have a negative effect on performance. In applications that use the same screen for a long time, on the other hand, you may find that using controls results in improved performance.

You can always use the performance tools [Ext.] to check the advantages and disadvantages in terms of network load that using a control brings.

Storing Documents, Picture, and Other Data

Release 4.6A sees the introduction of a frontend cache for accessing documents from the Business Document Service (BDS). You are strongly recommended to store desktop documents, images, and other data in the BDS and not in the R/3 database. Documents from the BDS can be cached at the frontend, and therefore only have to be loaded over the network once.

Special Considerations for the SAP HTML Viewer

Note the following when you use the SAP HTML Viewer.

1. If you use background pictures and graphics that only have cosmetic value on the HTML page, you should only use them over the WAN if they are in GIF or JPEG format.
2. Avoid unnecessary events (**`SAPEVENT`**).
3. If you have graphics that are necessary for data retrieval, use an event (**`SAPEVENT`**) to load them as and when they are needed.
4. Try to save your HTML documents on an HTTP server. This enables you to load the data asynchronously, independent of the application server, and to take advantage of the Internet Explorer cache.

Methods of Class CL_GUI_HTML_VIEWER

This class contains both specific methods for SAP Picture and inherited methods from the Control Framework. However, this section deals only with the methods specific to SAP Toolbar. For information about the Control Framework methods, refer to the Methods of the Control Framework [Page 33] section.

constructor

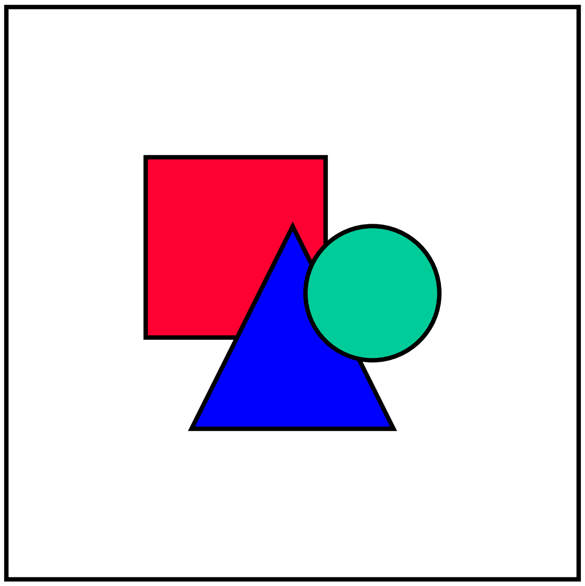
You use this method to instantiate the SAP HTML Viewer.

```
CREATE OBJECT html_viewer
     EXPORTING shellstyle       = shellstyle
            parent          = parent
            saphtmlp          = saphtmlp
            lifetime          = lifetime
     EXCEPTIONS cntl_error        = 1
             cntl_install_error = 2
             dp_install_error   = 3
             dp_error          = 4.
```

Parameter	Meaning
lifetime	Lifetime management [Ext.] parameter. The following values are permitted: **`html_viewer->lifetime_imode`**: The control remains alive for the duration of the internal session (that is, until the session is ended by one of the following statements: **`leave program. leave to transaction. set screen 0, leave screen.`**). After this, the finalize [Page 45] method is called. **`html_viewer->lifetime_dynpro`**: The control remains alive for the lifetime of the screen instance, that is, for as long as the screen remains in the stack. After this, the free [Page 44] method is called. Using this mode automatically regulates the visibility of the control. Controls are only displayed when the screen on which they were created is active. When other screens are active, the controls are hidden. **`html_viewer->lifetime_default`**: If you create the control in a container, it inherits the lifetime of the container. If you do not create the control in a container (for example, because it is a container itself), the lifetime is set to **`html_viewer ->lifetime_imode`**.
saphtmlp	'X': HTML pages are formatted for scriptlets and applets `' '`: HTML pages are formatted normally
Shellstyle	Controls the appearance and behavior of the control You can pass any constants from the ABAP include <**`CTLDEF`**> that begin with WS. You can combine styles by adding the constants together. The default value sets a suitable combination of style constants internally.
parent	Container in which the SAP HTML Viewer can be displayed (**see also** SAP Container [Ext.]).

show_url

You use this method to display data in the SAP HTML Viewer.



