SAP Container

HELP.BCCIDOCK

Release 4.6C

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

Contents

SAP Container

Use

A SAP Container is a control that accommodates other controls, such as the SAP Tree Control, SAP Picture Control, SAP Textedit Control, SAP Splitter Control, and so on. It manages these controls logically in a collection, and provides a physical area in which they are displayed.

All controls live in a container. Since containers are themselves controls, you can nest them. The container is the parent of the control within it.

There are five kinds of SAP Containers:

- SAP Custom Container [Page 15]
  The SAP Custom Container allows you to display controls in an area defined on a normal screen using the Screen Painter.
  Class: CL_GUI_CUSTOM_CONTAINER
- SAP Dialog Box Container [Page 18]
  The SAP Dialog Box container allows you to display controls in an amodal dialog box or fullscreen.
  Class: CL_GUI_DIALOGBOX_CONTAINER
- SAP Docking Container [Page 26]
  The SAP Docking Container allows you to attach a control to any of the four edges of a screen as a resizable screen area. You can also detach it so that it becomes an independent amodal dialog box.
  Class: CL_GUI_DOCKING_CONTAINER
- SAP Splitter Container [Page 37]
  The SAP Splitter Container allows you to display more than one control in a given area by dividing it into cells.
  Class: CL_GUI_SPLITTER_CONTAINER
- SAP Easy Splitter Container [Page 62]
  The SAP Easy Splitter Container allows you to divide an area into two cells with a control in each. The cells are separated by a moveable splitter bar.
  Class: CL_GUI_EASY_SPLITTER_CONTAINER

All SAP Containers have a common parent with the type CL_GUI_CONTROL. They are all derived from this basic container, and therefore all have the same object-oriented interface.

Displaying and Relinking Containers

Displaying Containers

You should remember the following points about displaying SAP Containers:

- The default size of a control is the same as the size of its container. You can override this default when you instantiate the control.
- Each container has a top level container (screen0 to 9) its parent. The top level container is automatically created when you use the corresponding ABAP Objects class, and is the equivalent of a dialog box level 0 to 9. The default screen is the current dialog box when you create the control. If you want to specify a particular screen on which to display the control, pass the screen as the parent when you create the container. For further information, refer to the documentation of the relevant constructor method.
- A container can only be displayed at the dialog box level to which it is attached when you create it. When the system leaves a particular level, the lifetime management system destroys all controls from that level. For further information, refer to Lifetime Management [Ext.].
- If you want to display controls dynamically, use the SET_VISIBLE [Page 88] method.
- If a parent is not visible, its children are also not visible.

Relinking Containers

You can only relink containers at the same dialog box level and at the top parent level (for example, Custom Container, Docking Container).

To relink containers, use the LINK [Page 12] method.

Methods

CLASS_CONSTRUCTOR

Use

This method creates the required dialog box levels for the SAP Container.

 Never call this method explicitly yourself. It is called automatically the first time you access a component of the class.

For further information, refer to Displaying and Relinking Containers [Page 7].

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (**`create object`** statement). You generally pass the parameters of the method in the **`create object`** statement. Since the SAP Container is a superclass, its constructor is called indirectly when the constructor method of one of its subclasses is called.

Features

create object container

```
  exporting
    clsid = clsid
    parent = parent
    style = style
    dynnr = dynnr
    repid = repid
    container_name = container_name
    lifetime = lifetime
    autoalign = autoalign
    no_autodef_progid_dynnr = no_autodef_progid_dynnr
  exceptions
    cntl_error = 1
    cntl_system_error = 2
    create_error = 3
    lifetime_error = 4
    lifetime_dynpro_dynpro_link = 5
    lifetime_dynpro_illegal_parent = 6.
```

