SAP Graphics (BC-FES-GRA)

HELP.BCFESGRA

Release 4.6C

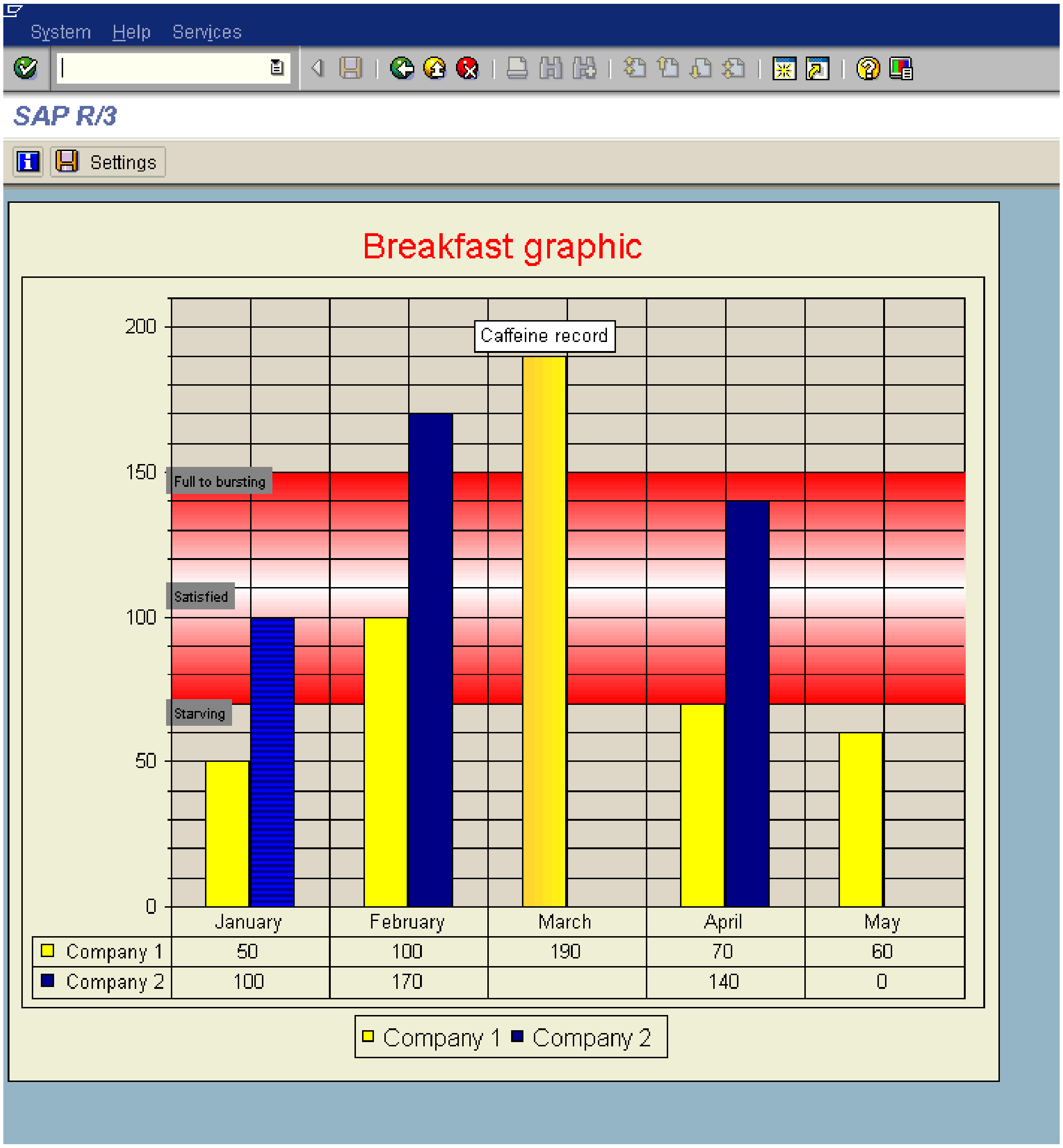
Copyright

Icons

Icon	Meaning
⚠	Caution
🗨	Example
💡	Note
🧭	Recommendation
SYN	Syntax

Contents

SAP Graphics (BC-FES-GRA)



	January	February	March	April	May
🟨 Company 1	50	100	190	70	60
🟦 Company 2	100	170		140	0

This documentation consists of three sections:

The Graphical Framework

The target audience for this section is developers who intend to use the Graphical Framework to program controls. It contains descriptions of the objects that comprise the Graphical Framework and the functionality behind it.

The main aspects covered include the following:

- Data Container (for more information, see Data Container [Page 28])
- Graphics Proxy (for more information, see Graphics Proxy [Page 40])
- Multiplexer (for more information, see Multiplexer [Page 52])
- Customizing Objects (for more information, see Customizing Objects [Page 57])
- Graphical Events (for more information, see Graphical Events [Page 103])
- Product Management (for more information, see Product Management [Page 114])

Business Graphics

The target audience for this section is end users who intend to work with a specific graphics product for business graphics. The documentation is procedure-oriented so that the graphic can be used without having to understand the programming behind it.

The first graphics product described is Chart. The functions available to the user are grouped under the following headings:

- General Settings
- Chart Types (for more information, see Chart Types [Page 145])
- Axes and Gridlines (for more information, see Axes and Gridlines [Page 161])
- Data Series and Data Points (for more information, see Data Series and Data Points [Page 181])
- Titles, Data Labels and Legends (for more information, see Titles, Data Labels and Legends [Page 186])
- Data Tables (for more information, see Data Tables [Page 196])

The second graphics product described is SAP BUSG.

SAP BUSG is available both as an external program (as before Release 4.6A) and also as an ActiveX control. However, if it is used as an ActiveX control some of the menu functions described can no longer be used.

The following information is provided on programming interfaces:

- Data Containers Supported [Page 201]
- Graphics Proxy Supported [Page 202]
- Graphics Products Supported [Page 204]
- Function Modules [Page 207]

Structure Graphics

The target audience for this section is end users who intend to work with a specific graphics product for structure graphics. The documentation is procedure-oriented so that the graphic can be used without having to understand the programming behind it.

SAP Graphics (BC-FES-GRA)

Two types of structure graphic are mentioned:

- SAP Tree [Page 213]
- Gantt Chart [Page 214]

The following information is provided on programming interfaces:

- Data Containers Supported [Page 216]
- Graphics Proxy Supported [Page 217]
- Graphics Products Supported [Page 220]

Links for Related Information

The following graphics types support the drag and drop function:

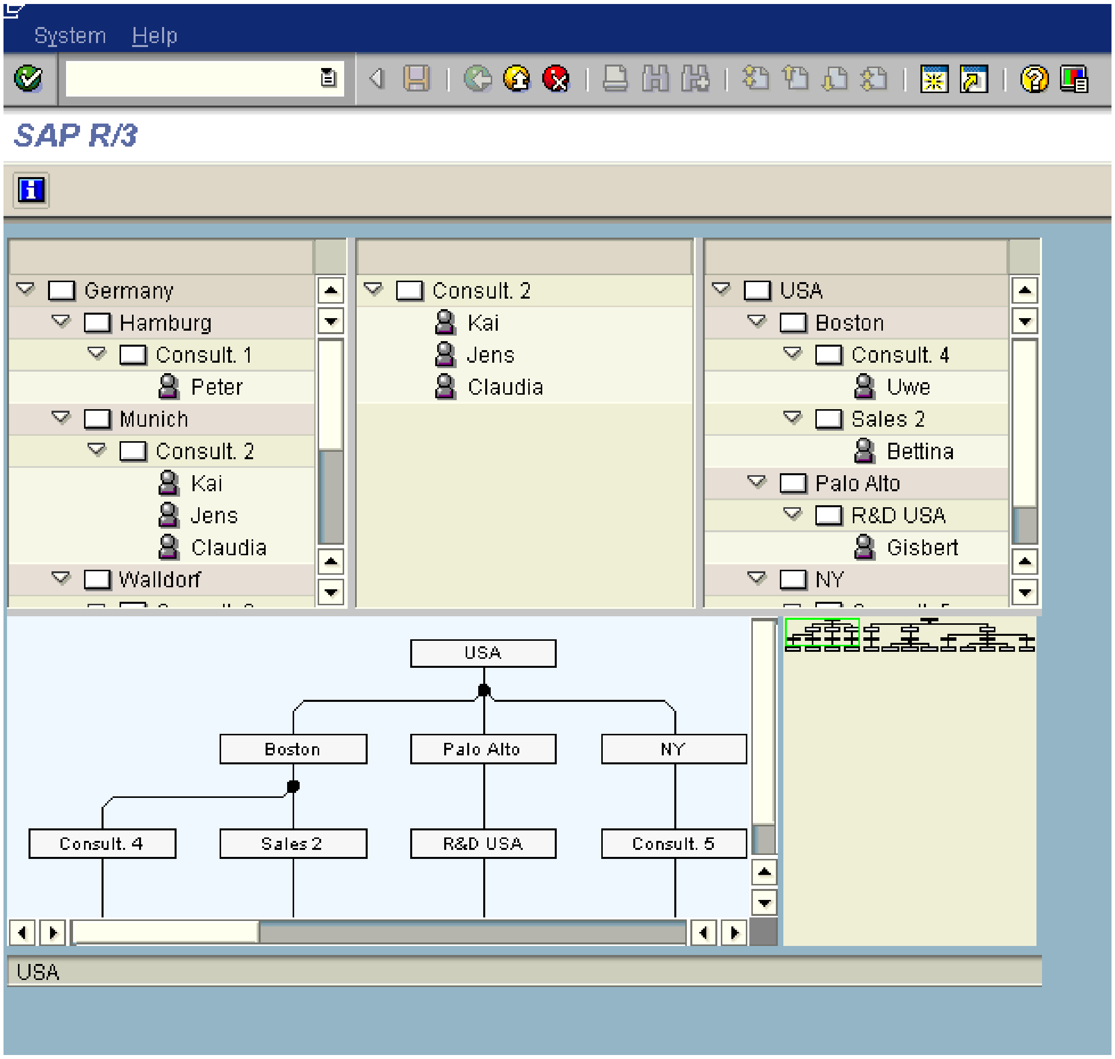
- SAP Tree
- Gantt Chart

For more information, see drag and drop in the documentation on the Control Framework.

Graphical Framework

Use

The Graphical Framework is used to simplify and improve the programming of presentation components in the SAP R/3 System. This involves the programming from the point of view of the application as well as incorporating new graphical objects that have either been developed within a company or supplied externally.



Features

The Graphical Framework has the following key features:

- It uses the Model View Controller design pattern: The data container, which stores the data, is the *model* and the graphics proxies / graphical displays are the *views* of this *model*. It is important to realize that the *views* are separate from the *model*. Several graphics proxies can subscribe to one data container if different graphical displays are required. An alternative model that merged the graphical display with the data storage would have meant that much of the programming would have to be duplicated if a different graphical display were required. The *controller* in the Graphical Framework is the application.

- The graphical objects (controls) are replaceable which means that you are independent of manufacturers and able to use the best the market has to offer.
- The graphical objects are integrated in the SAP front end so that the user can access them as GUI elements as with basic controls such as table controls or tab pages. However, the prerequisite for this control technology is that ActiveX controls or Java Beans are used.
- A flexible (and efficient) type of programming is to be used for application programming (object-oriented programming).
- You can display several views of the same data without having to duplicate the data.
- Every graphical object is configurable for each user - that is, you can configure Customizing for individual users.
- A wide range of frontend platforms is supported. Whatever runs on Windows 95 and NT should also run on Macintosh, OSF/Motif, OS/2 and JAVA VMs. The level of integration may not be as high as on the Windows platforms and the functionality of the graphics may not be as extensive but basic functionality is available.

Glossary

ABAP Event (See Internal Event)

An event within ABAP which contains an internal event with a standard event code and event parameters.

ABAP Objects

Object-oriented ABAP: Extension of the ABAP programming language which implements the concepts of object-oriented programming.

ActiveX Control

A control for the MS Windows platform.

API = Application Programming Interface

A set of routines used by an application program to call a service provided by the operating system (e.g. "open file") or other applications or subroutines.
The API is the interface to the control (data hiding).

Attribute

Attributes and their values describe the state of an object.

Back End

It generally comprises the database server and the application server. In the context of the Graphical Framework it includes the graphics proxy, the multiplexer, the data container, Customizing and the application. The back end is defined in contrast to the front end.

Business Graphic

A graphic type comprising line graphics, bar charts and column charts for example.
These sorts of graphics can also be found in MS EXCEL.

Change Flags

Flags indicating whether data has been changed. The change flag is stored in a separate flag table in the data container.

Class

A class is an abstract description of something in the real world. It describes a general element or a general concept, for example the abstract concepts "business partner" or "material". It contains attributes, methods and events.