There is no longer a distinction between this method and the `show_data` method. From Release 4.6C onwards, you should **always use this method** to display data in the HTML Viewer, regardless of its source.

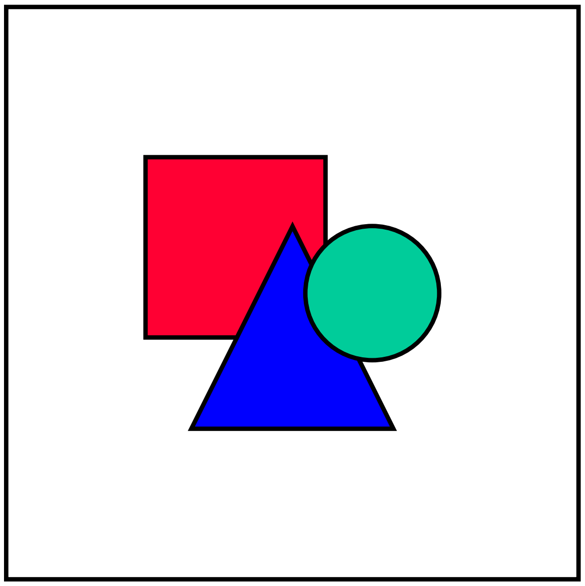
```
CALL METHOD html_viewer->show_url
   EXPORTING  url        = url
              frame      = frame
             in_place    = in_place
   EXCEPTIONS cntl_error = 1.
```

Parameter	Opt.	Meaning
url		URL address of the page that you want to display.
frame	X	Name of the HTML frame in which you want to display the page.
in_place		Indicates where the page should be displayed. Possible values: • `'X'` (default): Page is displayed in the SAPGUI window • `' '`: A separate browser window is opened to display the page

stop

Use this method to stop loading an HTML page.

```
CALL METHOD html_viewer->stop
    EXCEPTIONS cntl_error = 1.
```

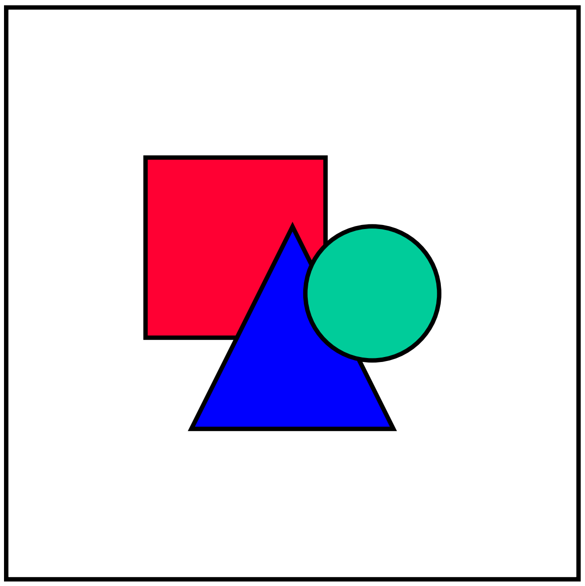


Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

go_back

Use this method to navigate backwards to a page in the navigation stack of the HTML browser.

```
CALL METHOD html_viewer->go_back
   EXCEPTIONS cntl_error = 1.
```