Parameters	Description	Possible values
clsid	Class ID	
parent	Parent of the instance, that is, the contain in which the control is to be displayed	
style	Controls the appearance and behavior of the control	Constants from the class CL_GUI_CONTROL that begin with WS_*
dynnr	Number of the screen to which you want to attach the control	
repid	Report ID: Program to which you want to attach the control	

container_name	Name of the Custom Container defined in the Screen Painter	
lifetime	Lifetime management parameter specifying the lifetime of the control	**cntl_lifetime_imode**: The control remains alive for the lifetime of the internal session (that is, until a statement such as **leave program** or **leave to transaction**). **cntl_lifetime_dynpro**: The control remains alive for the lifetime of the screen (that is, while it remains in the screen stack). It is not destroyed, for example, by a **call screen** or **call transaction** statement.
autoalign	Automatic alignment of the container	Default: None
no_autodef_progid_dynnr	Switches off automatic definition of the program ID and screen number.	**space** Program ID and screen number are used automatically (default value) Switches off automatic definition of the program ID and screen number.

LINK

Use

Use this method to link a control to a new Custom Container. The reference variable points to the new Custom Container. Consequently, the control is no longer displayed in the old Custom Container.


You can only relink containers at the same dialog box level and at the top parent level (for example, Custom Container, Docking Container).

Features

```
call method container->link

  exporting
    repid = repid
    dynnr = dynnr
    container = container

exceptions

    cntl_error = 1
    cntl_system_error = 2
    lifetime_dynpro_dynpro_link = 3.
```

Parameters	Description	Possible values
dynnr	Number of the screen to which you want to attach the control	
repid	Report ID: Program to which you want to attach the control	
container	Name of the Custom Container defined in the Screen Painter	

Instance Attributes

Instance Attributes	Type / Value	Description / Use
children	CNTO_CONTROL_LIST	List of children

Static Attributes

Static Attribute	Type / Value	Description / Use
default_screen	CL_GUI_CONTAINER	Dummy for default top level container (screen)
scree0 **to** screen9	CL_GUI_CONTAINER	Dummy for default top level0 to top level9 container (screen0 to screen9)

SAP Custom Container

Use

Use the SAP Custom Container to build a control into an area on a screen or subscreen. You define the area occupied by the control in the Screen Painter.

The default size of the control that you place in the Custom Container is the same as that of the container itself.

Prerequisites

Before you can include a control in a Custom Container, you must define the area that the container will occupy. You do this in the Screen Painter. For further information, refer to Creating a Custom Control [Ext.].

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (**`create object`** statement). You generally pass the parameters of the method in the **`create object`** statement.

Features

create object custom_container

```
  exporting
    parent = parent
    container_name = container_name
    style = style
    dynnr = dynnr
    repid = repid
    lifetime = lifetime

  exceptions
    cntl_error = 1
    cntl_system_error = 2
    create_error = 3
    lifetime_error = 4
    lifetime_dynpro_dynpro_link = 5.
```

Parameters	Description	Possible values
parent	Parent of the instance, that is, the contain in which the control is to be displayed	
container_name	Name of the Custom Container defined in the Screen Painter	
style	Controls the appearance and behavior of the control	Constants from the class CL_GUI_CONTROL that begin with WS_*
dynnr	Number of the screen to which you want to attach the control	
repid	Report ID: Program to which you want to attach the control	

lifetime	Lifetime management parameter specifying the lifetime of the control	**cntl_lifetime_imode**: The control remains alive for the lifetime of the internal session (that is, until a statement such as **leave program** or **leave to transaction**) **cntl_lifetime_dynpro**: The control remains alive for the lifetime of the screen (that is, while it remains in the screen stack). It is not destroyed, for example, by a **call screen** or **call transaction** statement.
no_autodef_progid_dynnr	Switches off automatic definition of the program ID and screen number.	**space** Program ID and screen number are used automatically (default value) **'X'** Switches off automatic definition of the program ID and screen number.

SAP Dialog Box Container

Use

Use the SAP Dialog Box Container to display controls in an amodal dialog box that can be moved anywhere on the screen. You can also use it to display a control in full screen.

Unlike the SAP Docking Container [Page 26], you cannot attach a SAP Dialog Box Container to a screen.

Methods

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (`create object` statement). You generally pass the parameters of the method in the `create object` statement.