Constructor

A method that runs automatically during the creation of an object and sets the initial attribute values. It may contain IMPORTING parameters.

Control

A software component with a public API. They can vary greatly in terms of functional complexity. A control can be an ActiveX control or a Java Bean.

Control Framework

A Basis technology involving ABAP objects. The GFW uses the Control Framework for communication between a graphics proxy and its graphic. The Control Framework is essential for displaying ActiveX controls and Java Beans in R/3. It is not to be confused with the GFW itself.

Create Object

An ABAP Objects command used to create an instance of a particular class.

Customizing Object

A customizing object is a set of bundled attributes. It is an object consisting of semantically-related attributes.

Data Container

A container holding data for the application. The objects it contains have a series of fields containing attributes. The objects are identified by a unique ID; the rest of the fields can be defined according to requirements. The graphics proxy takes the required data from the data container and uses it to display the graphic.

Data Container Wizard

The data container wizard is a report which uses a DDIC structure to generate a file with a data container implementation. This file is incorporated in the application as an Include.

Data Provider

It facilitates data transfer between the back end and the front end. It makes it possible to send the contents of tables to the graphic. Without the data provider data transfer between the back end and the front end would be more restricted and less efficient.

Encapsulation

A principle in object-oriented programming which means that an object's data is hidden from direct access. It can only be accessed from outside using the object's methods. The implementation of the methods are also hidden. Access from outside is only possible using the publicly-defined interface.

Event

An event is a message sent by an object to say that it has changed its state (attributes). It generally contains export parameters but no import parameters. It is like an inverse method and can be defined in a class or an interface. Events are commonly user-driven.

Filters

There is a column in the data container for a filter indicator. This allows several graphics proxies to use the same data container but to access different sets of data. This avoids the need to create separate data containers for each graphics proxy.

Front End

The front end in the context of the GFW comprises the graphics. The front end is defined in contrast to the back end.

Function Module

There is a function module with a function for each graphic type (structure graphics, business graphics) that uses the ABAP objects interface. The advantage is that implementation is simpler. The disadvantage is that it can only be used to display (but not to change) simple data with fixed Customizing.
For more complex functionality the ABAP objects interface must be used directly.

Graphical Event (See Event)

Events/user actions reported by the graphic. They are supplied by the GUI for specific products. An example of a graphical event would be a mouse click.

Graphical Framework

The purpose of the Graphical Framework is to simplify and improve the programming of graphics. The main elements of the Graphical Framework are the following: Data containers, graphics proxies, multiplexer, customizing objects, product management, standardized function modules and the data container wizard.

Graphical Object

Graphical representation of an application object that comprises all of the graphical elements and data required.
Example: An operation is represented by a red bar (as graphical element) from the earliest start

to the earliest end (as necessary dates).

Graphics Proxy

A defined interface and transport layer between the data container and the application which are both in the back end and the graphics in the front end. Each graphic has its own proxy. The graphics proxy is the only part of the ABAP program that deals with communication with the graphic.

Graphic Type

A distinction is made, for example, between business graphics (such as those in MS EXCEL) and structure graphics (such as hierarchy graphics).

Handle

It is a technical variable, a name with which to communicate with the front end about a control during runtime. It is the name by which the front end knows the control (ActiveX or JavaBean).

Hierarchy Graphic

A type of structure graphic comprising those graphics displaying hierarchies. An example would be a graphic in which work centers are displayed in a hierarchical structure.

Implementation

The concrete coding for a method or interface.

Inheritance

Inheritance defines the relationship between classes in which one class (subclass) uses the structure and behavior that has already been defined in another class (superclass).

Instance (See Create Object)

The concrete instantiation/realization of a class as an object. Objects both know things (they have data) and do things (they have functionality).

Interface

An interface defines the interaction between objects. It defines a set of methods and/or constants and events. Other objects can call the methods in the interface to communicate with any object that implements this interface.
An interface is an abstract definition without an implementation. Classes that have such an interface have to implement the methods defined in the interface.

Internal Event (See ABAP Event)

An internal event is the same as an ABAP event.

Java

Object-oriented programming language that is well-suited for use in the Internet because it is platform-independent.

Java Bean

Java's version of a component. Java Beans are more platform-independent. ActiveX controls are based on MS Windows (32bit).

Key Attribute

The attribute that identifies an object in the data container.

Master Graphics Proxy

When graphics are synchronized with respect to an event the master graphics proxy sends the event to one or more slave graphics proxies. This is a way of implementing a change notification.

Method

A function or a procedure that can be executed by a class/object. It contains the "coding" for an object and can be called by other objects.

Method Call

Where one object calls the method belonging to another object to get it to perform a specific function.

Multiplexer

The object responsible for handling program graphics and synchronizing events.

Named Datasets

A specific set of data in the data container. The existence of several datasets means that it is possible to revert to a previous dataset using the UNDO function.

Object

An object is a concrete instance of a class.
Typical examples of objects would be "customer", "order" or "invoice".

Object Oriented Programming

Has three defining elements: Encapsulation, polymorphism and inheritance

OCX

This is more or less a synonym for an ActiveX control. Documentation should, however, only use the latter term because the term "OCX" is obsolete.

PAI

Stands for "processing after input". It refers to the processing block after a screen has been displayed and includes event handling.

PBO

Stands for "processing before output". It refers to the processing block after a screen has been called but before it has been displayed.

Private

The relevant class component (attribute, method, event) cannot be accessed by instances of any other classes.

Product Management

This is a list of graphics products stored in a table with details of their platforms and graphic types. The graphics proxy can automatically choose a product on the current platform.

Program Graphics

This refers to graphics that are external programs. The graphic is displayed in a separate window and is running as a separate process. It was the only SAP graphic before release 4.0.

Protected

The relevant class component (attribute, method, event) can be accessed by instances of classes derived from the class in question (by means of inheritance) but NOT by instances of unrelated classes.
The definition of "protected" differs slightly from language to language.
The Java definition: Both derived classes and classes of the same package may access protected properties and methods.

Public

The relevant class component (attribute, method, event) can be accessed by instances of any class.

Slave Graphics Proxy

When graphics are synchronized with respect to an event the master graphics proxy sends the event to one or more slave graphics proxies.

Splitter

A multi-purpose splitter control that takes the Windows output of the controls and displays them in a grid. The areas can be resized relative to one another.

Structure Graphics

A type of graphic comprising network graphics and hierarchy graphics.

Synchronization

The coordination of two or more graphics with respect to an event. The event is sent from one graphic to another without the application necessarily being informed of this.

Synchronization Job

The application sends the multiplexer synchronization jobs that specify for events which graphics proxy is the master and which the slave(s).

UNDO

The data container keeps an archive of previous datasets and it is possible to revert to the most recently archived dataset or to a named dataset.

Vendor/Software Component Partner

A company that manufactures a control that is used in the Graphical Framework.

Overview of Basic Objects

The Graphical Framework uses the following basic objects:

Data containers

The data container is a container holding data for the application. It contains objects that have an unambiguous ID; the rest of their structure can be defined according to requirements. The graphics proxy takes the required data from the data container and displays it in the graphic. Since several graphics can subscribe to a single data container the data to be represented needs to be stored only in one place. Every change affects all the graphical objects concerned automatically. Data containers implement a specific interface. This means that an application development team can implement its own data container so long as it implements the defined interface (the interface methods must be implemented in the data container). For more information, see Data Container [Page 28].

Graphics proxies

A proxy acts as a defined interface and transport layer between objects - in this case between the data container and the application (in the back end) and the graphic (in the front end). Proxies generally have additional logic and cache functionality. Given the well-defined interface in the back end the application programming is independent of the graphical object that is in the front end.

The graphics proxies need to be adapted to the requirements of the application. Since they are objects they can be implemented in a more detailed form and enhanced with additional logic. For more information, see Graphics Proxy [Page 40].

Multiplexer

The multiplexer relieves the application of the following tasks:

Handling of external program graphics

The multiplexer ensures that several program graphics can be displayed at the same time (one endless loop only) and that their events continue to be processed correctly.

Synchronizing the graphics proxies with respect to specific events

The multiplexer sends an event processed by one graphics proxy to another (for example, to synchronize several graphics proxies on a time axis, to scroll several graphics simultaneously or to synchronize Customizing). It is important to point out that events within the Graphical Framework are product-independent. For more information, see Multiplexer [Page 52].

Customizing objects

The display attributes are bundled into customizing objects so that the user can change the settings of all graphics flexibly (and individually). The appearance of the graphics is determined by assigning these customizing objects to ports in the graphics proxies. The attributes defined in the customizing object can be overridden by user-specific values. For more information, see Customizing Objects [Page 57].

Product management

A graphics proxy can support several graphics products in the front end. There is a part of the Graphical Framework known as product management that specifies the product and the priority to be used on specific operating systems. The priority can be adapted for the company using R/3. For more information, see Product Management [Page 114].

Standardized function modules

There are some simple graphics with a standardized layout and no interactive functionality. To simplify the programming you can use function modules to display such graphics. There is a function module for every graphics type that encapsulates the powerful functionality provided by the objects described above (data container, graphics proxy and so on). For more information, see Function Modules Supported [Page 207].

Data container wizard

The user can implement an application-specific data container using a standard program (GFW_DCWIZARD) belonging to the Graphical Framework. The program generates appropriate ABAP source code for a given DDIC structure. For more information, see Data Container Wizard [Page 39].

Overview of Customizing Classes

Use

You use attributes bundles (such as CL_CU_AXIS, defined in the class builder) to customize graphics in the Graphical Framework. Below is an overview of the most important settings for the various attributes bundles. The list is not exhaustive.

Features

You can make customizing settings for the following:

Diagram (CL_CU_DIAGRAM)

- Used in business graphics
- Appearance of graphic using frame / area / text / miscellaneous attributes

Drawing area (CL_CU_DRAWING_AREA)

- Used in business and structure graphics
- Title for plot area
- Appearance of legend
- Layout for milestone trend analysis

Data series (CL_CU_VALUES)

- Used in business graphics
- Chart type / subtype (for example, column / line / pie)
- Labeling (type, placing, format)
- Standard distribution settings
- Smoothing curves
- Handling of missing values

Axes (CL_CU_AXIS)

- Used in business graphics
- Types of axes (time, linear, logarithmic)
- Specific settings for time axes
- Titles for axes
- Number of visible categories
- Ranges (visibility, values, title)
- Min. / optimum / max. values for axes
- Margins
- Tick marks
- Width of day

- Intersection points for axes

<u>Scale</u> (CL_CU_SCALE)

- Used in business graphics
- Time interval (value, format)
- Step size of scale
- Tick marks

<u>Points</u> (CL_CU_POINT)

- Used in business and structure graphics
- Special customizing for individual points
- Labeling (type, position, alignment)
- Sequence of items (hierarchy graphics)
- Layout of items (hierarchy graphics)
- Point expandable (hierarchy graphics)
- Icons for leaf, expanded \ collapsed (hierarchy graphics)
- Interaction (hierarchy graphics)

<u>Items</u> (CL_CU_ITEM)

- Used in business and structure graphics
- Types of item: Icon, checkbox, text, link, pushbutton
- Column headings
- ToolTips
- Editing of items
- Interaction with items

<u>Grids</u> (CL_CU_GRID)

- Used in business graphics
- Appearance of grid lines

<u>Data sheet</u> (CL_CU_DATA_SHEET)

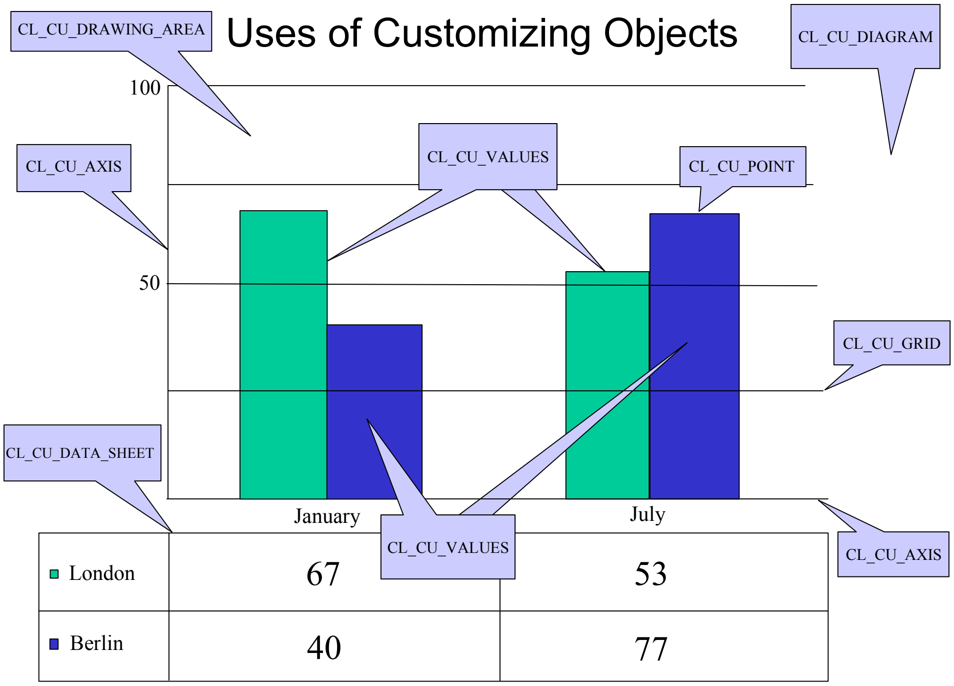
- Used in business graphics
- Data sheet / table
- Visibility of legend symbol within data sheet