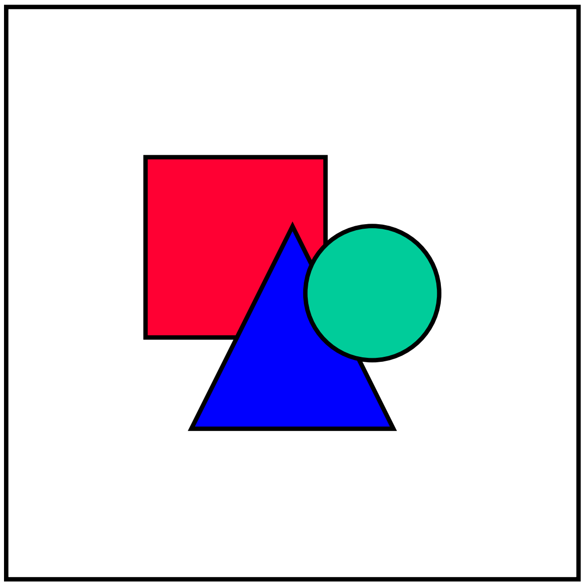


Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

go_forward

Use this method to navigate forwards to a page in the navigation stack of the HTML browser.

```
CALL METHOD html_viewer->go_forward
   EXCEPTIONS cntl_error = 1.
```



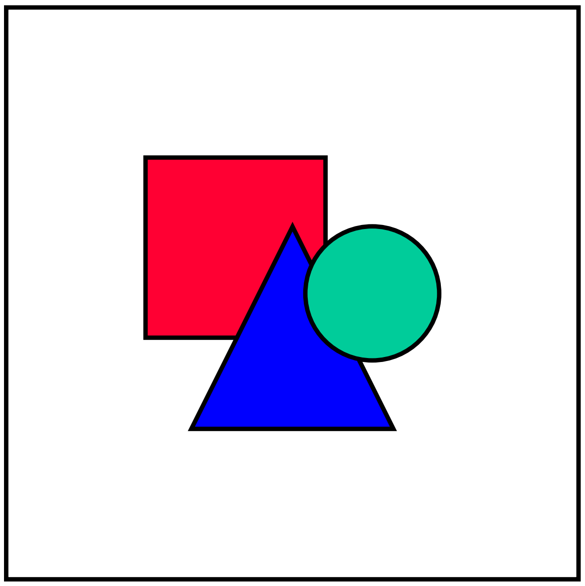
Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

go_home

Use this method to display the homepage of the HTML browser installed on the frontend.

CALL METHOD html_viewer->go_home

    EXCEPTIONS cntl_error = 1.



Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

do_refresh

Use this method in your ABAP program to reload the HTML page currently displayed.

```
CALL METHOD html_viewer->do_refresh
    EXCEPTIONS cntl_error = 1.
```

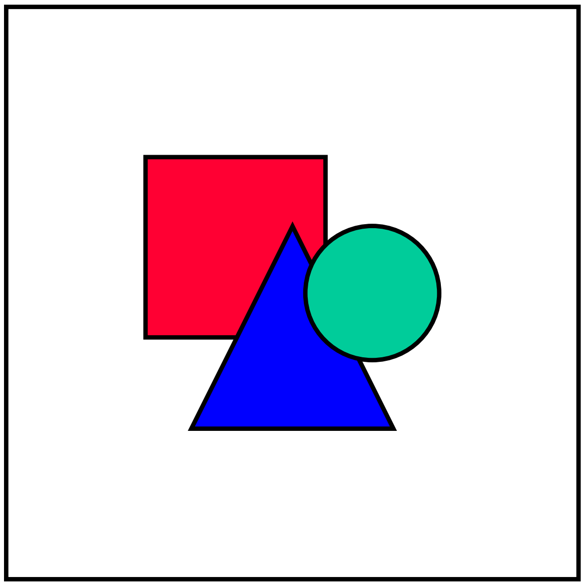
Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

get_current_url

Use this method to find out the URL address of the HTML document currently being displayed.

```
CALL METHOD html_viewer->get_current_url
    IMPORTING url = url
    EXCEPTIONS cntl_error = 1.
```

Parameter	Meaning
url	URL of the page displayed in the browser.