Features

create object dialogbox_container

```
  exporting
    parent = parent
    width = width
    height = height
    style = style
    dynnr = dynnr
    repid = repid
    lifetime = lifetime
    top = top
    left = left
    caption = caption
    no_autodef_progid_dynnr = no_autodef_progid_dynnr
  exceptions
    cntl_error = 1
    cntl_system_error = 2
    create_error = 3
    lifetime_error = 4
    lifetime_dynpro_dynpro_link = 5.
```

Parameters	Description	Possible values
parent	Parent of the instance, that is, the contain in which the control is to be displayed	
width	Width of the dialog box	
height	Height of the dialog box	
style	Controls the appearance and behavior of the control	Constants from the class CL_GUI_CONTROL that begin with WS_*
dynnr	Number of the screen to which you want to attach the control	
repid	Report ID: Program to which you want to attach the control	

lifetime	Lifetime management parameter specifying the lifetime of the control	**cntl_lifetime_imode**: The control remains alive for the lifetime of the internal session (that is, until a statement such as **leave program** or **leave to transaction**) **cntl_lifetime_dynpro**: The control remains alive for the lifetime of the screen (that is, while it remains in the screen stack). It is not destroyed, for example, by a **call screen** or **call transaction** statement.
top	Position of the dialog box: Top edge	
left	Position of the dialog box: Left-hand edge	
caption	Dialog box caption	
no_autodef_progid_dynnr	Switches off automatic definition of the program ID and screen number.	**space** Program ID and screen number are used automatically (default value) **'X'** Switches off automatic definition of the program ID and screen number.

SET_CAPTION

Use

Use this method to set a new caption in the dialog box.

Features

call method dialogbox_container->set_caption

```
  exporting
    caption = caption
```

exceptions

```
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
caption	Caption for the amodal dialog box	

Events

CLOSE [Page 24]

CLOSE

Use

This event is triggered when the user clicks the *Close* button in the dialog box. By handling the event, you can close the dialog box when the user does this.

For further information, refer to Event Handling [Ext.], and to Lesson 2: Event Handling [Ext.] of the Controls Tutorial.

Features

You need to add the following coding to your program.

Declaration Part

```
data close_event type cntl_simple_events.


class lcl_dialogbox_handler definition.
public section.
   methods: on_dialogbox_close
     for event close of cl_gui_dialogbox_container
     importing sender.
endclass.


class lcl_dialogbox_handler implementation.
  method on_dialogbox_close.
    if not sender is initial.
      call method sender->free
        exceptions
          others = 1.
      if sy-subrc <> 0.
        *Error handling
      endif.
      free dialogbox_container.
      clear dialogbox_container.
      endif.
  endmethod.
endclass.
```

```
data: dialogbox_handler type ref to lcl_dialogbox_handler.
```

After Creating the Dialog Box Container

```
If dialogbox_handler is initial.
    create object dialogbox_handler.
    endif.

  set handler dialogbox_handler->on_dialogbox_close
    for dialogbox_container.
```

SAP Docking Container

Use

Use the SAP Docking Container to attach one or more areas to a screen . You can attach an area to any or all of the four edges of the screen (top, bottom, left, or right). The screen is made smaller to accommodate the docking container. You can detach the docking container from the screen (floating) and reattach it.

The behavior of the areas in the container is determined by the sequence in which they are initialized. Docking Containers are attached to the screen from the inside out. This means that when you create a second container, it is attached to the edge of the screen, and the container that was already there is pushed outwards. From a purely technical point of view, you can attach any number of docking containers to a screen. However, remember that using too many can make your application confusing for the user.

Methods

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (`create object` statement). You generally pass the parameters of the method in the `create object` statement.

Features

create object docking_container

