SAP Toolbar (BC-CI)

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®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.

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

ORACLE® is a registered trademark of ORACLE Corporation.

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

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

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

JAVA® is a registered trademark of Sun Microsystems, Inc.

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

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPHIRE, 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

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Caution Icon" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="image" alt="Example Icon" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="image" alt="Note Icon" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="image" alt="Recommendation Icon" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="image" alt="Syntax Icon" /></td>
<td>Syntax</td>
</tr>
<tr>
<td><img src="image" alt="Tip Icon" /></td>
<td>Tip</td>
</tr>
</tbody>
</table>
Content

SAP Toolbar (BC-CI) .......................................................................................................................... 5
Instance for the SAP Toolbar ........................................................................................................... 6
Creating a Control: SAP Picture Example ....................................................................................... 7
Using the SAP Toolbar ..................................................................................................................... 9
Registering and Processing Events ................................................................................................ 11
Events for the SAP Toolbar ............................................................................................................ 13
Using Controls in a WAN ................................................................................................................ 15
Special Considerations for the SAP Toolbar ................................................................................ 17
Methods of Class CL_GUI_TOOLBAR ......................................................................................... 18
  constructor.................................................................................................................................... 19
  add_button .................................................................................................................................. 20
  add_button_group ....................................................................................................................... 22
  fill_buttons_data_table ........................................................................................................... 23
  delete_button ............................................................................................................................ 25
  delete_all_buttons ..................................................................................................................... 26
  set_button_state ....................................................................................................................... 27
  set_button_info ........................................................................................................................ 28
  set_static_ctxmenu ................................................................................................................... 29
  assign_static_ctxmenu_table ................................................................................................. 30
  track_context_menu ................................................................................................................. 32
Methods of the Control Framework ............................................................................................. 33
Methods of Class CL_GUI_CFW .................................................................................................. 34
  dispatch ................................................................................................................................. 35
  flush ........................................................................................................................................ 36
  get_living_dynpro_controls ...................................................................................................... 37
  set_new_ok_code ..................................................................................................................... 38
  update_view ............................................................................................................................ 39
Methods of Class CL_GUI_OBJECT ............................................................................................ 40
  is_valid ................................................................................................................................. 41
  free ......................................................................................................................................... 42
Methods of Class CL_GUI_CONTROL ......................................................................................... 43
  finalize ................................................................................................................................. 44
  set_registered_events .......................................................................................................... 45
  get_registered_events .......................................................................................................... 46
  is_alive ................................................................................................................................. 47
  set_alignment ....................................................................................................................... 48
  set_position ........................................................................................................................ 49
  set_visible ............................................................................................................................ 50
  get_focus .............................................................................................................................. 51
  set_focus .............................................................................................................................. 52
  get_height ............................................................................................................................ 53
  get_width .............................................................................................................................. 54
SAP Toolbar (BC-CI)

Purpose
The SAP toolbar allows you to define a separate toolbar in addition to the normal application toolbar:

Integration
You can embed the SAP Toolbar in any SAP Control Container.

Features
SAP toolbar allows you to create additional toolbars, which may contain the following objects:

- Pushbuttons
- Pushbuttons with a dropdown menu: If the user chooses the pushbutton, the action defined for the pushbutton is triggered. If the user chooses the arrow, a menu appears.
- Menus: A menu is displayed when the user clicks the button.
- Separators
- Pushbutton groups (similar to radio button groups)
- Toggle buttons (like checkboxes)

When the user chooses an item in the SAP Toolbar, control is passed back to the application program using the event control of the Control Framework. The chosen function is passed to the program as an event parameter.

Constraints
SAP Toolbar requires the Microsoft Common Control. You therefore need to install Microsoft Internet Explorer Version 4.0.
Instance for the SAP Toolbar

Definition
You define this instance with reference to the class cl_gui_toolbar:
data toolbar type ref to cl_gui_toolbar.

Use
A SAP Toolbar instance administers all of the information relating to an additional toolbar on your screen. You can call the methods of this instance to define and change the attributes of the toolbar control.

Integration
The class cl_gui_toolbar contains both control-specific methods [Page 18] and methods of the Control Framework [Page 33].
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 11].

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 42] 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 47].