Any occurrences of this method in programs running under SAPGUI for HTML are **ignored**, since it cannot be properly executed in that environment.

load_html_document

Use this method to load a HTML document from the SAP Web Repository. To maintain data in the SAP Web Repository, use Transaction SMW0.

```
CALL METHOD html_viewer->load_html_document
   EXPORTING document_id      = document_id
         document_textpool  = document_textpool
         document_url       = document_url
         as_compressed_data =
   IMPORTING assigned_url      = assigned_url
   CHANGING  merge_table       = merge_table
   EXCEPTIONS document_not_found   = 1
          dp_error_general     = 2
          dp_invalid_parameter = 3.
```

Parameter	Meaning
document_id	ID of the document in the SAP Web Repository
document_textpool	Program containing the text pool to be merged into the document. Text elements in the HTML document (format: **<!text-xxx!>, where xxx is the ID of the text in the text pool) are automatically replaced by texts from the text pool.**
document_url	URL to be assigned to the document. If you do not specify one, the system assigns a unique URL.
assigned_url	If you did not assign a URL in the **`document_url`** parameter, the system places the URL that it assigned into this parameter. If you did assign the URL yourself, the parameter contains the URL from the **`document_url`**.
merge_table	Table containing data to be merged into the current HTML document.
as_compressed_data	By default, the HTML Viewer compresses the data before transmitting it to the frontend control. In some cases (for example, after a SAPscript HTML conversion), this may cause problems. You can therefore switch this function on and off: • `'X'`: Compress data • `' '`: Do not compress data

load_mime_object

Use this method to load an object of any type from the SAP Web Repository and send it to the SAP HTML Viewer. To maintain data in the SAP Web Repository, use Transaction SMW0.

All SAP icons are already contained in Transaction SMW0. You can find them by searching for binary data in development class SWWW.

The executable program **SHOWICON** contains a list of all SAP icons.

```
CALL METHOD html_viewer->load_mime_object
    EXPORTING object_id = object_id
            object_url = object_url
    IMPORTING assigned_url = assigned_url
    EXCEPTIONS object_not_found     = 1
            dp_error_general     = 2
            dp_invalid_parameter = 3.
```

Parameter	Meaning
object_id	ID of the document in the SAP Web Repository
object_url	URL to be assigned to the document. If you do not specify one, the system assigns a unique URL.
assigned_url	If you did not assign a URL in the **document_url** parameter, the system places the URL that it assigned into this parameter. If you did assign the URL yourself, the parameter contains the URL from the **document_url**.

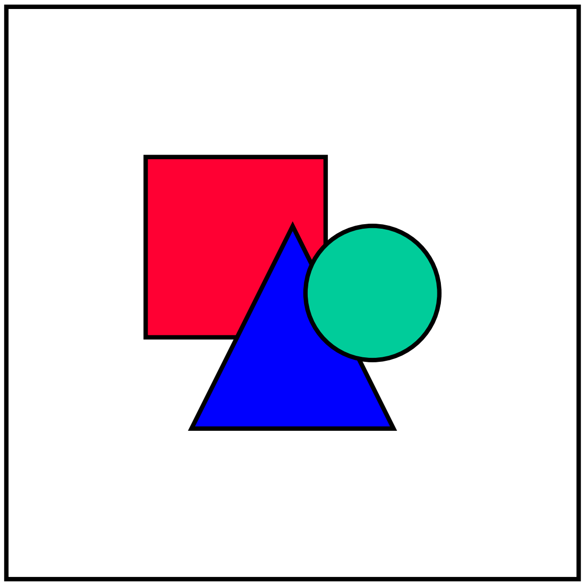
load_data

Use this method to send data from the ABAP program to the presentation server. On the presentation server, the data is available under the URL that you specify.

```
CALL METHOD html_viewer->load_data
   EXPORTING url          = url
         type         = type
         subtype      = subtype
         size         = size
   IMPORTING assigned_url = assigned_url
   CHANGING  data_table   = data_table
   EXCEPTIONS dp_invalid_parameter = 1
          dp_error_general     = 2.
```