```
  exporting
    parent = parent
    dynnr = dynnr
    repid = repid
    side = side
    extension = extension
    style = style
    lifetime = lifetime
    metric = metric
    caption = caption
    no_autodef_progid_dynnr = no_autodef_progid_dynnr

  exceptions
    cntl_error = 1
    cntl_system_error = 2
    create_error = 3
    lifetime_error = 4
    lifetime_dynpro_dynpro_link = 5.
```

Parameters	Description	Possible values
parent	Parent of the instance, that is, the contain in which the control is to be displayed	
dynnr	Number of the screen to which you want to attach the control	
repid	Report ID: Program to which you want to attach the control	

side	Edge of the screen to which you want to attach the container	docking_container->dock_at_left: **Left-hand edge** docking_container->dock_at_right: **Right-hand edge** docking_container->dock_at_top: **Top edge** docking_container->dock_at_bottom: **Bottom edge**
extension	Extension of the docking container (for example, width of the container when you attach it to the left-hand edge of the screen)	
style	Controls the appearance and behavior of the control	Constants from the class CL_GUI_CONTROL that begin with WS_*
lifetime	Lifetime management parameter specifying the lifetime of the control	**`cntl_lifetime_imode`**: The control remains alive for the lifetime of the internal session (that is, until a statement such as **`leave program`** or **`leave to transaction`**) **`cntl_lifetime_dynpro`**: The control remains alive for the lifetime of the screen (that is, while it remains in the screen stack). It is not destroyed, for example, by a **`call screen`** or **`call transaction`** statement.
metric	Metric	**`docking_container->metric_default`**: Screen metric (default) **`docking_container->metric_pixel`**: Pixels **`docking_container->metric_mm`**: Millimeters
caption	Caption	Any character field

no_autodef_progid_dynnr	Switches off automatic definition of the program ID and screen number.	**space** Program ID and screen number are used automatically (default value) **'X'** Switches off automatic definition of the program ID and screen number.

DOCK_AT

Use

Use this method to reattach the docking container to a different edge of the screen.


You can specify where you want to attach the container to the screen in the CONSTRUCTOR [Page 28] method when you instantiate the control.

Features

```
call method docking_container->dock_at
  exporting
    side = side
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
side	Edge of the screen to which you want to attach the container	docking_container->dock_at_left **Left-hand edge** docking_container->dock_at_right **Right-hand edge** docking_container->dock_at_top: **Top edge** docking_container->dock_at_bottom: **Bottom edge**

FLOAT

This method is currently not active.

Use

Use this method to switch the status of the Docking Container between fixed (docking) and free (floating).

Features

call method docking_container->float

```
  exporting
    do_float = do_float

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
do_float	Status of the docking container	**`docking_container->property_docking`**: Docking Container is fixed (docking) **`docking_container->property_floating`**: Docking Container is free (floating)

GET_EXTENSION

Use

Use this method to get the current extension of the Docking Container. For further information, refer to CONSTRUCTOR [Page 28]

Features

```
call method docking_container->get_extension
  importing
    extension = extension
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
extension	Extension of the Docking Container, that is, the value of the dimension of the container not attached to the screen	

SET_CAPTION

Use

Use this method to set a new caption in the Docking Container.

Features

call method docking_container->set_caption

```
  exporting
    caption = caption

  exceptions

    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
caption	Caption of the Docking Container	

SET_EXTENSION

Use

Use this method to get the current extension of the Docking Container. For further information, refer to GET_EXTENSION [Page 33] and CONSTRUCTOR [Page 28].

Features

```
call method docking_container->set_extension
  exporting
    extension = extension
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
extension	Extension of the Docking Container, that is, the value of the dimension of the container not attached to the screen	

Static Constants

Static Constant	Type / Value	Description / Use
dock_at_left	Type I Value 1	Left-hand edge of the screen
dock_at_top	Type I Value 2	Top edge of the screen
dock_at_bottom	Type I Value 4	Bottom edge of the screen
dock_at_right	Type I Value 8	Right-hand edge of the screen
property_docking	Type I Value 470	Docking property
property_floating	Type I Value 480	Floating property

SAP Splitter Container

Use

Use the SAP Splitter Container to construct a group of control within an area, each of which is placed in a separate cell. The Splitter Control manages and displays the cells. You can change the size of the cells using a splitter bar. Enlarging one cell reduces the size of the adjacent cell.

You can only use the Splitter Container within another container (parent), for example, a Custom Container or Docking Container. You can embed a Splitter Control within a cell of another Splitter Container and thus nest them.

The default setting for the splitter grid is 0X0. The maximum division is 16X16. You can specify the size of lines and columns either in pixels (absolute) or as a percentage (relative). The default setting is relative. You can set the splitter bar to immovable.

