

# SAP Picture (BC-CI)



HELP.BCIIIMAGE

**Release 4.6C**



## Copyright

© Copyright 2001 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<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> and SQL Server<sup>®</sup> are registered trademarks of Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup>, and OS/400<sup>®</sup> are registered trademarks of IBM Corporation.

ORACLE<sup>®</sup> is a registered trademark of ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP and Informix<sup>®</sup> Dynamic Server<sup>™</sup> are registered trademarks of Informix Software Incorporated.

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







HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

## Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

## Contents

<b>SAP Picture (BC-CI)</b> .....	<b>5</b>
SAP Picture Instance .....	7
Creating a Control: SAP Picture Example .....	8
Using the SAP Picture .....	10
Registering and Processing Events .....	12
SAP Picture Events .....	14
Using Controls in a WAN .....	16
Special Considerations for the SAP Picture.....	18
<b>Methods of Class CL_GUI_PICTURE</b> .....	<b>19</b>
constructor.....	20
load_picture_from_url .....	21
clear_picture .....	22
set_display_mode .....	23
load_picture_from_sap_icons .....	24
display_context_menu.....	25
<b>Methods of the Control Framework</b> .....	<b>26</b>
<b>Methods of Class CL_GUI_CFW</b> .....	<b>27</b>
dispatch .....	28
flush.....	29
get_living_dynpro_controls .....	30
set_new_ok_code .....	31
update_view .....	32
<b>Methods of Class CL_GUI_OBJECT</b> .....	<b>33</b>
is_valid .....	34
free .....	35
<b>Methods of Class CL_GUI_CONTROL</b> .....	<b>36</b>
finalize .....	37
set_registered_events.....	38
get_registered_events.....	39
is_alive .....	40
set_alignment.....	41
set_position .....	42
set_visible.....	43
get_focus.....	44
set_focus.....	45
get_height.....	46
get_width.....	47

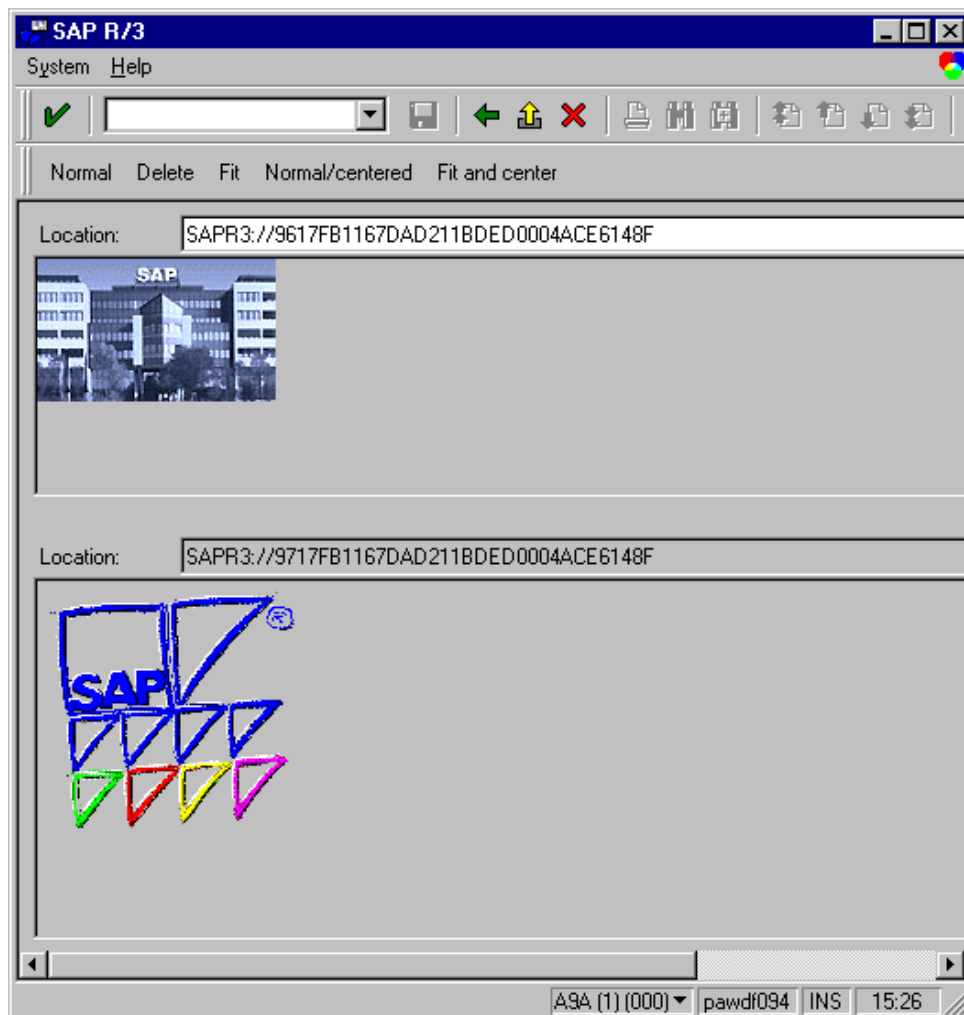
## SAP Picture (BC-CI)