Parameter	Meaning
url	URL with which you want to access the data.
type	Data type as a MIME type (for example, text).
subtype	Data subtype as a MIME type (for example, html).
size	Data size in bytes.
assigned_url	If you did not assign a URL in the **`url`** parameter, the system places the URL that it assigned into this parameter. If you did assign the URL yourself, the parameter contains the URL from the **`url`**.
data_table	Table containing data.

show_data



From Release 4.6C, this method is obsolete. You should now use the show_url [Page 21] method to display data in the HTML Viewer, regardless of its source. Occurrences of **`show_data`** in programs from previous releases are, however, still valid and do not need to be changed.

This method displays, in the control, the data sent to the front end using the methods load_mime_object [Page 29], load_data [Page 30], and load_html_document [Page 28]. If you have not already sent the data to the frontend using one of these methods, use the method show_url [Page 21] instead.

```
CALL METHOD html_viewer->show_data
      EXPORTING url   = url
             frame = frame
      EXCEPTIONS cntl_error = 1.
```

Parameter	Meaning
url	URL address of the document.
frame	Name of the HTML frame in which you want to display the page.

show_url_in_browser

Use this method to show the given URL in a separate browser window.

From Release 4.6C, this method is obsolete. You should now use the show_url [Page 21] method to display data in the HTML Viewer, regardless of its source. Occurrences of **`show_data`** in programs from previous releases are, however, still valid and do not need to be changed.

```
CALL METHOD html_viewer->show_url_in_browser
          EXPORTING URL = URL.
```

Parameter and Type	Opt.	Description
URL **`TYPE C`**		URL of the page you want to display in the browser.

Methods of the Control Framework

This section describes the methods of the Control Framework that you need to implement the SAP HTML Viewer.

Methods of Class CL_GUI_CFW

The class **`CL_GUI_CFW`** contains static methods that apply to all instantiated custom controls when you call them.

dispatch

Use this method to dispatch application events (**see** Event Handling [Ext.]) to the event handlers registered for the events. If you do not call the method within the PAI event of your application program, it is called automatically by the system after the PAI has been processed. The method returns a return code from which you can tell if the call was successful.

```
CALL METHOD cl_gui_cfw=>dispatch
     IMPORTING return_code = return_code.
```

Parameters	Description
return_code	**`cl_gui_cfw=>rc_found`**: The event was successfully directed to a handler method. **`cl_gui_cfw=>rc_unknown`**: The event was not registered in the event list. **`cl_gui_cfw=>rc_noevent`**: No event was triggered in a control. The function code was therefore a normal one (for example, from a menu entry). **`cl_gui_cfw=>rc_nodispatch`**: No handler method could be assigned to the event.

An event can only be dispatched once. After that, it is "spent". Consequently, attempting to dispatch the events a second time does not trigger the handler events again.

flush

Use this method to synchronize the automation queue [Ext.]. The buffered operations are sent to the frontend using GUI RFC. At the frontend, the automation queue is processed in the sequence in which you filled it.

If an error occurs, an exception is triggered. You must catch and handle this error. Since it is not possible to identify the cause of the error from the exception itself, there are tools available in the Debugger and the SAPgui to enable you to do so.

Debugger: Select the option *Automation Controller: Always process requests synchronously*. The system then automatically calls the method **`cl_gui_cfw=>flush`** after each method called by the Automation Controller.