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

FREE PICTURE.
FREE CONTAINER.
Using the SAP Toolbar

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

Prerequisites

The process described here is an extension of the general process for using controls [Page 7] 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.

Your system contains a demonstration program called SAPTOOLBAR_DEMO1.

Creating the Instance

1. Define a reference variable for the SAP Toolbar:
   
   DATA toolbar TYPE REF TO cl_gui_toolbar.

2. Create an instance [Ext.] of the SAP toolbar:
   
   CREATE OBJECT toolbar EXPORTING parent = container.

3. Insert either individual pushbuttons [Page 20] or pushbutton groups [Page 22] into the toolbar control.

   CALL METHOD toolbar->add_button 
   
   EXCEPTIONS cntl_error = 1.

Register the Events

4. Register the events for the SAP Toolbar [Page 13]. It supports the following events:

<table>
<thead>
<tr>
<th>Event name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_SELECTED</td>
<td>Pushbutton chosen</td>
</tr>
<tr>
<td>DROPOUT_CLICKED</td>
<td>Context menu of a pushbutton (type cntb_btype_dropdown and cntb_btype_menu) chosen</td>
</tr>
</tbody>
</table>
Using the SAP Toolbar

Changing Control Attributes at Runtime

5. You can add extra pushbuttons or pushbutton groups.

6. You can change the status [Page 27] of individual pushbuttons:

   CALL METHOD toolbar->set_button_state
   EXPORTING enabled = 'X'
   fcode   = 'FUNC_1'
   EXCEPTIONS cntl_error = 1.

7. You can delete one [Page 25] or all [Page 26] of the pushbuttons:

   CALL METHOD toolbar->DELETE_BUTTON
   exporting fcode = 'FUNC_1'
   exceptions CNTL_ERROR = 1.

8. You can interpret the SAP Toolbar functions chosen by the user using event handler methods.

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:

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

   CALL METHOD toolbar->free.

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

    FREE toolbar.

    The program SAPTOOLBAR_DEMO1 provides an example of how to use the SAP Toolbar Control in an application.
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 = 'S') 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

1. 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.

2. 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.

3. 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

4. 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.

5. 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.].
Events for the SAP Toolbar

Use

When the user chooses a pushbutton in the SAP Toolbar, one of the following events is triggered, depending on the type of the pushbutton:

<table>
<thead>
<tr>
<th>Event</th>
<th>Event ID</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| FUNCTION_SELECTED   | M_ID_FUNCTION_SELECTED | Pushbutton chosen
|                     |          | User chose a menu with type cntb_btype_dropdown or cntb_btype_menu.     |
| DROPDOWN_CLICKED    | M_ID.DropDOWN_CLICKED | Context menu of a pushbutton (type cntb_btype_dropdown and cntb_btype_menu) chosen |

The events have the following parameters:

<table>
<thead>
<tr>
<th>Event</th>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_SELECTED</td>
<td>fcode</td>
<td>Function code of pushbutton</td>
</tr>
<tr>
<td>DROPDOWN_CLICKED</td>
<td>fcode</td>
<td>Function code of pushbutton</td>
</tr>
<tr>
<td></td>
<td>posx</td>
<td>Position of the pushbutton for which the menu should be displayed</td>
</tr>
<tr>
<td></td>
<td>posy</td>
<td>Position of the pushbutton for which the menu should be displayed</td>
</tr>
</tbody>
</table>

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 45]. 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 event handling [Ext].

Features

Event FUNCTION_SELECTED

This event is always triggered when the user chooses a pushbutton or a menu entry from a pushbutton with type cntb_btype_dropdown or cntb_btype_menu.

Pushbuttons with type cntb_btype_dropdown have two parts: On the left-hand side is the actual pushbutton, with a particular function assigned to it, just like a normal pushbutton. If you click the button, the FUNCTION_SELECTED event is triggered. On the right-hand side is a pushbutton with an arrow. If you click this, the event DROPDOWN_CLICKED is triggered.

If the pushbutton has the type cntb_btype_menu, the event DROPDOWN_CLICKED is always triggered.
Events for the SAP Toolbar

The function code of the pushbutton is passed as a parameter of the event FUNCTION_SELECTED. This allows you to identify the pushbutton and react accordingly in your program.