### Purpose

SAP Picture allows you to display graphics in BMP, JPG, or GIF format on screens.



The program `SAP_PICTURE_DEMO` is an example in which two pictures are displayed on a screen:



A second example, `SAP_PICTURE_DEMO_ICON`, demonstrates how you can display SAP icons.

### Requirements

To use SAP Picture, you must have at least basic knowledge of [ABAP Objects \[Ext.\]](#). You should also be familiar with the [SAP Control Framework \[Ext.\]](#).

---

**SAP Picture (BC-CI)****Integration**

The graphics are identified using a URL address. This can be an Internet address (http or ftp), an address in a directory (file), or a memory address on the frontend (sapr3, generated by the Data Provider).

**Features**

SAP Picture contains functions for displaying and resizing graphics and icons.

## SAP Picture Instance

### Definition

You define this instance with reference to the class `cl_gui_picture`:

```
DATA PICTURE TYPE REF TO CL_GUI_PICTURE.
```

### Use

A SAP Picture instance administers all of the information relating to a picture on your screen. You can call the methods of this instance to define and change the attributes of the picture.

### Integration

The class `cl_gui_picture` contains both [control-specific methods \[Page 19\]](#) and methods of the [Control Framework \[Page 26\]](#).

## Creating a Control: SAP Picture Example

# Creating a Control: SAP Picture Example

## Prerequisites

The following process applies to all SAP custom controls. The programming examples use the SAP Picture Control. However, to apply the example to other controls, you would only have to change the name of the control class.

The example also assumes that you are using the custom control in a Custom Container. The SAP Container documentation contains details of further scenarios.

## Process Flow

### Create the Instance

1. Define a reference variable for the Custom Container in which you want to place the custom control (see [SAP Container \[Ext.\]](#)).

```
DATA container TYPE REF TO cl_gui_custom_container.
```

2. Define a reference variable for the SAP Picture:

```
DATA picture TYPE REF TO cl_gui_picture.
```

3. Create the Custom Container. You must already have created the area 'CUSTOM' for the Custom Container in the Screen Painter. When you create the container, you must also specify its [lifetime \[Ext.\]](#) (see [constructor \[Ext.\]](#)).

```
CREATE OBJECT container  
  EXPORTING container_name = 'CUSTOM'  
           lifetime = lifetime.
```

4. Create the SAP Picture Control. You can also specify a lifetime for the SAP Picture, but it must not be longer than that of its container.

```
CREATE OBJECT picture  
  EXPORTING parent = container  
           lifetime = lifetime.
```

### Register the Events

5. There are three steps: Registering the events with the Control Framework, defining a handler method, and registering the handler method. These steps are explained under [Registering and Processing Events \[Page 12\]](#).