If you want to divide a sector into only two cells (either horizontally or vertically), use the SAP Easy Splitter Container [Page 62].

Methods

ADD_CONTROL

Use

Use this method to add a control to a particular cell. An existing control in the same cell is not deleted. You set the visibility of the control using SET_VISIBLE [Page 88].

 The control that you want to insert must already exist at the same dialog box level (you can query this using the IS_ALIVE [Page 83] method).

Features

```
call method splitter_container->add_control

  exporting
    row = row
    column = column
    control = control

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2
    lifetime_error = 3.
```

Parameters	Description	Possible values
row	Line in which you want to insert the column	
column	Column which you want to insert the column	
control	Instance of the control you want to insert	
result	Result code	

ADD_CONTROL_HANDLE

 Do not use this method. Use the method ADD_CONTROL [Page 39] instead.

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (**`create object`** statement). You generally pass the parameters of the method in the **`create object`** statement.

Features

```
create object splitter_container
  exporting
    link_dynnr = link_dynnr
    link_repid = link_repid
    shellstyle = shellststyle
    left = left
    top = top
    width = width
    height = height
    metric = metric
    align = align
    parent = parent
    rows = rows
    columns = columns
    no_autodef_progid_dynnr = no_autodef_progid_dynnr
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
link_dynnr	Number of the screen to which you want to attach the control	
link_repid	Report ID: Program to which you want to attach the control	
Shellstyle	Controls the appearance and behavior of the control	Constants from the class CL_GUI_CONTROL that begin with WS_*
left	Left-hand position of the container	
top	Position of top of container	
width	Width of the container	
height	Height of the container	

CONSTRUCTOR

metric	Metric	**splitter_container->metric_default**: Screen metric (default) **splitter_container->metric_pixel**: Pixels **splitter_container->metric_mm**: millimeters
align	Alignment of the Splitter Container	Constants from the class CL_GUI_CONTROL that begin with ALIGN_*
parent	Parent of the instance, that is, the container in which you want to place the control	
rows	Number of rows	Maximum = 16
columns	Number of columns	Maximum = 16
no_autodef_progid_dynnr	Switches off automatic definition of the program ID and screen number.	**space** Program ID and screen number are used automatically (default value) **'X'** Switches off automatic definition of the program ID and screen number.

GET_CONTAINER

Use

This method returns the reference variable of a selected cell. You use the reference variable to include a new control in the container.

Features

call method splitter_container->get_container

```
  exporting
    row = row
    column = column

  receiving
    container = container
```

Parameters	Description	Possible values
row	Line of the cell	
column	Column of the cell	

GET_COLUMNS

Use

This method returns the number of columns in the Splitter Container.

Features

call method splitter_container->get_columns

```
  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
result	Result	

GET_COLUMN_MODE

Use

This method returns the mode (absolute or relative) of the columns in the Splitter Container. For further information, refer to SAP Splitter Container [Page 37].

Features

```
call method splitter_container->get_column_mode

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
result	Result	

GET_COLUMN_WIDTH

Use

This method returns the width of a column.

Features

call method splitter_container->get_column_width

```
  exporting
    id = id

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the column	
result	Result	

GET_COLUMN_SASH

Use

This method returns the configuration of the columns of the splitter bar.

Features

call method splitter_container->get_column_sash

```
  exporting
    id = id
    type = type

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the splitter bar	The first splitter bar has the value 1, and so on
type	Attribute of the splitter bar	type_movable
result	Result	**`splitter_container->true`**: Splitter bar is movable **`splitter_container->false`**: Splitter bar is fixed

GET_ROWS

Use

This method returns the number of lines in the Splitter Container.

Features

```
call method splitter_container->get_rows

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
result	Result	

GET_ROW_HEIGHT

Use

This method returns the height of a row.

Features

call method splitter_container->get_row_height

```
  exporting
    id = id

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the row	
result	Result	

GET_ROW_MODE

Use

This method returns the mode (absolute or relative) of the rows in the Splitter Container. For further information, refer to SAP Splitter Container [Page 37].

Features

```
call method splitter_container->get_row_mode

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
result	Result	

GET_ROW_SASH

Use

This method returns the configuration of the rows of the splitter bar.