<u>Display context</u> (CL_CU_DISPLAY_CONTEXT)

- Used in business and structure graphics
- Visibility of graphical element
- Appearance of background
- Colors

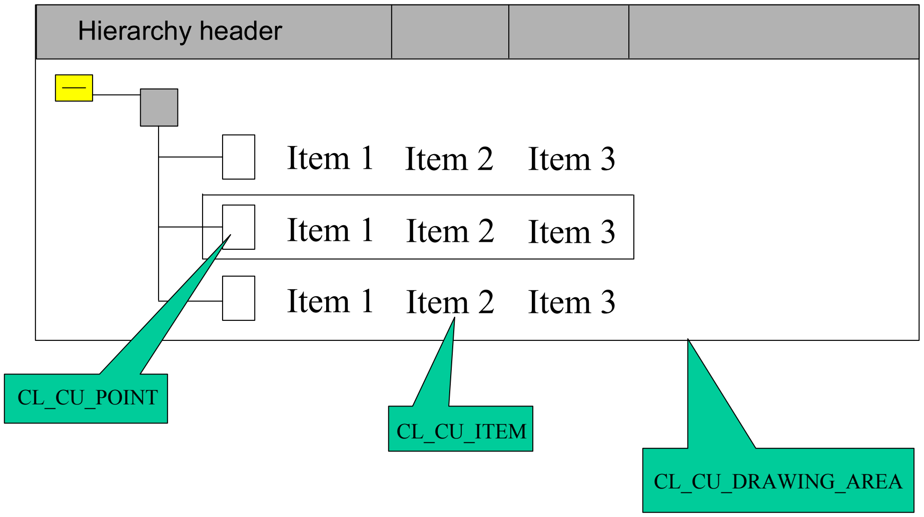
Overview of Customizing Classes

- Lines
- Border lines
- Area
- Type face
- Markers



🟩 London	67	53
🟦 Berlin	40	77

Uses of Customizing Objects



Interaction of Basic Objects

Purpose

The basic objects in the Graphical Framework interact for various purposes including the following:

- One object creating or destroying another (for example, the application creating the data container)
- One object subscribing to another (graphics proxies registering with the data container)
- One object sending data to another (the application filling the data container with data)

Prerequisites

For some types of interaction one object needs to be registered with another. For example, a graphics proxy must be registered with a data container to access data from the latter.

Process Flow

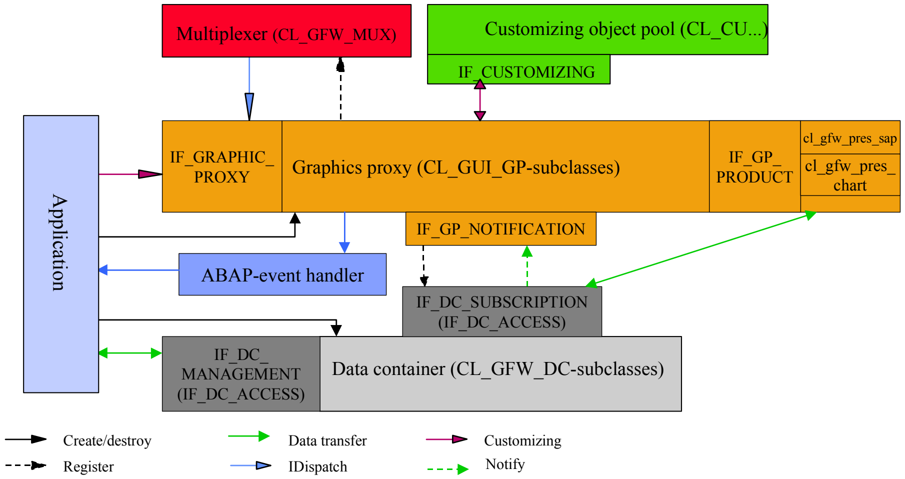
First the application in the figure below creates the data container and then fills it with the data to be displayed. The application also creates the graphics proxies.

Each graphics proxy then subscribes to a data container (for reasons of consistency a graphics proxy can take its data from one data container only). The data container distributes data to the graphics proxies that are registered with it.

Next, customizing objects are assigned to the graphics proxies so that the graphics proxies can change the appearance of graphical objects - the customizing objects define the display attributes.

The graphics proxies then register with the multiplexer which coordinates communication between graphics proxies.

Classes and Interfaces in the Graphical Framework



Changing data

If you change data by interactively changing the graphical object then the application is informed of the change to the data and can query the data container concerning the changes. The advantage of this technology is that the event parameters of the graphical objects need not be known to the application and any change is represented only in the data. If the application validates and accepts the change to the data the data container distributes the changed data to the graphics proxies that are registered with it (using the notify mechanism). The same mechanism is used if the application adds, changes or deletes entries in the data container. This guarantees that all graphical objects always display the same data (without a complex distribution mechanism being necessary on the part of the application).

For more information, see Overview of Basic Objects [Page 20].

Data Container

Definition

The data container is a container holding data for the application. The data consists of objects (rows in the data container) and each object has a series of attributes (columns in the data container).

Use

The data container provides data that can be represented in one or more graphics.

The data to be displayed and its layout generally differ from one application to another. Therefore it is often appropriate to implement an application-specific data container. For this purpose a program called a data container wizard can be used to implement a data container for a given DDIC structure (for more information, see Data Container Wizard [Page 39]). However, there is a standard implementation of the data container for each graphics type that you can use to save time.

You can set values for objects in the data container using *Set* methods. These methods do not send return values. You can access values from the data container using *Get* methods. In this case the return value indicates any ambiguity involved. If an object is not identified uniquely by the field identifier (name of the attribute of a specific data object) the data container returns an error - depending on the method concerned. For more information, see Reading and Changing Values [Page 31].

The data contained in the data container at any one time is known as the dataset. For every dataset there is a flag *Reversible* (Y/N) and an indication as to whether there are changes (locked/released and if applicable the number of changes). You can reverse changes made by reverting to previous datasets. For more information, see Undo and Named Datasets [Page 35].

The local data in the data container contains a list of the graphics proxies subscribed to the data container and a status for the current dataset. The status could indicate, for example, that an event/change from a graphics proxy has been made in the data container (and which proxies) and that the changes have not yet been released/validated.

Structure

A possible structure of a data container is described in detail in Data Container for Business Graphics [Page 36].

Integration

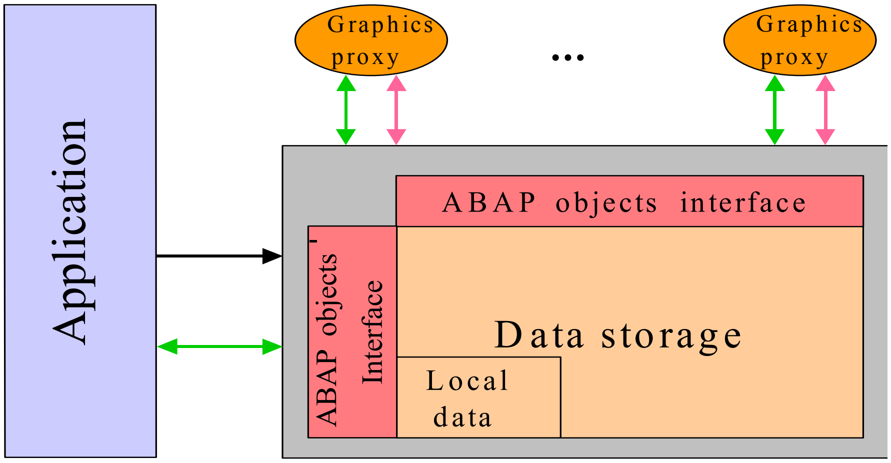
The application generates the data container that serves as the central pool of data for all the graphics proxies subscribed to it. It is defined by its interface IF_DC_MANAGEMENT so any development team can implement it.

The data container has the following global interfaces:

- IF_DC_MANAGEMENT - the interface for the application
- IF_DC_SUBSCRIPTION - the interface for the graphics proxy
- IF_DC_ACCESS - the interface for accessing data

  This final interface is contained in both of the other interfaces - it is a nested interface.

Graphic Showing Integration of the Data Container



The graphic shows the data container, its data storage and local data and its interfaces to other objects within the Graphical Framework. It has interfaces both to the application and to its graphics proxies.

Creating and Destroying the Data Container

Use

The application creates and destroys the data container. Two methods, INIT and FREE, have been defined in the interface for this purpose.

The method INIT replaces the constructor and must be called by the application explicitly. There are two reasons why a constructor is not used: first, so that all data containers are initially called in the same way and second because INIT sends a return value.

The method FREE replaces the destructor but it must be called by the application explicitly.

Integration

For more information, see Data Container [Page 28].

Reading and Changing Values

Use

You can read data from the data container, read changes to values of data and maintain data in the following ways:

Reading/Changing data	Fields/Rows/Tables	Comments
Reading data	Field by field	Using method GET_VALUE
Reading data	Row by row	For specific data containers
Reading changes to values	Field by field	Using method GET_CHANGES
Maintaining data	Field by field	Using method SET_VALUE
Maintaining data	Row by row	Only for some data containers (1)
Maintaining data	Complete tables	Tables must have a fixed layout

(1) The method is not available in the interface; it is in the implementing class.

Features

To set a value the data container checks the data type and if possible converts it according to the usual ABAP conversion rules.

The data container manages several datasets which means that the *Undo* function can be used (see also Undo and Named Datasets [Page 35]). The current dataset is distributed to the graphics proxies that are subscribed to the data container on request - for example after a change has been made.

Activities

To be able to inform all subscribed graphics proxies if data is changed it is necessary to recognize such changes. Any changes the application makes to data are marked automatically as *released changed* by the data container or the dataset is marked as *initial* if no graphics proxy is subscribed to the data container. Data changed by a graphics proxy is set to *changed*.

If data is changed and the dataset has not been distributed the data container overwrites its current data and sets the change status of the fields accordingly. If the dataset has already been distributed the changes to the data are entered initially in a copy of the current state. A copy is created automatically after the first change message for a dataset that has been distributed. All further changes are written in the copy and marked as *locked* (that is, *released changed* or *changed*) until they are released (by the application that has been informed of changes by the graphics proxy).

Deleting an object from the data container (method DEL_OBJECT) is a special case of changing a value. This means that a difference-update of the graphics proxies is also possible where objects are deleted.

A sequence of changes must be completed with a distribution order to the data container (see method DISTRIBUTE_CHANGES in Communication Data Container <-> Graphics Proxy [Page 32]). This is so the data container tells the graphics proxies that are subscribed to it about the changes and sets the change flag to *initial* or deletes the objects.

Communication Data Container <-> Graphics Proxy

Use

All the graphics proxies for which the data container provides data must be subscribed to it. This means that they are informed automatically of changes to data. The data container ensures that values can be changed or requested only by graphics proxies that are subscribed to it.

You do not need to know the details about communication between the data container and the graphics proxy if you only want to use a data container and a graphics proxy. You need to know the details if you want to program your own data container without using the data container wizard or if you want to implement your own graphics proxy.

The data container can tell its graphics proxies to request relevant changes to data from the data container - using the method NOTIFY.

Integration

For more information, see

Data Container

[Page 28]Graphics Proxy [Page 40]

Activities

To subscribe for data in the data container or to cancel a subscription a graphics proxy calls the methods SUBSCRIBE or UNSUBSCRIBE in the data container interface IF_DC_SUBSCRIPTION.

The graphics proxy also calls the method SUBSCRIBE if it wants to change its data selection (using the filter function) - that is, to resubscribe. In this case, if the current dataset is already distributed (that is, there is no *normal* distribution order) the graphics proxy is immediately instructed to read all its data (using the filters that are now new).

The data container uses its distribution methods DISTRIBUTE_CHANGES and RESTORE to call the method NOTIFY in the subscribed graphics proxies. The graphics proxies use this method to find out about the data changes of interest to them - they do this using the value change methods in the data container.