### Use the Control

6. These steps are control-specific and therefore not described here.

### Destroy 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:

7. Use the method [free \[Page 35\]](#) to destroy the Custom Control at the frontend. If you no longer need the control container, release it as well:

## Creating a Control: SAP Picture Example

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



Pay careful attention to the sequence in which you destroy controls at the frontend. When you destroy a container, all controls in it are automatically destroyed as well. If you have already destroyed a control and try to destroy it again, an error occurs. You can check whether a control has already been destroyed using the method [is\\_alive \[Page 40\]](#).

8. Delete the reference variables to the custom control and the control container.

```
FREE PICTURE.
FREE CONTAINER.
```

## Using the SAP Picture

# Using the SAP Picture

This section lists the functions that are specific to the SAP Picture.

## Prerequisites

The process described here is an extension of the [general process for using controls \[Page 8\]](#) that is specific to SAP Picture. 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. Define a reference variable for the SAP Picture:

```
DATA picture TYPE REF TO cl_gui_picture.
```

2. [Create an instance \[Page 20\]](#) of the SAP Picture:

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

## Registering the Events

3. Register the events for the SAP Picture that you want to handle. The control supports the following events:

Event	Meaning
EVENTID_PICTURE_CLICK	Single-click on the graphic
EVENTID_PICTURE_DBLCLICK	Double-click on the graphic
EVENTID_CONTROL_CLICK	Single-click within the control area
EVENTID_CONTROL_DBLCLICK	Double-click within the control area
EVENTID_CONTEXT_MENU	Right mouse click (or Shift+F10) to call a context menu
EVENTID_CONTEXT_MENU_SELECTED	Item chosen from a context menu

## Using SAP Picture

1. Load a [graphic \[Page 21\]](#) or a [SAP icon \[Page 24\]](#).

```
CALL METHOD picture->load_picture_from_url
  EXPORTING url = 'http://www.xxx.com/images/picture.gif'
  EXCEPTIONS error = 1 .
```

2. [Define \[Page 23\]](#) how you want the graphic to appear.

---

**Using the SAP Picture**

```
CALL METHOD picture->set_display_mode
  EXPORTING display_mode = picture->display_mode_stretch
  EXCEPTIONS error = 1.
```

3. If necessary, [delete \[Page 22\]](#) the graphic from the display:

```
CALL METHOD picture->clear_picture.
```

### **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:

4. Destroy the custom control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD picture->free.
```

5. Delete the reference variables to the SAP Picture and the control container.

```
FREE PICTURE.
```

## Registering and Processing Events

# 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 your 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
```