Features

call method splitter_container->get_row_sash

```
  exporting
    id = id
    type = type

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the splitter bar	The first splitter bar has the value 1, and so on
type	Attribute	type_movable
result	Result	**`splitter_container->true`**: Splitter bar is movable **`splitter_container->false`**: Splitter bar is fixed

REMOVE_CONTROL

Use

Use this method to remove a specific control from the Splitter Container.

 The control is not destroyed. If you want to destroy it, you must call the FREE [Page 90] method and, if necessary, destroy the reference variable. For further information, refer to Lifetime Management [Ext.].

Features

```
call method splitter_container->remove_control
  exporting
    row = row
    column = column
  importing
    result = result
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
row	Line of the cell	
column	Column of the cell	
result	Result	

SET_BORDER

Use

Use this method to display or hide the border around the Splitter Container.

 You should switch the border off when you nest Splitter Containers.

Features

call method splitter_container->set_border

```
  exporting
    border = border

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
border	Indicates whether to draw the border	**`cl_gui_cfw=>true`**: Draw the border **`cl_gui_cfw=>false`**: Suppress the border

SET_COLUMN_MODE

Use

Use this method to set the mode of the rows in the Splitter Container to absolute or relative. For further information, refer to SAP Splitter Container [Page 37].

Features

```
call method splitter_container->set_column_mode
  exporting
    mode = mode
  importing
    result = result
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
mode	Column mode	**`splitter_container->mode_absolut`**: Absolutle mode (pixels) **`splitter_container->mode_relative`**: Relative mode (percentage)
result	Result	

SET_COLUMN_WIDTH

Use

Use this method to set the width of a column.

Features

call method splitter_container->set_column_width

```
  exporting
    id = id
    width = width

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the column	
width	Width of the column	
result	Result	

SET_COLUMN_SASH

Use

Use this method to set the configuration of the splitter bar for the columns.

Features

call method splitter_container->set_column_sash

```
  exporting
    id = id
    type = type
    value = value

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the splitter bar	The first splitter bar has the value 1, and so on
type	Attribute of the splitter bar	type_movable
value	Value of the attribute	**`splitter_container->true`**: Movable splitter bar **`splitter_container->false`**: Fixed splitter bar
result	Result	

 If you want to change the splitter bar attribute, change the value of **`type`** to **`type_moveable`** and then choose the corresponding value of the attribute.

SET_GRID

Use

Use this method to create a new cell grid. The existing grid is destroyed. You pass the grid when you create the Splitter Container. For further information, refer to the CONSTRUCTOR [Page 41] method.

 Do not use this method at present - it could cause unexpected results at the frontend.

Features

```
call method splitter_container->set_grid

  exporting
    rows = rows
    columns = columns

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
rows	Number of columns	
columns	Number of rows	
result	Result	

SET_ROW_HEIGHT

Use

Use this method to set the height of a row.

Features

call method splitter_container->set_row_height

```
  exporting
    id = id
    height = height

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the row	
height	Height of the row	
result	Result	

SET_ROW_MODE

Use

Use this method to set the mode of the rows in the Splitter Container to absolute or relative. For further information, refer to SAP Splitter Container [Page 37].

Features

```
call method splitter_container->set_row_mode
  exporting
    mode = mode
  importing
    result = result
  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
mode	Row mode	**`splitter_container_mode_absolut`**: Absolutle mode (pixels) **`splitter_container_mode_relative`**: Relative mode (percentage)
result	Result	

SET_ROW_SASH

Use

Use this method to set the configuration of the splitter bar for the rows.

Features

call method splitter_container->set_row_sash

```
  exporting
    id = id
    type = type
    value = value

  importing
    result = result

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
ID	ID of the splitter bar	The first splitter bar has the value 1, and so on
type	Attribute of the splitter bar	type_movable
value	Value of the attribute	**splitter_container->true**: Movable splitter bar **splitter_container->false**: Fixed splitter bar
result	Result	

If you want to change the splitter bar attribute, change the value of **type** to **type_moveable** and then choose the corresponding value of the attribute.