If the current dataset has already been distributed (that is, all the graphics display the current data) then the graphics proxies are not informed of the change.

The graphics proxies call the interface methods of the data container to change values (see Reading and Changing Values [Page 31])

All of the changes must be released at the same time. This ensures that no intermediate states are generated that would be anomalous in the graphic. An event can be reversed only as a whole, so the changes are not separable.

The method NOTIFY is *not* in the data container interface; it is in the graphics proxy interface IF_GP_NOTIFICATION.

Filters

Use

A data container can contain filter attributes so that graphics proxies read only data relevant to them. When the graphics proxy subscribes to the data container it is specified whether it can read all the data in the data container or just the data for objects where certain filters are set.

The application defines data relevant for a graphics proxy by first sending the filter list to the graphics proxy. The data type I (integer) was specified for filters because the filter list must have a data type that can be used by all the graphics proxies.

Example

The application sets the filters when filling the data container:

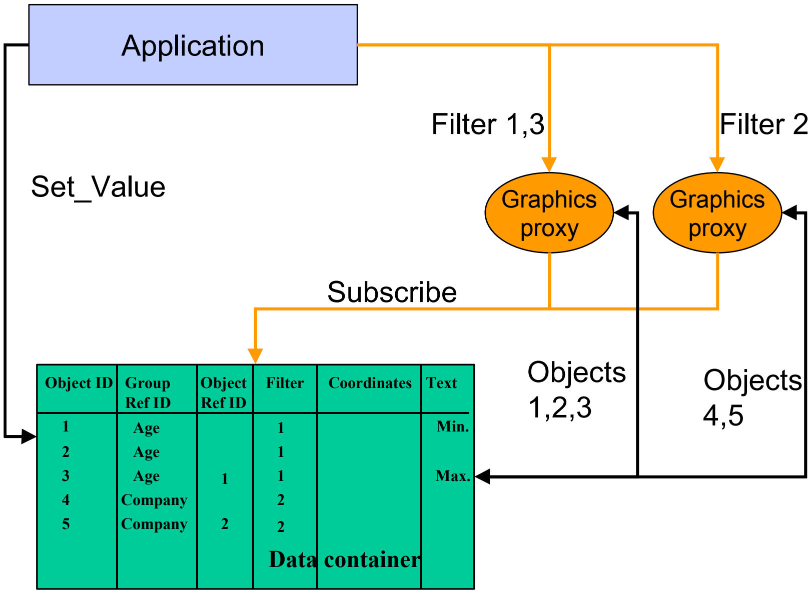
```
OBJ-OBJID = '1'.

OBJ-GRPID = TEXT-G01.

OBJ-Y_VAL = '30'.

OBJ-FILTER = 1.

CALL METHOD DC_INSTANCE -> SET_OBJ_VALUES…
```

The application tells the graphics proxy which filters to use:

```
REFRESH FILTER_LIST.

APPEND 1 TO FILTER_LIST.

CALL METHOD GP -> INIT

          EXPORTING…

                    FILTER_LIST = FILTER_LIST.
```

Integration

Filter Function of Data Container



Object ID	Group Ref ID	Object Ref ID	Filter	Coordinates	Text
1	Age		1		Min.
2	Age		1		
3	Age	1	1		Max.
4	Company		2		
5	Company	2	2		

As shown in the figure above the application sets the filters for the data container.

The application in the figure above specifies filters 1 and 3 for one graphics proxy and filter 2 for the other. The graphics proxies subscribe to the same data container and access different sets of data from it.

Undo and Named Datasets

Use

The data container generates a history for the datasets automatically. Once a distributed dataset has been replaced by another dataset it is entered in the history. A restriction to the history can be specified in the INIT method (default 1). There is also a list of named datasets. A dataset is entered in the list using the method DATASET_STORE in the interface IF_DC_MANAGEMENT. The data container can revert to a named dataset at any time.

Events cannot be split when the data container reverts to a named dataset; just as events cannot be split when datasets are released. The application is responsible for changing application data that is not part of the data in the data container. Note that a dataset in the data container can be marked as *not reversible* (method DISTRIBUTE_CHANGES).

You can delete a named dataset from the list using the method DATASET_DELETE in the interface IF_DC_MANAGEMENT.

You can use the method DATASET_RESTORE in the interface IF_DC_MANAGEMENT to revert to the last history dataset or to a named dataset. In the first case the current dataset is deleted and in the second case it is entered in the history list (if it has been distributed, otherwise it is deleted). The dataset to which the data container reverts becomes the current dataset. If the data container reverts to a named dataset the current dataset becomes a copy of the named dataset.

Integration

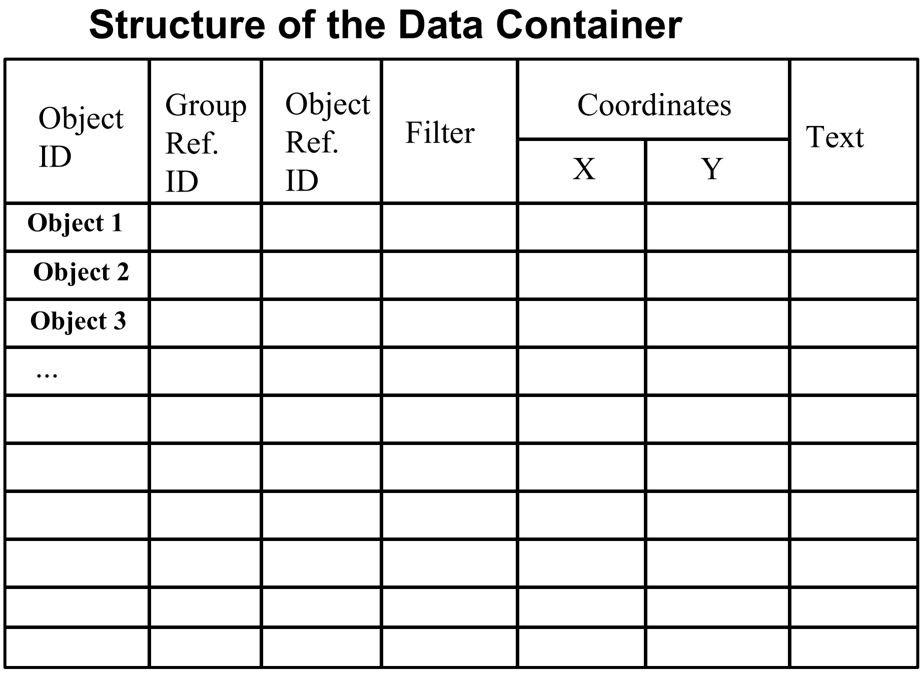
If the old dataset had already been distributed or if the data container reverted to a named dataset all the subscribed proxies are informed of the change (*reverted dataset*). The data container tells the proxies to fetch the new (=all) data (distribution) and then marks the dataset as *distributed*.

In the case of a history reversal if the old dataset had not been distributed then all the subscribed proxies are based on the dataset that is now current again. The change information (*reverted*) need not be distributed. Exception: If changes were made by a proxy then it will already have changed its data to the new state. To change it back to its former state the data provider marks the data relevant for the special proxy as changed and tells just this proxy to fetch the current data. In this way the dataset is again distributed and the markings are removed again from the data container.

Data Containers for Business Graphics

Example of a data container

A definition of a data container for business graphics is provided below. Business graphics can obtain their data from various data containers because the concept of a data container interface is very flexible and can be realized in many ways. For information on the programming interface for data containers, see Data Containers Supported [Page 201].



Structure of the Data Container

Object ID	Group Ref. ID	Object Ref. ID	Filter	Coordinates		Text
				X	Y	
Object 1						
Object 2						
Object 3						
...						

The business graphics data container provided by SAP is based on the data structure GFWDCPRES.

The first column is reserved for the object ID because it must be possible to access every object to be displayed in a graphic.

The second column for the group reference ID splits up the objects into groups that can be assigned their own customizing object and so have their own layout. In the case of a business graphic these could be curves or segments of a pie chart for example.

To highlight objects within a group an object reference ID is also specified (customizing object-specific). For example, you could specify the minimum value, the median value or the maximum value for a curve. In the graphics proxy these points are then assigned their own customizing objects and as a result have different display attributes.

The next column can contain a filter indicator to be evaluated by a graphics proxy. This filter means that the data is available for several graphics proxies at the same time and yet they are distinguishable. In this way several graphics proxies (with different quantities of data) can be subscribed to the data container without having to create a separate data container for each graphics proxy. The data type i (integer) was specified because the filter must have a data type that can be used by all the graphics proxies. For more information, see Filter [Page 33].

The next three columns contain the coordinates of the objects. The final column contains a descriptive data field for every column. For example, you can use this to enter an additional text for each point on a curve.

Global Data Container

Definition

The global class CL_GFW_DC_PRES defined in the class builder represents a data container for business graphics.

Use

You can create instances of the global data container to store the data for your business graphics application. This saves you the trouble of creating your own data container.

There are two alternative methods for filling the data container:

- SET_OBJ_TAB to transfer a whole table of data to the data container

  You should use this method when a lot of the data is new or has been changed, for example when initially filling the data container.

- SET_OBJ_VALUES to fill an individual line in the data container

  You should use this method when only a small amount of the data has been changed.

Structure

The global data container is based on the structure GFWDCPRES.

Integration

Data Container Wizard

Definition

To make the Graphical Framework more user-friendly a wizard for creating a data container has been provided. The data container wizard is a report used with a given DDIC structure to generate a file with a data container implementation.

Use

The report GFW_DCWIZARD uses a DDIC structure to create a data container that can be used by the Graphical Framework.

The data container is derived from the global data container class CL_GFW_DC.

Since, for security reasons, Include files cannot be generated automatically in the SAP R/3 System the file is created in the file system. This means that it has to be imported later (by upload) into a manually-created Include.

If a manually-created data container is out of date it can be regenerated using the most recent version of the wizard.

💡

The wizard generates data containers without the UNDO function and without named datasets.

💡

You retain the option of programming your own data container.

To ensure that corrections and improvements to the data container take effect in all applications a (wizard-) version control is integrated in the data containers created by the wizard.

For more information on the data container wizard read the documentation for the report GFW_DCWIZARD.

Graphics Proxy

Definition

A graphics proxy is an object representing a frontend component. It defines and implements the programming interface for application development so that the latter can program the graphical objects (encapsulation). The graphics proxy determines what the graphic looks like and deals with interaction with the front end. The data to be displayed is taken from the data container.

Use

A graphics proxy is used by the application that wants to display a graphic. The application has to interact with the graphics proxy in several ways:

Creating and Destroying the Graphics Proxy [Page 42])

Initializing the Graphics Proxy [Page 43])

Setting Parameters [Page 45])

Making customizing settings - that is - assigning customizing objects to the graphics proxy (see Customizing Settings [Page 47])

Activating/Deactivating the Graphics Proxy [Page 48])

Processing events

All graphics proxies support the global interface IF_GRAPHIC_PROXY. SAP provides a graphics proxy for each graphic type. You can find some special hints on these graphics proxies (such as the class name) in sections on the programming interfaces for the relevant graphic type (for example, business graphics).

A graphics proxy requires a data container to provide the data. For more information on other basic objects in the Graphical Framework, see Interaction of Basic Objects [Page 26].

See also:

Notifying Changes to Data or Customizing Objects [Page 49]

Graphic Type-Specific Method Calls [Page 50]

Structure

A graphics proxy has an ABAP Objects interface which the application uses to access the graphic. The proxy can also collect method calls in the form of a *flush* and can also contain parts of the application logic that are relevant to graphics. The graphics proxy controls the front end independently of how the graphic is implemented in the front end.

Integration

Once the application has generated the graphics proxy the latter subscribes to the data container. The proxy subscribes to selected columns and, if necessary, with a data filter. After this, changes to data are sent from the data container to the graphics proxy automatically. The graphics proxy must also subscribe to the multiplexer to synchronize events.

In keeping with object-oriented programming the graphics proxy manages local object data as instance variables (member variables).

There are two global interfaces for the graphics proxy: one for the application and the multiplexer (IF_GRAPHIC_PROXY) and one for the notify mechanism for the data container (IF_GP_NOTIFICATION).

Creating and Destroying the Graphics Proxy

Use

The application creates and destroys the graphics proxy using the two methods INIT and FREE that are defined in the interface IF_GRAPHIC_PROXY.

The method INIT replaces the constructor and must be called by the application explicitly after the object has been created (with the command CREATE OBJECT). For more information, see Initializing the Graphics Proxy [Page 43].

The method FREE replaces the destructor and must be called by the application explicitly.

Integration

In addition to the interface IF_GRAPHIC_PROXY a structure graphics proxy also implements the interface IF_GP_HIER. This is a structure-specific interface for the structure proxy. For more information, see the interface documentation in the class builder.