Event DROPDOWN_CLICKED

This event is triggered whenever a menu is requested for a pushbutton. This is only possible for pushbuttons with the type cntb_btype_dropdown and cntb_btype_menu.

The event parameters are the function code and position of the pushbutton. You can use the function code to identify the pushbutton. You can use the position to place the context menu in the correct position.

To construct the context menu, use the methods of class CL_CTMENU. To display it, use the method track_context_menu [Page 32].

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
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 Toolbar

There are no specific problems to bear in mind when you use the SAP Toolbar Control in a WAN.
Methods of Class CL_GUI_TOOLBAR

This class contains both specific methods for the SAP Toolbar 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 Toolbar.

CREATE OBJECT toolbar

    EXPORTING parent  = parent
                  shellstyle = shellstyle
                  lifetime   = lifetime
                  display_mode = display_mode

    EXCEPTIONS cntl_install_error = 1
                           cntl_error         = 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lifetime</td>
<td><strong>Lifetime management [Ext.]</strong> parameter. The following values are permitted:</td>
</tr>
<tr>
<td></td>
<td><strong>toolbar-&gt;lifetime_imode</strong>: 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. leave to transaction. set screen 0, leave screen.</code>). After this, the <strong>finalize [Page 44]</strong> method is called.</td>
</tr>
<tr>
<td></td>
<td><strong>toolbar-&gt;lifetime_dynpro</strong>: 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 <strong>free [Page 42]</strong> 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.</td>
</tr>
<tr>
<td></td>
<td><strong>toolbar-&gt;lifetime_default</strong>: 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 <strong>toolbar-&gt;lifetime_imode</strong>.</td>
</tr>
<tr>
<td>Shellstyle</td>
<td>Controls the appearance and behavior of the control</td>
</tr>
<tr>
<td></td>
<td>You can pass any constants from the ABAP include <code>&lt;CTLDEF&gt;</code> that begin with WS. You can combine styles by adding the constants together. The default value sets a suitable combination of style constants internally.</td>
</tr>
<tr>
<td>parent</td>
<td>Container in which the SAP Toolbar can be displayed (see also <strong>SAP Container [Ext.]</strong>).</td>
</tr>
<tr>
<td>display_mode</td>
<td>The alignment of the toolbar:</td>
</tr>
<tr>
<td></td>
<td>• <code>cl_gui_toolbar=&gt;m_mode_horizontal</code>: Horizontal</td>
</tr>
<tr>
<td></td>
<td>• <code>cl_gui_toolbar=&gt;m_mode_vertical</code>: Vertical</td>
</tr>
</tbody>
</table>
add_button

You use this method to add a new pushbutton to the toolbar.

CALL METHOD toolbar->add_button

  EXPORTING  fcode       = fcode
  icon        = iconid
  is_disabled = is_disabled
  butn_type   = butn_type
  text        = text
  quickinfo   = quickinfo
  is_checked  = is_checked

  EXCEPTIONS cntl_error = 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcode</td>
<td>Function code that will be passed to the application program by an event when the user chooses the pushbutton.</td>
</tr>
<tr>
<td>icon</td>
<td>Icon to be displayed on the pushbutton.</td>
</tr>
</tbody>
</table>
| is_disabled| 'X': Pushbutton is inactive 
           | ' ': Pushbutton is active |
| butn_type  | cntb_btype_button: Pushbutton 
           | cntb_btype_dropdown: Pushbutton with menu 
           | cntb_btype_menu: Menu 
           | cntb_btype_sep: Separator 
           | cntb_btype_group: Pushbutton group 
           | cntb_btype_check: Toggle button |

**Note:** You should use static menus wherever possible, since this helps to eliminate excessive roundtrips. This is especially important under SAPGUI for HTML. For further details, refer to set_static_ctxmenu [Page 29] or assign_static_ctxmenu_table [Page 30].

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Text to be displayed on the pushbutton.</td>
</tr>
<tr>
<td>quickinfo</td>
<td>Quick info for the pushbutton</td>
</tr>
</tbody>
</table>
| is_checked | Only for pushbuttons with type cntb_btype_group and cntb_btype_check: 
           | 'X': Button is chosen 
           | ' ': Button not chosen |

You can address the icon using its name, for example, ICON_ANNOTATION. To do this, the statement INCLUDE <ICON>. must appear in your program. The executable program SHOWICON displays all of the icons in your system.
Otherwise, you can address the icons using the form `@xy@`, where `xy` is the relevant icon code.
add_button_group

You use this method to add a list of pushbuttons to your toolbar.