SAPGUI: In the SAPgui settings, under *Trace*, select *Automation*. The communication between the application server and the Automation Controller is then logged in a trace file that you can analyze at a later date.

```
CALL METHOD cl_gui_cfw=>flush
          EXCEPTIONS CNTL_SYSTEM_ERROR = 1
                CNTL_ERROR = 2.
```

Do not use any more synchronizations in your program than are really necessary. Each synchronization opens a new RFC connection to the SAPgui.

get_living_dynpro_controls

This method returns a list of reference variables to all active custom controls.

```
CALL METHOD cl_gui_cfw=>get_living_dynpro_controls
                  IMPORTING control_list = control_list.
```

Parameters	Description
`control_list`	List of reference variables of active custom controls. The list has the type **`CNTO_CONTROL_LIST`** (defined in class **`CL_GUI_CFW`**).

set_new_ok_code

You may only use this method in the handler method of a system event. It sets an **`OK_CODE`** that triggers PAI processing. This means that data is transferred from the screen to the program, and you can take control of the program in your PAI modules.

```
CALL METHOD cl_gui_cfw=>set_new_ok_code
        EXPORTING new_code = new_code
        IMPORTING      rc = rc.
```

Parameters	**Description**
new_code	Function code that you want to place in the **`OK_CODE`** field (**`SY-UCOMM`**).
return_code	**`cl_gui_cfw=>rc_posted`**: The OK_CODE was set successfully and the automatic field checks and PAI will be triggered after the event handler method has finished. **`cl_gui_cfw=>rc_wrong_state`**: The method was not called from the handler method of a system event. **`cl_gui_cfw=>rc_invalid`**: The **`OK_CODE`** that you set is invalid.

update_view

Calling the flush [Page 36] method only updates the automation queue if the queue contains return values.

If you have a queue with no return values, and want to ensure that it is synchronized, you can use the Control Framework method **`CL_GUI_CFW=>UPDATE_VIEW`**. You should only use this method if you absolutely need to update the GUI. For example, you might have a long-running application in which you want to provide the user with regular updates on the status of an action.

```
CALL METHOD cl_gui_cfw=>update_view
          EXCEPTIONS CNTL_SYSTEM_ERROR = 1
                CNTL_ERROR        = 2.
```

Methods of Class CL_GUI_OBJECT

The class **`CL_GUI_OBJECT`** contains important methods for custom control wrappers. The only one relevant for application programs is the is_valid [Page 41] method.

is_valid

This method informs you whether a custom control for an object reference still exists at the frontend.

```
CALL METHOD my_control->is_valid
        IMPORTING result = result.
```

Parameters	Description
result	0: Custom control is no longer active at the frontend 1: Custom control is still active

is_alive

This method informs you whether a custom control for an object reference still exists at the frontend.

```
CALL METHOD my_control->is_alive
    RETURNING state = state.
```

Parameters	Description
state	**`my_control->state_dead`**: Custom control is no longer active at the frontend **`my_control->state_alive`**: Custom control is active on the current screen. **`my_control->state_alive_on_other_dynpro`**: Custom control is not active on the current screen, but is still active (but invisible) at the frontend.

Methods of Class CL_GUI_CONTROL

The class **`CL_GUI_CONTROL`** contains methods that you need to set control attributes (for example, displaying the control), register events, and destroy controls.

free

Use this method to destroy a custom control at the frontend. Once you have called this method, you should also initialize the object reference (`FREE my_control`).

```
CALL METHOD my_control->free
    EXCEPTIONS cntl_error        = 1
           cntl_system_error = 2.
```

finalize

This method is redefined by the relevant control wrapper. It contains specific functions for destroying the corresponding control. This method is called automatically by the free [Page 44] method, before the control is destroyed at the frontend.

```
CALL METHOD my_control->finalize.
```

set_registered_events

Use this method to register the events of the control. **See also:** Event Handling [Ext.]

```
CALL METHOD my_control->set_registered_events
     EXPORTING  events          = events
     EXCEPTIONS cntl_error       = 1
            cntl_system_error = 2
       illegal_event_combination = 3.
```

Parameters	Description
events	Table of events that you want to register for the custom control **`my_control`**.