For more information, see Graphics Proxy [Page 40].

Prerequisites

You need to have a data container and a container object (to place the graphic on the screen) for initialization.

Initializing the Graphics Proxy

Use

The application calls the method INIT to initialize the graphics proxy.

The method INIT replaces the constructor and must be called by the application explicitly after the object has been created (CREATE OBJECT). Some of the parameters are optional and can be specified later (for example, EVTCODE_LIST).

Example of an optional parameter

A data container can provide data for several graphics proxies. It may have a filter attribute for this purpose. The application defines the parameters for the graphics proxy if such a data container is used by means of the list of filters whose data the graphic is to display. This list of filters is copied when the graphics proxy is initialized or by using the method SET_FILTER_LIST. For more information on filters, see Filter [Page 33].

You can find more information in the documentation for INIT in the interface IF_GRAPHIC_PROXY in the class builder.

For more information, see Graphics Proxy [Page 40].

Prerequisites

All the parameters for INIT (apart from RETVAL) are supplied by the application. All input parameters are optional apart from the following:

DC

This parameter refers to the data container that provides the data to be represented in the graphic.

PARENT

This parameter is important for specifying where the control graphic is to be placed. It identifies the container object in which the graphic is to be placed. This parameter is ignored for graphics displayed as an external program.

Although the parameter PROD_ID is optional it is very important and is therefore explained in detail below.

Features

A graphics proxy encapsulates several graphics products with different functionality. The graphics products supported are specified as constants in the definition of the graphics proxy class (for more information, see Graphics Proxy Supported [Page 202]).

When creating the graphics proxy the application can also specify the product to be used (product ID or priority). The application can specify a particular product or leave the selection of the product to the graphics proxy´s default strategy.

Product management can request a product with a specific priority. Graphics proxies make use of this option to implement a default strategy if the desired product is not available on the platform.

If a default strategy is permitted the graphics proxy requests an appropriate product (given graphics proxy, platform and specified priority) from product management.

In such a case it is not guaranteed that the product offers the full functionality required by the application.

For more information, see Product Management [Page 114].

The parameter PROD_ID specifies the ID of the graphics product to be used. You must note the graphics products supported by the graphics proxy and use the constants defined in the graphics proxy class.

If the specified product is mandatory and there are problems with creation (product not supported by the graphics proxy or product not available on the current platform and so on) the method returns an error message. The object reference for the graphics proxy should be rejected by the application.

Setting Parameters

Use

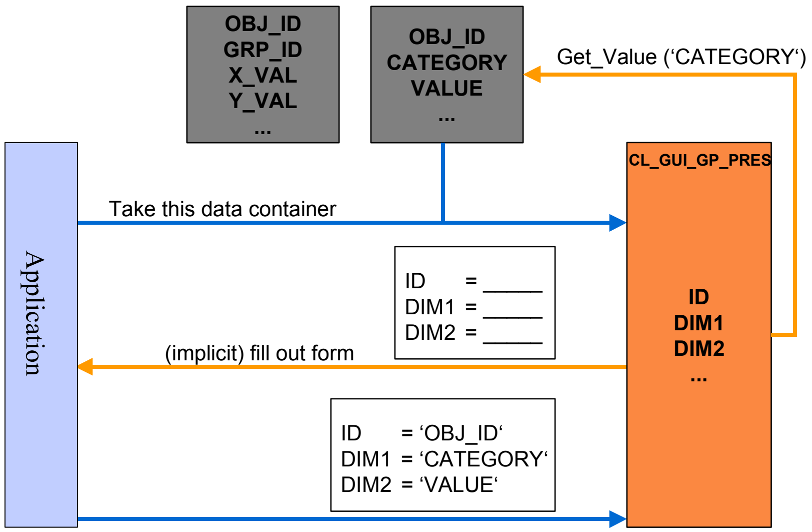
A graphics proxy may have some parameters that the application can set using the method SET_PARAMETER.

The most important are parameters for the reading rule for the data container. These reading parameters are described below.

Integration

The graphics proxy can access data from any data container. The application must tell the proxy what the attributes in the data container of interest to it are called (read rule). The names are parameters of the graphics proxy (CO_PARAM_DC_xxx) that can be set using the method SET_PARAMETER.

Telling GP about Names in DC



ID = _____
DIM1 = _____
DIM2 = _____

ID = 'OBJ_ID'
DIM1 = 'CATEGORY'
DIM2 = 'VALUE'

The graphic shows how the graphics proxy is informed of the names of attributes in the data container.

1. First, the application tells the graphics proxy to use a specific data container (*Take this data container*).
2. The graphics proxy then implicitly fills out a form to obtain the names of the data container attributes (*fill out form*).
3. The application then tells the graphics proxy the names of the attributes in the data container.

The list of parameters (defined as constants in the interface IF_GRAPHIC_PROXY) can be extended if necessary (for example, for types of graphics proxies other than business graphics).

Setting Parameters

A graphics proxy should offer its own method SET_DC_NAMES for the read rule for the data container that indicates immediately the names the graphics proxy requires and which it can process. SET_DC_NAMES is not an interface method. If it is supported it is described in the documentation for the graphics proxy concerned.

Customizing Settings

Use

You use Customizing Objects [Page 57] and the ports [Page 70] that are provided for them to make customizing settings.

A graphics proxy contains several ports for customizing objects (= instances of attribute bundles). Constants (CO_PORT_ ....) are defined for all the ports in the interface IF_GRAPHIC_PROXY.

Depending on the graphic type and the implementation a graphics proxy offers a selection of ports and specifies the customizing bundles that can be assigned to the ports, how often and with which parameters.

Customizing objects can be assigned to and deleted from ports using the interface methods ADD_CU_BUNDLE and DEL_CU_BUNDLE respectively. You do not have to make customizing settings. See the *Features* section below.

Example

Assigning the drawing area to the port 'Chart' of the graphics proxy

```
CALL METHOD GP_INSTANCE -> IF_GRAPHIC_PROXY~ADD_CU_BUNDLE
                EXPORTING PORT = IF_GRAPHIC_PROXY~ADD_CU_BUNDLE
                                 BUNDLE = BUNDLE_DRAWING_AREA
```

Features

If no customizing objects are assigned to a graphics proxy or if some of the possible ports are not occupied the graphic is generated using the appropriate default customizing objects.

If you make changes interactively in the graphic that relate to Customizing settings of the graphics proxy concerned the customizing objects are updated. The multiplexer in the Graphical Framework ensures that all the graphics attached to a customizing object that has been changed are informed of the change. For more information, see Multiplexer [Page 52].

For general information and graphics on which customizing classes are used for which purposes, see Overview of Customizing Classes [Page 22].

Activating/Deactivating the Graphics Proxy

Purpose

The graphics proxy has to be activated to display the graphic.

It can be deactivated to make the graphic invisible without destroying the graphics proxy.

Prerequisites

The application must have created the graphics proxy [Page 40] and the parameters must have been set.

Process Flow

Activating the graphics proxy

The application calls the method ACTIVATE in IF_GRAPHIC_PROXY. The graphics proxy subscribes to the multiplexer and to the data container if it has not already done so. After subscription it reads the data from the data container, checks customizing, prepares this information for the graphics product at the front end and sends the information to the graphic.

Deactivating the graphics proxy

If a graphic is to be made invisible without destroying the graphics proxy object then the application calls the method DEACTIVATE in IF_GRAPHIC_PROXY. The graphics proxy cancels its subscription with the multiplexer - that is, it reduces the search quantity of the multiplexer (performance).

Result

When the graphics proxy is activated the graphic is displayed.

When the graphics proxy is deactivated the graphic is invisible.

Notifying Changes to Data or Customizing Objects

Purpose

The data container informs graphics proxies of changes to data and/or customizing objects automatically so that they can update their graphics accordingly.

Process Flow

The data container calls the graphics proxy's method NOTIFY in IF_GP_NOTIFICATION. It can give one of three reasons for the message:

- Data was changed in the data container. Changed data should be read from the data container and the graphic updated.
- Data has changed in the data container. All the data should be read from the data container and the graphic updated.
- The data container cancels the registration - that is, it does not send any more data. This message is possible/necessary if the data container is destroyed before all the graphics proxies have cancelled their registration.
- Data was not changed but customizing objects were changed (see documentation on the method NOTIFY in the class builder).

For more information, see Reading and Changing Values [Page 31].

Result

The graphics proxies are now aware of changes to data in the data container or that the registration has been cancelled.

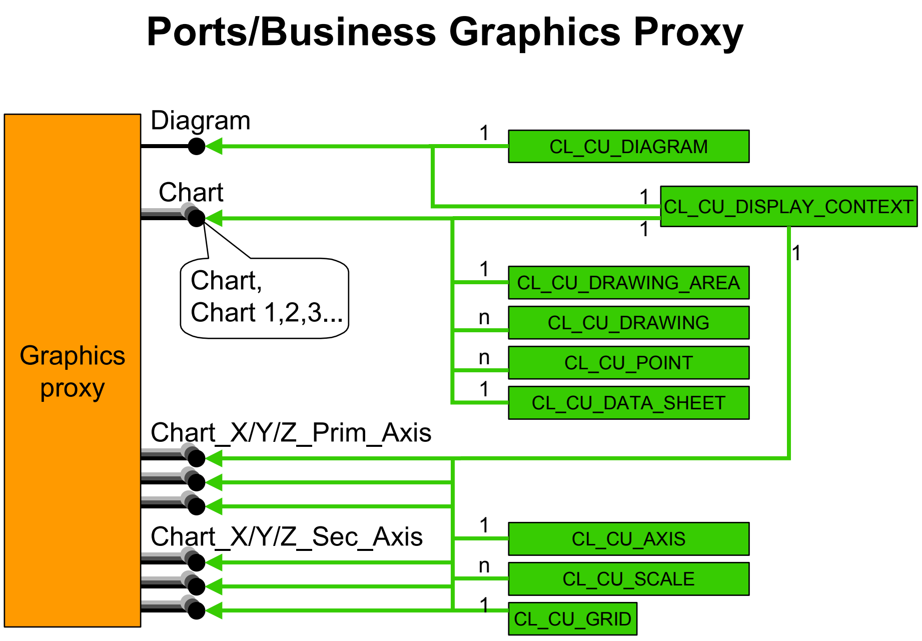
Graphic Type-Specific Method Calls

Use

The graphics proxy classes can implement other methods in addition to the general interface methods; for example, methods for node handling (ADD_NODE, SET/GET_PARENT and so on) may be useful for hierarchy graphics. If a graphics proxy provides such methods they are documented in the graphics proxy-specific documentation.

The interface for graphics proxies is general and can be used independently of the required graphic type (business graphic, hierarchy graphic and so on). The graphic type is determined by the selection of the graphics proxy class that implements the interface. An appropriate interface should be defined for every graphic type (for example, IF_GP_HIER for hierarchy graphics) so that different implementations can use the same method names.

Integration



The black dots to the right of the graphics proxy are the ports. The customizing objects on the right *CL_CU_DIAGRAM*, *CL_CU_DISPLAY_CONTEXT*, *CL_CU_DRAWING_AREA* and so on are assigned to the ports.

Prefixes and Suffixes of Attributes

To find out the exact function of an attribute, see the documentation on that attribute in the class builder. However, the attribute name, including the prefixes and suffixes, should serve as a guide to the attribute's function.

Table indicating the significance of the prefixes and suffixes of attributes

Attribute Prefix / Suffix	Meaning of Prefix / Suffix	Example of Attribute	Function of Attribute
CO_EVTPARAM	Event parameter	IF_GRAPHIC_PROXY~CO_EVTPARAM_AXIS	Axis, for example primary X axis
CO_EVT_	Event code	IF_GRAPHIC_PROXY~CO_EVT_EXPANDED	Event code for event GRAPHIC_OBJ_EXPANDED
CO_PARAM_	Attribute containing name of column in data container	IF_GRAPHIC_PROXY~ CO_PARAM_DC_ID	Data container object ID
CO_…_CONTEXT	Display context as part of a customizing object	CL_CU_AXIS=> CO_TITLE_CONTEXT	Title context of an axis
GRAPHIC_	Event	IF_GRAPHIC_PROXY~ GRAPHIC_CLOSED	Standard ABAP event sent when GUI event CO_EVT_CLOSED received
CO_PORT_	Name of port	IF_GRAPHIC_PROXY~ CO_PORT_CHART1	Port for plot area 1
CO_PROD_	Name of graphics product	CL_GUI_GP_PRES~ CO_PROD_CHART	Intellicube chart graphic

Multiplexer

Definition

The multiplexer is responsible for handling program graphics and synchronizing graphics proxies with respect to certain graphical events.