CALL METHOD toolbar->add_button_group
  EXPORTING  data_table = data_table
  EXCEPTIONS dp_error  = 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_table</td>
<td>Table of pushbuttons that you want to add to the toolbar. You create the table with reference to the type TTB_BUTTON. You can use the method fill_buttons_data_table [Page 23] to fill the table.</td>
</tr>
</tbody>
</table>

You should use static menus wherever possible, since this helps to eliminate excessive roundtrips. This is especially important under SAPGUI for HTML. For further details, refer to set_static_ctxmenu [Page 29] or assign_static_ctxmenu_table [Page 30].
fill_buttons_data_table

You use this method to fill the internal table that you pass to the method \texttt{add_button_group}\,[Page 22] in order to create a set of new pushbuttons in your toolbar.

\begin{verbatim}
CALL METHOD toolbar->fill_buttons_data_table
  EXPORTING  fcode       = fcode
              icon        = iconid
              disabled   = disabled
              butn_type   = butn_type
              text        = text
              quickinfo   = quickinfo
              checked     = checked
  CHANGING   data_table  = data_table.
\end{verbatim}

\begin{tabular}{|l|p{30cm}|}
\hline
Parameters & Meaning \\
\hline
fcode & Function code that will be passed to the application program by an event when the user chooses the pushbutton. \\
icon & Icon to be displayed on the pushbutton. \\
disabled & 'X': Pushbutton is inactive  \\
 & ' ': Pushbutton is active \\
butn_type & \texttt{cntb_btype_button}: Pushbutton  \\
 & \texttt{cntb_btype_dropdown}: Pushbutton with menu  \\
 & \texttt{cntb_btype_menu}: Menu  \\
 & \texttt{cntb_btype_sep}: Separator  \\
 & \texttt{cntb_btype_group}: Pushbutton group \\
 & \texttt{cntb_btype_check}: Toggle button \\
\textbf{Note:} You should use static menus wherever possible, since this helps to eliminate excessive roundtrips. This is especially important under SAPGUI for HTML. For further details, refer to \texttt{set_static_ctxmenu}\,[Page 29] or \texttt{assign_static_ctxmenu_table}\,[Page 30]. \\
text & Text to be displayed on the pushbutton. \\
quickinfo & Quick info for the pushbutton \\
checked & Only for pushbuttons with type \texttt{cntb_btype_group} and \texttt{cntb_btype_check}:  \\
 & 'X': Button is chosen  \\
 & ' ': Button not chosen \\
data_table & Table of pushbuttons that you want to add to the toolbar. \\
 & You create the table with reference to the type \texttt{TTB_BUTTON}. \\
 & You then pass the table to the method \texttt{add_button_group}\,[Page 22]. \\
\hline
\end{tabular}
You can address the icon using its name, for example, `ICON_ANNOTATION`. To do this, the statement `INCLUDE <ICON>` must appear in your program. The executable program `SHOWICON` displays all of the icons in your system.

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

You use this method to delete a pushbutton from the toolbar.

CALL METHOD toolbar->DELETE_BUTTON
  exporting  fcode   = fcode
  exceptions CNTL_ERROR = 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcode</td>
<td>Function code of the pushbutton that you want to delete.</td>
</tr>
</tbody>
</table>

If you created more than one pushbutton with the same function code, the system only deletes the last one.
delete_all_buttons

You use this method to delete all pushbuttons from the toolbar.