The table **`events`** is a list of the events that you want to register. It is defined with reference to table type **`CNTL_SIMPLE_EVENTS`**. The table type is based on the structure **`CNTL_SIMPLE_EVENT`**, which consists of the following fields:

Field	Description
EVENTID	Event name
APPL_EVENT	Indicates whether the event is a system event (initial) or an application event (X).

The values that you assign to the field **`EVENTID`** are control-specific and therefore described in the documentation of the individual controls.

get_registered_events

This method returns a list of all events registered for custom control **`my_control`**.

```
CALL METHOD my_control->get_registered_events
    IMPORTING  events      = events
    EXCEPTIONS cntl_error = 1.
```

Parameters	Description
events	Table of events that you want to register for the custom control **`my_control`**.

The table **`events`** is a list of the events that you want to register. It is defined with reference to table type **`CNTL_SIMPLE_EVENTS`**. The table type is based on the structure **`CNTL_SIMPLE_EVENT`**, which consists of the following fields:

Field	Description
EVENTID	Event name
APPL_EVENT	Indicates whether the event is a system event (initial) or an application event (X).

The values that you assign to the field **`EVENTID`** are control-specific and therefore described in the documentation of the individual controls.

For general information about event handling, refer to the Event Handling [Ext.] section of the SAP Control Framework documentation.

set_alignment

Use this method to align the custom control within its container:

```
CALL METHOD my_control->set_alignment
     EXPORTING  alignment         = alignment
     EXCEPTIONS cntl_error        = 1
                cntl_system_error = 2.
```

Parameters	Description
alignment	Control alignment

The `alignment` parameter may consist of combinations of the following alignments:

Name	Description
my_control->align_at_left	Alignment with left-hand edge
my_control->align_at_right	Alignment with right-hand edge
my_control->align_at_top	Alignment with top edge
my_control->align_at_bottom	Alignment with bottom edge

You can combine these parameters by adding the components:

alignment = my_control->align_at_left + my_control->align_at_top.

set_position

Use this method to place the control at a particular position on the screen.

⚠

The position of the control is usually determined by its container.

```
CALL METHOD my_control->set_position
    EXPORTING  height        = height
           left          = left
           top           = top
           width         = width
    EXCEPTIONS cntl_error        = 1
           cntl_system_error = 2.
```

Parameters	Description
height	Height of the control
left	Left-hand edge of the control
top	Top edge of the control
width	Width of the control

set_visible

Use this method to change the visibility of a custom control.

```
CALL METHOD my_control->set_visible
    EXPORTING  visible          = visible
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
```

Parameters	Description
visible	`X`: Custom control is visible `' '`: Custom control is not visible

get_focus

This static method returns the object reference of the control that has the focus.

```
CALL METHOD cl_gui_control=>get_focus
    IMPORTING  control          = control
    EXCEPTIONS cntl_error       = 1
            cntl_system_error = 2.
```

Parameters	**Description**
control	Object reference (`TYPE REF TO cl_gui_control`) to the control that has the focus.

set_focus

Use this static method to set the focus to a custom control.

```
CALL METHOD cl_gui_control=>set_focus
     EXPORTING  control          = control
     EXCEPTIONS cntl_error        = 1
            cntl_system_error = 2.
```

Parameters	Description
control	Object reference (**`TYPE REF TO cl_gui_control`**) to the control on which you want to set the focus.

get_height

This method returns the height of the control.

```
CALL METHOD control->get_height
    IMPORTING  height           = height
    EXCEPTIONS cntl_error        = 1.
```

Parameters	Description
height	Current height of the control

get_width

This method returns the width of the control.

```
CALL METHOD control->get_width
    IMPORTING  width           = width
    EXCEPTIONS cntl_error       = 1.
```

Parameters	Description
width	Current width of the control