The multiplexer as handler of program graphics

Graphics started as an external program to the R/3 GUI (program graphics) are programmed differently than other graphics. The graphics proxies encapsulate most of the functionality involved in programming the graphics (formatting the data and so on) in their product-specific section.

Implementing endless loops necessary for program graphics and sending events are independent of the Control Framework but must be carried out by an instance. The multiplexer in the Graphical Framework carries out these tasks.

The multiplexer as synchronization instance

The multiplexer sends selected events from one graphical object to another automatically. For more information, see Synchronizing Events [Page 53].

Use

To ensure that program graphics are handled properly the application has to call the method ACTIVATE in CL_GFW_MUX at the end of PBO. If you do not want to use program graphics at any time make sure that you enforce the required graphics product when the graphics proxy is initialized. To do this set the parameter FORCE_PROD to GFW_TRUE. Unless you do this you will have to call the method ACTIVATE at the end of PBO.

Activating the multiplexer

```
CALL METHOD CL_GFW_MUX=>ACTIVATE IMPORTING RETVAL = RETVAL.

IF RETVAL <> CL_GFW=>OK.

CALL METHOD CL_GFW=>SHOW_MSG EXPORTING MSGNO = RETVAL.

ENDIF.
```

If you want the multiplexer to synchronize graphics with respect to certain events you have to give the multiplexer a list of synchronization jobs. For more information, see Multiplexer as Synchronization Instance [Page 53].

Structure

The definition of the individual methods in the multiplexer can be found in the class definition (CL_GFW_MUX) in the class builder.

There can be only one multiplexer during the runtime of a program or else graphics registered with another multiplexer might not be dealt with. Therefore there is only one implementation of the multiplexer in the R/3 System. The multiplexer has no interface, only a class definition. SAP provides the implementation of the global class.

Synchronizing Events

Purpose

The multiplexer makes it possible for events to be sent automatically from one graphic to another without involving the application.

Example

A case of synchronization is when scrolling two graphics that have the same axis.

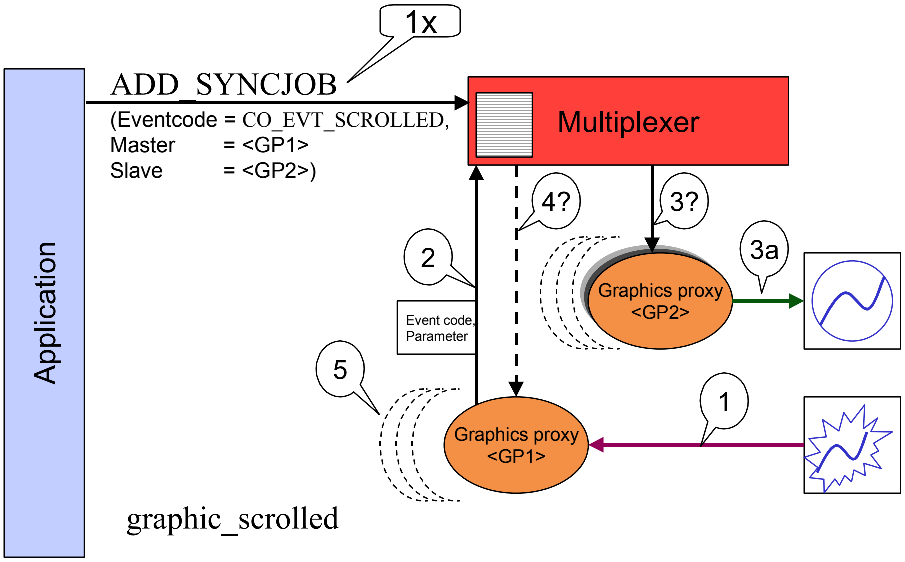
Prerequisites

The application must ensure that the master graphics proxy (in the back end) receives the events to be synchronized from the front end. This means that the application must give the graphics proxy the event for registration even if the application itself is not interested in this event (that is, the application does not implement an event handler).

Process Flow

The application sends synchronization jobs to the multiplexer specifying for which event a certain graphics proxy is the event-sending (master) proxy and which graphics proxies are the receiving (slave) proxies. Every graphics proxy sends each event it receives to the multiplexer for synchronization. If there is no synchronization job the event is sent to the graphics proxy which sends it to the application. If there is a synchronization job then the event is automatically sent to the relevant slave graphics proxies.

Synchronization of Events/Process



1x: First the application sends the synchronization job to the multiplexer.

1: The graphic is changed (for example, the user scrolls the axis) and the graphic sends a GUI event to its GP, the master proxy.

2: The standardized event, along with its parameters, is sent to the multiplexer. An example of a parameter would be the new starting point after scrolling.

3: The GP <GP2> is informed of the change. It is known as the slave GP. There can be several of them.

3a: This is a method call sent to the graphic of <GP2> to synchronize this graphic with the first one.

4+5: In the synchronization job it is specified whether the application is to be informed of the event and if so whether before or after synchronization. If the application is to be informed of the event GRAPHIC_SCROLLED then <GP1> carries this out.

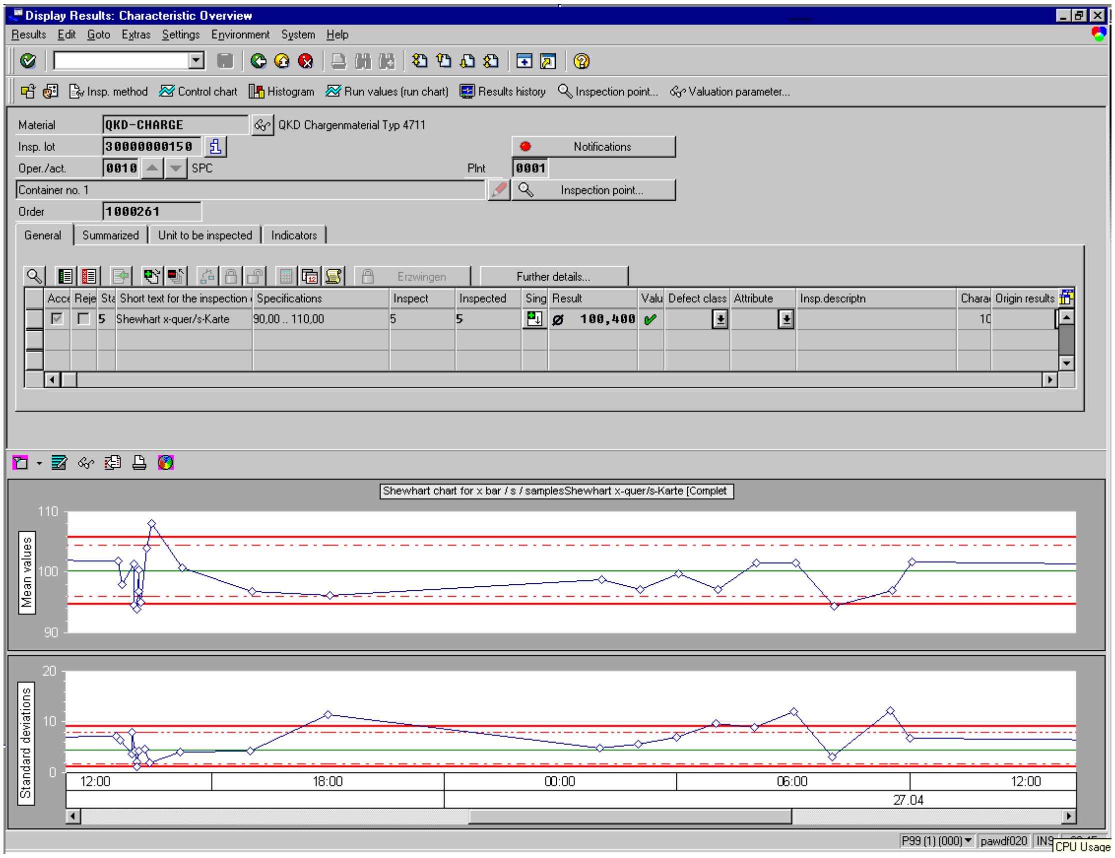
For an example, see Example: Synchronizing Events [Page 55].

Result

All the graphics proxies specified as slave proxies in the synchronization job receive the same event as the graphic proxy that created/raised the event.

Example: Synchronizing Events

Graphic showing a two-track control chart



Material	QKD-CHARGE	QKD Chargenmaterial Typ 4711
Insp. lot	30000000150	
Oper./act.	0010	SPC
Plnt	0001	
Container no. 1		
Order	1000261	

Acc	Reje	Sta	Short text for the inspection	Specifications	Inspect	Inspected	Sing	Result	Valu	Defect class	Attribute	Insp.descriptn	Chara	Origin results
☑	□	5	Shewhart x-quer/s-Karte	90,00 .. 110,00	5	5		Ø 100,400	✓				10	

The figure provides an illustration of the synchronization of events. There are two graphics but only one scroll bar (beneath the lower graphic). When the time scale is changed by scrolling (this is the event) the display in both graphics is synchronized.

Customizing

Use

In addition to the data from the data container the graphic can use information on display attributes such as colors and fonts to display the graphic. These visualization attributes are stored in customizing objects.

Customizing settings can be saved on the database where there are tables for general settings and for user-specific settings. For more information, see Customizing Objects in Database [Page 76]. They can also be transported to other systems.

The display attributes are already bundled in customizing objects [Page 57] which are instances of attributes bundles [Page 58] in the class builder.

Integration

The customizing objects have to be assigned to ports [Page 70] in the graphics proxy.

Features

General display attributes are grouped in the attribute bundle known as the display context [Page 59], CL_CU_DISPLAY_CONTEXT. This attribute bundle can be used as an attribute in other attribute bundles, for example in CL_CU_DRAWING_AREA.

Activities

You can change Customizing settings either interactively in the graphic or by using the ABAP Objects interface. For more information, see Changing Appearance of Graphic [Page 86].

Customizing Objects

Definition

Customizing objects are instances of attribute bundle classes. Attribute bundles are global classes that encapsulate semantically-related attributes (=properties) of a graphic. They start with the prefix CL_CU_.

In addition to the data to be represented the graphic requires other information before it can be displayed: the display attributes (colors, line styles and so on). This information is stored in the customizing objects and can easily be made user-specific.

For information on the types of attribute bundle supported, see Attribute Bundles [Page 58].

Use

An application generates a customizing object by instantiating an attribute bundle class and specifying a name under which the desired settings are (to be) stored on the database. For more information on the instance name, see Creating Customizing Bundles [Page 62].

Customizing objects provide methods so that they can be saved persistently on the database; programs can change individual attributes.

An object-oriented conception was used for Customizing so that fixed settings can be read from the database and values can be changed online or in the application program.

Existing settings are loaded for the name when the customizing object's method IF_CUSTOMIZING~LOAD is called after generation. The application can then adjust/change individual attributes to its own requirements and then assign the customizing object to one or more graphics proxies.

One advantage of bundling attributes that belong together is that you can set attribute values more efficiently; the attributes can be sent to the front end more quickly.

The graphics proxies have graphics-specific ports such as *primary X axis* (see the documentation on the graphics proxy).

For more information, see

Customizing: From Database to Graphic [Page 77]

Customizing: From Graphic to Database [Page 81]

Integration

The graphics proxy automatically converts the contents of customizing objects into product-specific settings. If a graphics product does not support a setting it attempts an approximation. If necessary, the setting for this product is ignored.

Attribute Bundles

Definition

The following attribute bundles (global classes) are available:

- **CL_CU_DISPLAY_CONTEXT**: This contains all of the visual display information such as the colors, the patterns and the thickness of the lines. The settings for this bundle can be *inherited* using the ports. For more information, see Hierarchy of Ports [Page 75].
- **CL_CU_DIAGRAM:** This contains all the global information on the graphic as a whole. It can contain one or more plot areas. In particular the diagram area contains information on the category and the layout of a graphic, for example the number of plot areas in which data can be displayed.
- **CL_CU_DRAWING_AREA:** This contains all the attributes for a plot area; for example, information concerning the visual appearance of the plot areas within the chart. Note that it also contains information concerning the graphic's title and legend.
- **CL_CU_AXIS:** This specifies the value range within which graphical objects are to be displayed. Instead of specifying the value range explicitly it can be determined automatically.
- **CL_CU_SCALE:** This contains scale attributes. A time axis alone does not have any display attributes, only orientation and value range. It is the scale that makes it possible to display an axis with different increments (for example, minutes, hours, days). Other information includes frequency and thickness of separators.
- **CL_CU_VALUES:** This is only used in business graphics and contains the display attributes (for example, its chart type, that is line / bar…).
- **CL_CU_DATA_SHEET:** This bundle specifies the appearance of a table containing all the data in text form.
- **CL_CU_POINT:** The settings for a data series (CL_CU_VALUES) can be overridden in favor of settings for specific data points. Business graphics involve only point categories; with hierarchies and networks you can specify both the structure of a node and the elements that belong to it, which are called items.
- **CL_CU_ITEM:** This contains customizing for a part (an item) of a node. A point or node can consist of several parts, for example texts and icons. An item customizing object can be assigned to a graphics proxy only as a (numbered) sub-object of a point customizing object.