CALL METHOD toolbar->DELETE_ALL_BUTTONS

  exceptions CNTL_ERROR = 1.
**set_button_state**

You use this method to change the status of an individual pushbutton:

CALL METHOD toolbar->set_button_state
   EXPORTING  enabled = enabled
               checked = checked
               fcode   = fcode
   EXCEPTIONS cntl_error = 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>’ ‘: Pushbutton is inactive</td>
</tr>
<tr>
<td></td>
<td>’x’: Pushbutton is active</td>
</tr>
<tr>
<td>is_checked</td>
<td>Only for pushbuttons with type cntb_btype_group and cntb_btype_check:</td>
</tr>
<tr>
<td></td>
<td>’x’: Button is chosen</td>
</tr>
<tr>
<td></td>
<td>’ ‘: Button not chosen</td>
</tr>
</tbody>
</table>
**set_button_info**

Use this method to change the text, icon, or quick info for a button.

```plaintext
CALL METHOD toolbar->set_button_info
   EXPORTING fcode = fcode
       icon  = icon
       text  = text
   quickinfo = quickinfo.
```

<table>
<thead>
<tr>
<th>Parameter and Type</th>
<th>Optional</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcode</td>
<td></td>
<td>Function code of the button (used to identify the button - may not be changed)</td>
</tr>
<tr>
<td>icon</td>
<td>X</td>
<td>New icon in the form '@XY@'</td>
</tr>
<tr>
<td>text</td>
<td>X</td>
<td>New text for the button</td>
</tr>
<tr>
<td>quickinfo</td>
<td>X</td>
<td>New quickinfo text for the button</td>
</tr>
</tbody>
</table>
set_static_ctxmenu

Use this method to assign a context menu to a pushbutton for the entire lifetime of the control. When you assign a context menu statically, the system loads it into the frontend control and it remains there after the user has closed it. Normally the system triggers the DROPDOWN_CLICKED event when the user clicks a pushbutton that has a dropdown menu. In your application, you would then assign the context menu to the button, but only for that one occasion. If you assign the context menu statically, it resides at the frontend, and is available each time the user clicks the corresponding pushbutton without you having to reassign it each time.

You should use static context menus in all but the most context-sensitive cases.

Any changes that are made to the context menu during the lifetime of the control are automatically updated at the frontend.

CALL METHOD toolbar->set_static_ctxmenu
   EXPORTING   fcode = fcode
               ctxmenu = ctxmenu

<table>
<thead>
<tr>
<th>Parameter and Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcode</td>
<td>Function code of the button to which you want to assign the context menu. Note: The button must have the type cntb_btype_dropdown or cntb_btype_menu.</td>
</tr>
<tr>
<td>TYPE UI_FUNC</td>
<td></td>
</tr>
<tr>
<td>ctxmenu</td>
<td>Reference variable pointing to the context menu instance you want to assign (see Context Menus [Ext])</td>
</tr>
<tr>
<td>TYPE REF TO CL_CTMENU</td>
<td></td>
</tr>
</tbody>
</table>
**assign_static_ctxmenu_table**

Use this method to assign context menus to a group of pushbuttons for the entire lifetime of the control. When you assign a context menu statically, the system loads it into the frontend control and it remains there after the user has closed it. Normally the system triggers the DROPDOWN_CLICKED event when the user clicks a pushbutton that has a dropdown menu. In your application, you would then assign the context menu to the button, but only for that one occasion. If you assign the context menus statically, they reside at the frontend, and are available whenever the user clicks the corresponding pushbutton without you having to reassign it each time.

You should use static context menus in all but the most context-sensitive cases.

Any changes that are made to the context menu during the lifetime of the control are automatically updated at the frontend.

```plaintext
CALL METHOD toolbar->assign_static_ctxmenu_table
  EXPORTING table_ctxmenu = table_ctxmenu.
```

<table>
<thead>
<tr>
<th>Parameter and Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_ctxmenu</td>
<td>Internal table containing the assignments of context menus to pushbuttons in the toolbar instance. It has the line type STB_BTNMNU (described below).</td>
</tr>
</tbody>
</table>

### Structure STB_BTNMNU

<table>
<thead>
<tr>
<th>Component and Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>The function code of the pushbutton in the toolbar to which you want to assign the context menu</td>
</tr>
<tr>
<td>ctmnu</td>
<td>A reference variable pointing to the context menu you want to assign to the pushbutton (see Context Menus [Ext])</td>
</tr>
</tbody>
</table>
assign_static_ctxmenu_table
track_context_menu

Use this method to display a context menu. It is particularly useful in connection with the events of pushbuttons with type `cntb_btype_dropdown` and `cntb_btype_menu`.