## Registering and Processing Events

```
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 \[Ext.\]](#).

## SAP Picture Events

## SAP Picture Events

### Use

Certain user actions on the SAP Picture trigger events:

Event	Event ID	Meaning
	picture->	
picture_click	eventid_picture_click	Single-click on the graphic
picture_dbclick	eventid_picture_dbclick	Double-click on the graphic
control_click	eventid_control_click	Single-click within the control area
control_dbclick	eventid_control_dbclick	Double-click within the control area
context_menu	eventid_context_menu	Right mouse click (or Shift+F10) to call a context menu
context_menu_selected	eventid_context_menu_selected	Item chosen from a context menu

Some events also export parameters:

Event	Parameters	Meaning
picture_click	MOUSE_POS_X MOUSE_POS_Y	Relative position of the mouse when the click occurred. The position is determined in relation to the original size of the picture.
picture_dbclick	MOUSE_POS_X MOUSE_POS_Y	Relative position of the mouse when the click occurred. The position is determined in relation to the original size of the picture.
control_click	MOUSE_POS_X MOUSE_POS_Y	Position of the mouse relative the SAP Picture when the click occurred.
control_dbclick	MOUSE_POS_X MOUSE_POS_Y	Position of the mouse relative the SAP Picture when the click occurred.
context_menu		
context_menu_selected	FCODE	Function code of the selected context menu item.

### Integration

To react to an event in your ABAP program, you must have registered it. To do this, use the method [set\\_registered\\_events \[Page 38\]](#). Events that are triggered but for which you are not registered are filtered by the presentation server, and not passed to the application server.

**See also:** [Event Handling \[Ext.\]](#)

## Activities

Read the general [process \[Page 12\]](#) for working with events in the Control Framework.

## Using Controls in a WAN

### 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 \[Page 29\]](#) 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 \[Page 32\]](#). 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

---

## Using Controls in a WAN

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 Picture**

## Special Considerations for the SAP Picture

In addition to the considerations that apply to all controls, you should note the following:

In a WAN environment, you must be particularly careful with pictures that you load using **SAPR3** URLs and the Data Provider. These URLs are generated by the programmer using the function module **DP\_CREATE\_URL**. In this case, you should ensure that the total size of all pictures that you intend to display simultaneously does not exceed 30 KB. For this reason, you should always use GIF or JPG format instead of BMP. (GIF and JPG are compressed, and therefore occupy less memory.)

The same applies if you load pictures from a web server that you address over a WAN.

## Methods of Class CL\_GUI\_PICTURE

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 Picture. For information about the Control Framework methods, refer to the [Methods of the Control Framework \[Page 26\]](#) section.

**constructor****constructor**

You use this method to instantiate the SAP Picture.

```
CREATE OBJECT picture
  EXPORTING lifetime = lifetime
           shellstyle = shellstyle
           parent = parent
  EXCEPTIONS ERROR = 1.
```

Parameter	Meaning
lifetime	<p><a href="#">Lifetime management [Ext.]</a> parameter. The following values are permitted:</p> <p><b>picture-&gt;lifetime_imode</b>: The control remains alive for the duration of the internal session (that is, until the session is ended by one of the following statements: <code>leave program.</code> <code>leave to transaction.</code> <code>set screen 0, leave screen.</code>). After this, the <a href="#">finalize [Page 37]</a> method is called.</p> <p><b>picture-&gt;lifetime_dynpro</b>: 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 <a href="#">free [Page 35]</a> method is called.</p> <p>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.</p> <p><b>picture-&gt;lifetime_default</b>: 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 <b>picture-&gt;lifetime_imode</b>.</p>
Shellstyle	<p>Controls the appearance and behavior of the control</p> <p>You can pass any constants from the ABAP include &lt;CTLDEF&gt; that begin with WS. You can combine styles by adding the constants together. The default value sets a suitable combination of style constants internally.</p>
parent	<p>Container in which the SAP Picture can be displayed (<a href="#">see also SAP Container [Ext.]</a>).</p>

## load\_picture\_from\_url

You use this method to display a graphic in GIF, WMF, or JPG format in the SAP Picture.

```
CALL METHOD picture->load_picture_from_url
  EXPORTING url = url
  IMPORTING result = result
  EXCEPTIONS error = 1.
```

Parameter	Meaning
url	Address under which the picture is stored. This can be an HTTP or FTP address, an address from a file system (FILE), or an address in memory on the frontend generated by the <a href="#">SAP Data Provider [Ext.]</a> .
result	0: Error loading the picture 1: Picture loaded successfully

---

clear\_picture

## clear\_picture

This method clears the current picture from the display.

```
CALL METHOD picture->clear_picture  
      EXCEPTIONS error = 1.
```

## set\_display\_mode

You use this method to set the way in which the graphic is displayed.