Static Constants

Static Constant	Type / Value	Description / Use
mode_absolute	Type I Value 0	Absolute mode (pixels)
mode_relative	Type I Value 1	Relative mode (percentage)
type_movable	Type I Value 0	Value for moveable splitter bar
true	Type I Value 1	Attribute value: True
false	Type I Value 0	Attribute value: False

SAP Easy Splitter Container

Use

The SAP Easy Splitter Container is a version of the SAP Splitter Container with a reduced set of features. Use the Easy Splitter Container when you want to display controls in a sector divided into two cells. You can set the splitter bar between the two cells either horizontally or vertically.

You can only use the Easy Splitter Container within another container (parent), for example, a Custom Container or Docking Container.

You can nest Easy Splitter Containers to any depth.

If you need more features than are offered by the Easy Splitter Container, use the SAP Splitter Container [Page 37].

Methods

CONSTRUCTOR

Use

The constructor creates, initializes, and positions the control.

 The constructor method is called automatically when you instantiate the class (`create object` statement). You generally pass the parameters of the method in the `create object` statement.

Features

create object easy_splitter_container

```
  exporting
    link_dynnr = link_dynnr
    link_repid = link_repid
    metric = metric
    parent = parent
    orientation = orientation
    sash_position = sash_position
    with_border = with_border

  exceptions
    cntl_error = 1
    cntl_system_error = 2.
```

Parameters	Description	Possible values
link_dynnr	Number of the screen to which you want to attach the control	
link_repid	Report ID: Program to which you want to attach the control	
metric	Metric	`easy_splitter_container->metric_default`: Screen metric (default) `easy_splitter_container->metric_pixel`: Pixels `easy_splitter_container->metric_mm`: Millimeters
parent	Parent of the instance, that is, the container in which the control is to be displayed	
orientation	Orientation of the splitter bar	`easy_splitter_container->orientation_vertical`: Vertical splitter bar `easy_splitter_container->orientation_horizontal`: Horizontal splitter bar

sash_position	Position of the splitter bar	Percentage from left to right or top to bottom. Default: 50
with_border	Setting for border	0: Without border 1: With border

SET_SASH_POSITION

Use

Use this method to set a new position for the splitter bar.

Features

call method easy_splitter_container->set_sash_position

```
  exporting
    sash_position = sash_position
```

Parameters	Description	Possible values
sash_position	Position of the splitter bar	Percentage from left to right or top to bottom.

Static Attributes

Static attribute	Type / Value	Description / Use
orientation_horizontal	Type I Value 1	Horizontal orientation
orientation_vertical	Type I Value 0	Vertical orientation

Instance Attributes

Instance Attributes	Type / Value	Description / Use
top_left_container	Type ref to CL_GUI_CONTAINER	ID of left-hand or upper container
bottom_right_container	Type ref to CL_GUI_CONTAINER	ID of the right-hand or lower container
pane_orientation	Type I	Orientation of subareas or splitter bar

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 71] 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_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.

constructor

This method is called by the control wrapper when you instantiate a control.

To instantiate a SAP control, always call the constructor of its class.

```
CREATE OBJECT my_control
  EXPORTING   clsid          = clsid
              lifetime       = lifetime
              shellstyle     = shellstyle
              parent         = parent
              autoalign      = autoalign
  EXCEPTIONS cntl_error      = 1
              cntl_system_error = 2
              create_error   = 3
              lifetime_error = 4.
```

Parameters	Description
clsid	ID of the class
lifetime	Lifetime management parameter. The following values are permitted: **`my_control->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 78] method is called. **`my_control->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 90] 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. **`my_control->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 **`my_control->lifetime_imode`**.
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 Picture Control can be displayed (**see also** SAP Container [Page 1]).
autoalign	' ': Control is not automatically aligned 'X': Control is automatically aligned. This uses the maximum available space within a container.

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 90] method, before the control is destroyed at the frontend.

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

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.

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_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.

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

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.

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_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.

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_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.

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

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 91] method.

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.
```

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