For more general information and graphics showing which attribute bundles are used where, see Overview of Customizing Classes [Page 22].

Use

All attribute bundles can be used to create customizing objects (for more information, see Customizing Objects [Page 57]). However, an attribute bundle might not be valid for a specific graphics proxy. For example, a structure graphic does not need axis bundles. You can find relevant information in the documentation for the graphics proxy you want to use.

You can use the method SET in IF_CUSTOMIZING to change the attributes of a customizing object. Each attribute bundle has constants (static attributes) for all the values that you can set (the names of the constants start with CO_).

Display Context

Definition

The display context is a class containing general visualization attributes.

Use

As with other attribute bundles the application can create an instance of the class, a customizing bundle. The application can assign this customizing bundle to a port in the graphics proxy.

You can use the display context in one of two ways:

- You can use the display context as a substructure in other customizing objects, for example in the drawing area. The display context counts as *one* attribute in another customizing bundle. Since the display context can be used in different customizing objects not all of the attributes are relevant. The other attributes bundles contain contexts, such as the title context and these contexts contain a subset of the attributes of the display context.
- The display context can also be assigned to a port on its own. This is an efficient way to use the display context if the same attributes are used in several different customizing bundles.

The display context can be used for different purposes with different classes. For example, it could be used to set the color of the title for the drawing area and the line attributes for gridlines.

Example

The attribute bundle CL_CU_DRAWING_AREA has the contexts CO_DISPLAY_CONTEXT, CO_TITLE_CONTEXT and CO_LEGEND_CONTEXT each of which contains a subset of the attributes in CL_CU_DISPLAY_CONTEXT.

Structure

The display context contains six types of attribute:

Border

Line

Area

Text

Marker

Miscellaneous

Integration

Those attributes (in customizing bundles) with the suffix *_CONTEXT* refer to the display context.

The application does not have to save the display context separately because it is saved along with the customizing bundle that contains it. It is also loaded along with the customizing bundle that contains it.

The display context is copied, not referenced, so changing an attribute in CL_CU_DISPLAY_CONTEXT does not automatically change the customizing objects that have already been instantiated.

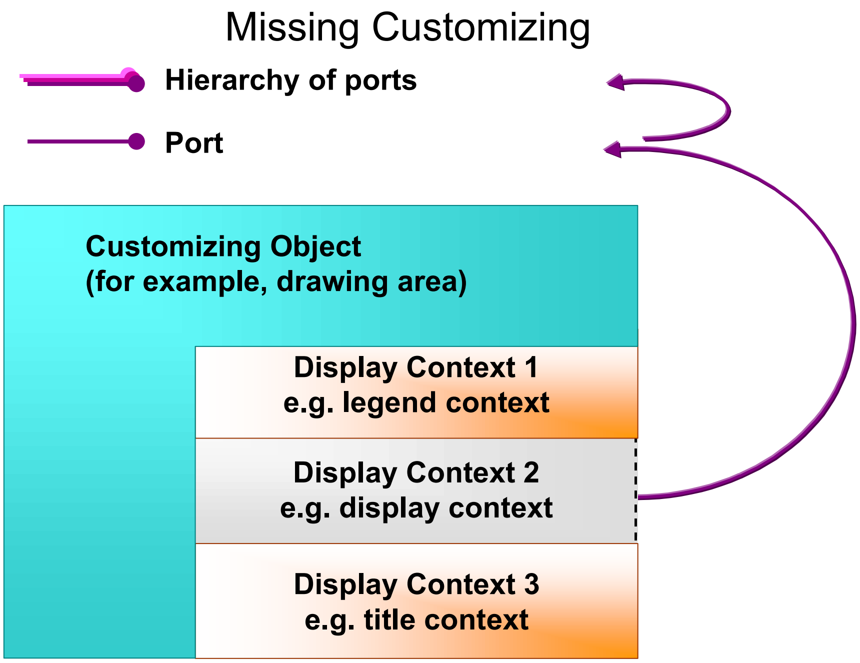
Just as there is a hierarchy of ports so too there is a hierarchy of display contexts. For more information, see Hierarchy of Display Contexts [Page 61].

Hierarchy of Display Contexts

Use

The hierarchy of display contexts determines where the graphics proxy looks for a display context if it does not find it in the customizing bundle assigned to the port. If a display context is used in several customizing bundles it is more efficient to assign it once to a port higher up the hierarchy where it can be used by all of the customizing bundles instead of assigning it to each of the individual customizing bundles.

Features



The whole customizing object is assigned to the port. The display contexts are attributes of the customizing object. Context 1 and context 3 are inside the customizing object. Context 2, however, is assigned to a port further up the hierarchy of ports.

The graphics proxy looks for context 2 in the customizing object (as a substructure). When it does not find it there it looks for it at the same port but not at the substructure level. The graphics proxy searches up the hierarchy of ports until it finds context 2.

Creating Customizing Bundles

Use

To set customizing attributes you have to create an instance of the relevant attribute bundle.

Features

Example of creating a customizing bundle for the drawing area

1. Define a reference variable for the attribute bundle class

```
<reference variable> TYPE REF TO CL_CU_DRAWING_AREA.
```

2. Create an instance of the attribute bundle using the reference variable defined earlier.

```
CREATE OBJECT <reference variable> EXPORTING
INSTANCE_ID  = <'instance ID'>
ADMIN               = <GFW_TRUE or GFW_FALSE>.
```

The instance name is required for accessing the database. The parameter ADMIN is optional and defines whether the bundle contains general or user-specific settings.

Instance name

The instance name is used in two places:

- As part of the key on the database

  You must select a unique key for a customizing class so that no other application is affected.

- In the communication between R/3 and the GUI when assigning attributes

  You must ensure that instances of one and the same class have different names.

Example of two instances with the same key

1. Create the reference variables for the attribute bundle class

```
DATA:  CU1 TYPE REF TO CL_CU_VALUES,
             CU2 TYPE REF TO CL_CU_VALUES.
```

2. Create the instances of the attribute bundles.

```
CREATE OBJECT CU1 EXPORTING INSTANCE_ID = 'MyApplication'.
CREATE OBJECT CU2 EXPORTING INSTANCE_ID = 'MyApplication'.
```

This results in an error ! Only one of the instances will be saved on the database; only one instance will affect the appearance of both data series in the GUI.

Assigning Customizing Bundles to Proxies

Use

To display the customizing settings in the graphic the customizing bundles have to be assigned to ports in the graphics proxy. You use the method IF_GRAPHIC_PROXY~ADD_CU_BUNDLE to make this assignment.

Integration

If the graphic is not yet active (the method IF_GRAPHIC_PROXY~ACTIVATE has not been called yet) the new setting is saved and takes effect when the graphic is activated. If the graphic is already active the new setting takes effect immediately (with the next call of the data container method IF_DC_MANAGEMENT~DISTRIBUTE_CHANGES).

The graphics proxy tells the multiplexer that the new object is to be included in its customizing settings. It is then informed of all subsequent changes to this object.

Prerequisites

The method INIT has already been called.

Features

Depending on the implementation of the graphics proxy certain customizing objects should be assigned to specific ports. For information on the possible combinations of customizing objects and ports, see the documentation in the relevant graphics proxy.

IF_GRAPHIC_PROXY~DEL_CU_BUNDLE

This method is the counterpart to ADD_CU_BUNDLE. It does *not* delete attribute values from the database. Instead it deletes the graphic proxy's reference to the customizing bundle.

For more information, see the documentation on DEL_CU_BUNDLE and ADD_CU_BUNDLE in the class builder.

Activities

CALL METHOD GP->IF_GRAPHIC_PROXY~ADD_CU_BUNDLE EXPORTING

PORT = IF_GRAPHIC_PROXY=>(name of port)

KEY   = (variant key, if required)

BUNDLE = (customizing bundle)

CUOBJ_LIST = (list of items for hierarchy graphics).

For more information on the method ADD_CU_BUNDLE, see the documentation in the class builder.

For more information, see Ports [Page 70].

For more information, see Variant Keys [Page 65].

For more information, see Customizing Objects [Page 57] and Attribute Bundles [Page 58].

Variant Key

Definition

A variant key is a key for a customizing object. A variant of a customizing object consists of the combination: variant key + customizing object. Each key is unique for a given attribute bundle type (for example, CL_CU_VALUES) and for a given port.

Use

You can use a variant key to save several customizing bundles belonging to the same class at the same port.

Example

You have two curves, one showing the age of an employee and the other indicating how long the employee has worked at a company. To give these curves a different appearance the application assigns them different customizing bundles. The variant keys are used to distinguish between the two curves.

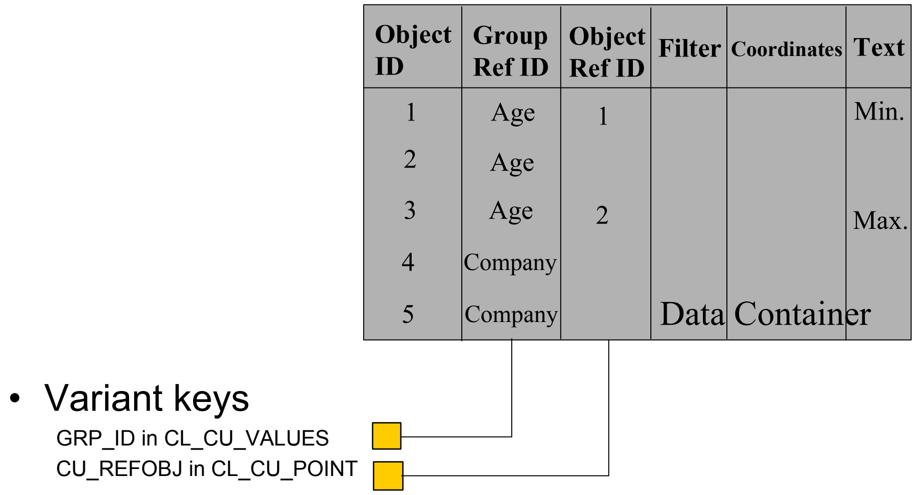
Integration

Variants of customizing objects for a data series can be defined for the port *drawing area*. The keys for them are in the data container under *Group Reference ID*.

Variants of customizing objects for points can be defined for the port *drawing area*. The keys for them are in the data container under *Object Reference ID*.

The application assigns a customizing bundle to a port in a graphics proxy using the method ADD_CU_BUNDLE in the interface IF_GRAPHIC_PROXY. This method has a parameter KEY with the text for the variant key. The same key must be used when filling the data container!

Links to the Data Container



Object ID	Group Ref ID	Object Ref ID	Filter	Coordinates	Text
1	Age	1			Min.
2	Age				
3	Age	2			Max.
4	Company				
5	Company		Data	Container	

- Variant keys

  GRP_ID in CL_CU_VALUES

  CU_REFOBJ in CL_CU_POINT

For more information on the attributes *Group Reference ID* and *Object Reference ID* in the data container, see Data Container for Business Graphics [Page 36].

Setting Parameters

Use

To set an attribute value for a customizing object use the method IF_CUSTOMIZING~SET. You can use this method to set a whole display context for a customizing object.

Prerequisites

You have already created an instance of the relevant customizing object.

Features

If a display context is set all the attributes of this context instance are copied. No reference to the original context is held.

Since the graphics proxy is given a *reference* to the customizing object the parameters for the customizing object can be set before or after it is assigned to the graphics proxy using IF_GRAPHIC_PROXY~ADD_CU_BUNDLE.

Activities

Setting a title for the drawing area:

```
CALL METHOD <reference variable for drawing area>->IF_CUSTOMIZING~SET EXPORTING

ATTR_ID = CL_CU_DRAWING_AREA=>CO_TITLE

VALUE    = (text title).
```

ATTR_ID specifies the attribute to be changed in the bundle. VALUE contains the new value of the attribute.

Setting a display context for the title of the drawing area:

1. Create display context:

```
CREATE OBJECT <display context> EXPORTING INSTANCE_ID = 'Dummy'.
```

2. Set area style of display context to waves

```
CALL METHOD <display context>=>IF_CUSTOMIZING~SET
    EXPORTING
    ATTR_ID = CL_CU_DISPLAY_CONTEXT=>CO_AREA_STYLE
    VALUE = 1. "Structure
CALL METHOD <display context>=>IF_CUSTOMIZING~SET
    EXPORTING
    ATTR_ID = CL_CU_DISPLAY_CONTEXT=>CO_BG_STRUCTURE
    VALUE = 13. "Waves
```

3. Set display context of drawing area as title context

```
CALL METHOD <drawing area>->IF_CUSTOMIZING~SET
    EXPORTING
```

```
ATTR_ID = CL_CU_DRAWING_AREA=>CO_TITLE_CONTEXT
VALUE    = <display context>.
```

Creating Default Bundles

Use

When you call the constructor of a customizing bundle it automatically calls the method IF_CUSTOMIZING~SET_DEFAULTS. This method ensures that there are default customizing settings to determine the appearance of the graphic in case the application does not make its own settings.