```
CALL METHOD picture->set_display_mode
EXPORTING display_mode = display_mode
EXCEPTIONS error = 1.
```

Parameter	Meaning
display_mode	Parameter for the display type. The following values are permitted: <b>picture-&gt;display_mode_stretch</b> : The graphic is resized to fit precisely in the SAP Picture area. <b>picture-&gt;display_mode_fit</b> : The graphic is enlarged proportionally so that it fits either the length or width (or both) of the SAP Picture precisely. <b>picture-&gt;display_mode_normal</b> : The graphic is displayed in its original size. <b>picture-&gt;display_mode_fit_center</b> : Like <b>picture-&gt;display_mode_fit</b> , but the graphic is displayed in the middle of the control area. <b>picture-&gt;display_mode_normal_center</b> : Like <b>picture-&gt;display_mode_fit</b> , but the graphic is displayed in the middle of the control area.

---

`load_picture_from_sap_icons`

## load\_picture\_from\_sap\_icons

You use this method to display an SAP icon.

```
CALL METHOD picture->load_picture_from_sap_icons
  EXPORTING icon = icon
  EXCEPTIONS error = 1.
```

Parameter	Meaning
icon	Name of the SAP icon



You can address the icon using its name, for example, `ICON_ANNOTATION`. To do this, the statement `INCLUDE <ICON>` must appear in your program.

Otherwise, you can address the icons using the form `@xy@`, where `xy` is the relevant icon code.

## display\_context\_menu

You use this method to display a context menu `menu` when the event `context_menu` occurs.

```
CALL METHOD picture->display_context_menu
  exporting context_menu = context_menu
  exceptions error = 1.
```

Parameter	Meaning
context_menu	Instance of the class CL_CTMENU for constructing and displaying a context menu ( <b>see also:</b> <a href="#">Context Menu [Ext.]</a> ).

## Methods of the Control Framework

# Methods of the Control Framework

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

## 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

## 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	<p><b>cl_gui_cfw=&gt;rc_found:</b> The event was successfully directed to a handler method.</p> <p><b>cl_gui_cfw=&gt;rc_unknown:</b> The event was not registered in the event list.</p> <p><b>cl_gui_cfw=&gt;rc_noevent:</b> No event was triggered in a control. The function code was therefore a normal one (for example, from a menu entry).</p> <p><b>cl_gui_cfw=&gt;rc_nodispatch:</b> No handler method could be assigned to the event.</p>



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`

## 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
<code>control_list</code>	List of reference variables of active custom controls. The list has the type <code>CNTO_CONTROL_LIST</code> (defined in class <code>CL_GUI_CFW</code> ).

## 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 <b>OK_CODE</b> field ( <b>SY-UCOMM</b> ).
return_code	<p><b>cl_gui_cfw=&gt;rc_posted</b>: The <b>OK_CODE</b> was set successfully and the automatic field checks and PAI will be triggered after the event handler method has finished.</p> <p><b>cl_gui_cfw=&gt;rc_wrong_state</b>: The method was not called from the handler method of a system event.</p> <p><b>cl_gui_cfw=&gt;rc_invalid</b>: The <b>OK_CODE</b> that you set is invalid.</p>

---

**update\_view****update\_view**

Calling the [flush \[Page 29\]](#) 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 \[Page 34\]](#) method.

---

is\_valid

## 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

## 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.
```

---

**Methods of Class CL\_GUI\_CONTROL****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.

## 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 \[Page 35\]](#) method, before the control is destroyed at the frontend.

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

**set\_registered\_events****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 <code>my_control</code> .

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 <code>my_control</code> .

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.

is\_alive

## 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	<b>my_control-&gt;state_dead:</b> Custom control is no longer active at the frontend <b>my_control-&gt;state_alive:</b> Custom control is active on the current screen. <b>my_control-&gt;state_alive_on_other_dynpro:</b> Custom control is not active on the current screen, but is still active (but invisible) at the frontend.

## 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****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****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 ( <b>TYPE REF TO cl_gui_control</b> ) 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 ( <b>TYPE REF TO cl_gui_control</b> ) to the control on which you want to set the focus.

**get\_height****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