CALL METHOD toolbar->track_context_menu

    EXPORTING context_menu = menu
    posx = posx
    posy = posy

EXCEPTIONS ctmenu_error = 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context_menu</td>
<td>Name of the context menu.</td>
</tr>
<tr>
<td></td>
<td>You create the menu with reference to the class <code>CL_CTMENU</code>.</td>
</tr>
<tr>
<td></td>
<td>You must use this class to fill it.</td>
</tr>
<tr>
<td>posx</td>
<td>Horizontal display position for the menu</td>
</tr>
<tr>
<td>posy</td>
<td>Vertical display position for the menu</td>
</tr>
</tbody>
</table>
Methods of the Control Framework

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

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>return_code</td>
<td>cl_gui_cfw=&gt;rc_found: The event was successfully directed to a handler method.</td>
</tr>
<tr>
<td></td>
<td>cl_gui_cfw=&gt;rc_unknown: The event was not registered in the event list.</td>
</tr>
<tr>
<td></td>
<td>cl_gui_cfw=&gt;rc_noevent: No event was triggered in a control. The function code was therefore a normal one (for example, from a menu entry).</td>
</tr>
<tr>
<td></td>
<td>cl_gui_cfw=&gt;rc_nodispatch: No handler method could be assigned to the event.</td>
</tr>
</tbody>
</table>

⚠️ 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.
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control_list</td>
<td>List of reference variables of active custom controls. The list has the type CNTO_CONTROL_LIST (defined in class CL_GUI_CFW).</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_code</td>
<td>Function code that you want to place in the OK_CODE field (SY-UCOMM).</td>
</tr>
<tr>
<td>return_code</td>
<td>cl_gui_cfw=&gt;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.</td>
</tr>
<tr>
<td></td>
<td>cl_gui_cfw=&gt;rc_wrong_state: The method was not called from the handler method of a system event.</td>
</tr>
<tr>
<td></td>
<td>cl_gui_cfw=&gt;rc_invalid: The OK_CODE that you set is invalid.</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0: Custom control is no longer active at the frontend</td>
</tr>
<tr>
<td></td>
<td>1: Custom control is still active</td>
</tr>
</tbody>
</table>
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

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

**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` 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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td>Table of events that you want to register for the custom control my_control.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENTID</td>
<td>Event name</td>
</tr>
<tr>
<td>APPL_EVENT</td>
<td>Indicates whether the event is a system event (initial) or an application event (X).</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td>Table of events that you want to register for the custom control <code>my_control</code>.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENTID</td>
<td>Event name</td>
</tr>
<tr>
<td>APPL_EVENT</td>
<td>Indicates whether the event is a system event (initial) or an application event (X).</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>my_control-&gt;state_dead: Custom control is no longer active at the frontend</td>
</tr>
<tr>
<td></td>
<td>my_control-&gt;state_alive: Custom control is active on the current screen.</td>
</tr>
<tr>
<td></td>
<td>my_control-&gt;state_alive_on_other_dynpro: Custom control is not active on the</td>
</tr>
<tr>
<td></td>
<td>current screen, but is still active (but invisible) at the frontend.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignment</td>
<td>Control alignment</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_control-&gt;align_at_left</td>
<td>Alignment with left-hand edge</td>
</tr>
<tr>
<td>my_control-&gt;align_at_right</td>
<td>Alignment with right-hand edge</td>
</tr>
<tr>
<td>my_control-&gt;align_at_top</td>
<td>Alignment with top edge</td>
</tr>
<tr>
<td>my_control-&gt;align_at_bottom</td>
<td>Alignment with bottom edge</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Height of the control</td>
</tr>
<tr>
<td>left</td>
<td>Left-hand edge of the control</td>
</tr>
<tr>
<td>top</td>
<td>Top edge of the control</td>
</tr>
<tr>
<td>width</td>
<td>Width of the control</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>X: Custom control is visible</td>
</tr>
<tr>
<td></td>
<td>’ ’: Custom control is not visible</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>Object reference (TYPE REF TO cl_gui_control) to the control that has the focus.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>Object reference (TYPE REF TO cl_gui_control) to the control on which you want to set the focus.</td>
</tr>
</tbody>
</table>
get_height

This method returns the height of the control.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Current height of the control</td>
</tr>
</tbody>
</table>
get_width

This method returns the width of the control.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Current width of the control</td>
</tr>
</tbody>
</table>