You can also call this method to reset the attribute values of a customizing bundle.

Integration

The default values are specified in the constructor method of the customizing bundle.

Prerequisites

An instance of the relevant customizing bundle has been created.

Features

You can reset all the attributes to their default values using the method IF_CUSTOMIZING~SET_DEFAULTS – this means that constructor default settings are used. Any display contexts that exist are removed.

Ports

Definition

A port indicates which attribute bundles can be used with which graphics proxies.

Use

The customizing objects are assigned to the ports in the graphics proxies. Only some combinations of ports and customizing objects are possible. Not every customizing object can be assigned to every port. For more information on possible combinations see the class documentation for the appropriate graphics proxy.

The ports in a graphics proxy are arranged in a hierarchy that determines where the proxy looks for customizing objects. For more information, see Hierarchy of Ports [Page 75].

Structure

The ports are defined as constants in IF_GRAPHIC_PROXY. Their names have the prefix CO_PORT_ they are all documented in the class builder.

Ports for Business Graphics Proxy

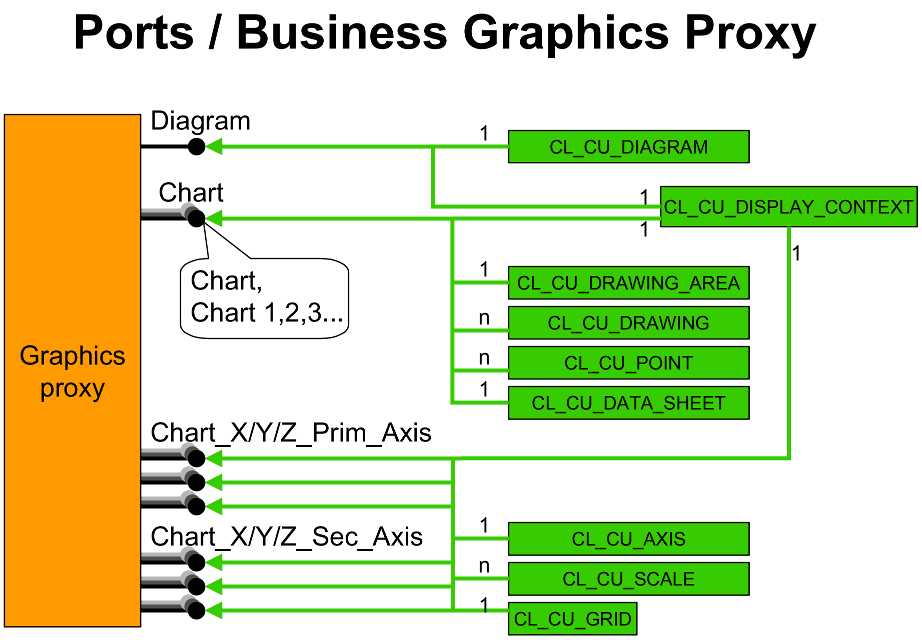
Use

The customizing bundles are assigned to ports in the graphics proxy. However, the combinations of ports and bundles are NOT arbitrary. You must ensure that you assign the right bundles to the right ports.

Prerequisites

You have already created an instance of the graphics proxy and of the customizing bundles to be assigned to the ports.

Features



The graphic above shows the business graphics proxy CL_GUI_GP_PRES with it ports. The black dots on the right of the graphics proxy are the ports. The customizing objects CL_CU_DIAGRAM, CL_CU_DISPLAY_CONTEXT, CL_CU_DRAWING_AREA and so on are assigned to the ports.

A display context can be assigned to a port or it can be inside another customizing bundle. The graphic above shows that CL_CU_DISPLAY_CONTEXT can be assigned, for example, to the port *Chart* or to CL_CU_DRAWING_AREA.

Ports for Business Graphics Proxy

For information on permitted combinations of ports and bundles for business graphics, see the class documentation of CL_GUI_GP_PRES.

Ports for Structure Graphics Proxy

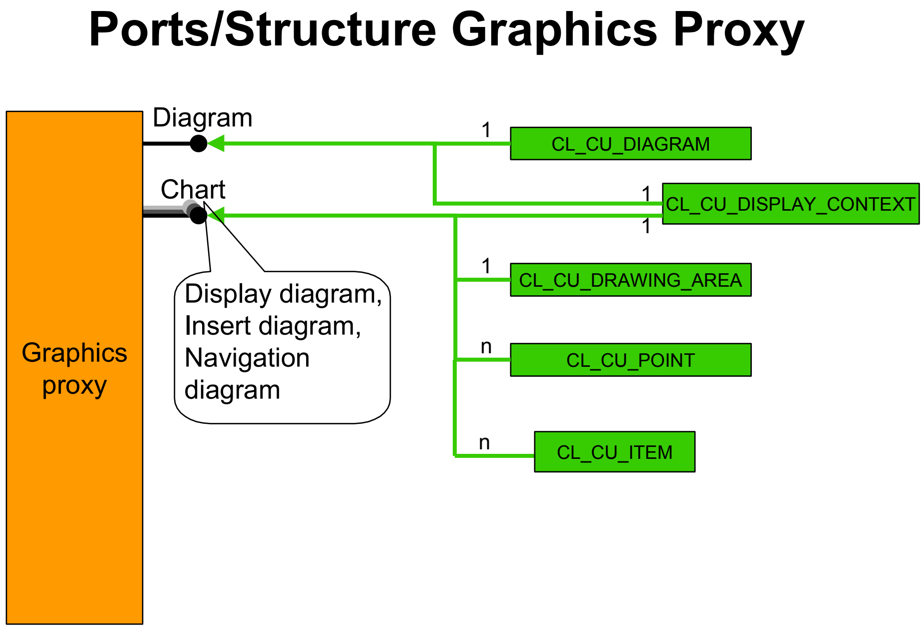
Use

The customizing bundles are assigned to ports in the graphics proxy. However, the combinations of ports and bundles are NOT arbitrary. You must ensure that you assign the right bundles to the right ports.

Prerequisites

You have already created an instance of the graphics proxy and of the customizing bundles to be assigned to the ports.

Features



The graphic above shows the structure graphics proxy CL_GUI_GP_HIER with its ports. The black dots on the right of the graphics proxy are the ports. The customizing objects CL_CU_DIAGRAM, CL_CU_DISPLAY_CONTEXT, CL_CU_DRAWING_AREA and so on are assigned to the ports.

Every point bundle can have an associated list of item bundles.

A display context can be assigned to a port or it can be inside another customizing bundle. The graphic above shows that CL_CU_DISPLAY_CONTEXT can be assigned, for example, to the port *Chart* or to CL_CU_DRAWING_AREA.

Ports for Structure Graphics Proxy

For information on permitted combinations of ports and bundles for structure proxies, see the class documentation of CL_GUI_GP_HIER.

Hierarchy of Ports

Use

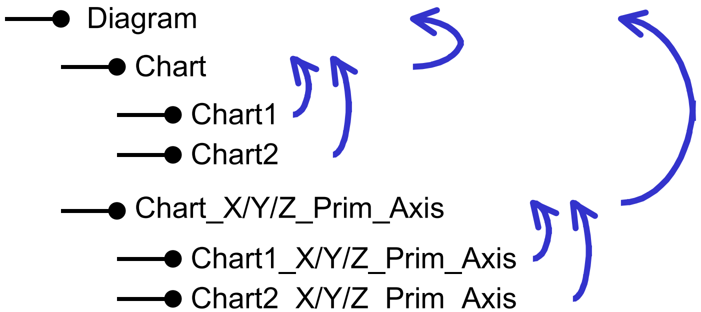
The ports in a graphics proxy are arranged as a hierarchy. This hierarchy determines where the graphics proxy looks for a customizing object if it does not find it at the port where it expects to find it.

Integration

The application creates the customizing object and transfers it to the graphics proxy specifying the port to which the customizing object is to be assigned.

Features

Hierarchy of Ports



The arrows refer to where the graphics proxy looks for a customizing object if it cannot find it at the port where it last looked for it.

For example, the application assigns the customizing object containing attributes for the axis to the port CHART_X_PRIM_AXIS. The graphics proxy first looks for the customizing object at the most specific port, say, CHART1_X_PRIM_AXIS and since it cannot find it there it looks at the next most general port CHART_X_PRIM_AXIS where it finds it.

Connection to the Database

Use

The application can save customizing objects in tables in the database.

Features

There are different tables for the various types of customizing object (for example, for the drawing area, the axis, the scale, and so on).

Customizing Object	General Settings	Customer-Specific Settings
Display context	GFWCUDC	GFWCUDCU
Chart area	GFWCUDI	GFWCUDIU
Drawing area	GFWCUDA	GFWCUDAU
Axis	GFWCUAX	GFWCUAXU
Scale	GFWCUSC	GFWCUSCU
Data series	GFWCUVA	GFWCUVAU
Point / object	GFWCUPO	GFWCUPOU
Data sheet	GFWCUDS	GFWCUDSU
Grid	GFWCUGR	GFWCUGRU
Item	GFWCUIT	GFWCUITU

In the table above the middle column lists names of database tables containing customizing objects available to all users. They contain general settings.

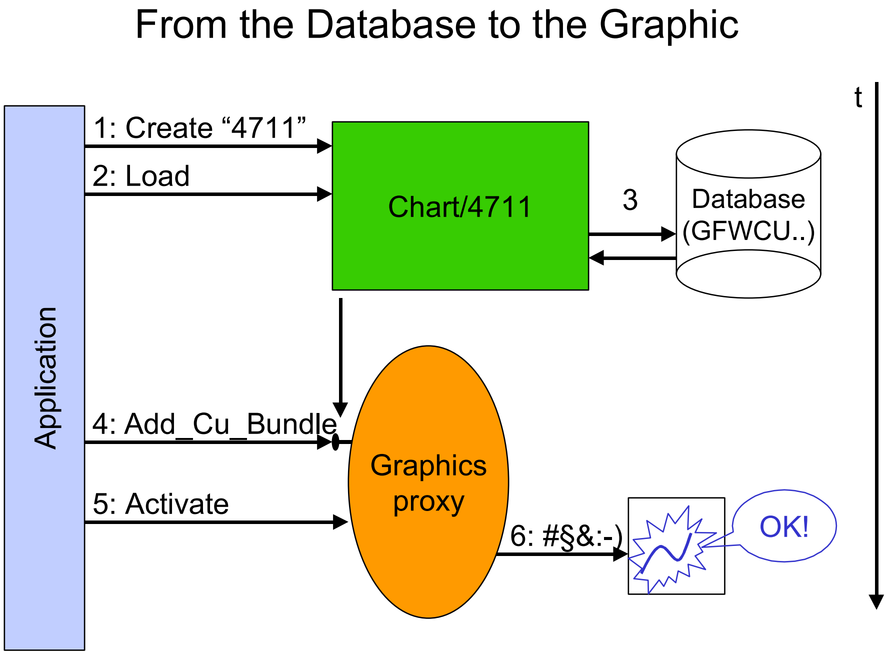
The column on the right lists names of database tables containing customizing objects for specific users. Customizing settings for specific users take priority over general customizing settings during loading.

Customizing: From Database to Graphic

Purpose

A group of display attributes for the graphic is created and then transferred to the graphic.

Process Flow



The graphic shows what happens when customizing data is transferred from the database to the graphic. The order of events can be read vertically down the graphic:

1. The application creates the customizing object. In this example it has the ID 4711.

```
CREATE OBJECT <drawing area> EXPORTING
    INSTANCE_ID = "4711".
```

2. The application loads the attribute values from the database.

```
CALL METHOD <drawing area>→IF_CUSTOMIZING~LOAD.
```

3. Data is read from the database.

4. Using the method ADD_CU_BUNDLE the application notifies the graphics proxy about the customizing object and at which port it is to be used.

```
CALL METHOD <graphics proxy>→IF_GRAPHIC_PROXY~ADD_CU_BUNDLE
        EXPORTING   PORT = IF_GRAPHIC_PROXY  => CO_PORT…
                                    BUNDLE = <drawing area>
        IMPORTING   RETVAL = <return value>.
```

5. After all the customizing steps have been completed the graphics proxy has to be activated.

```
    CALL METHOD <graphics proxy>→ACTIVATE
            IMPORTING RETVAL = <return value>.
```

6. The graphic is displayed with the display attributes specified in the customizing object. The label 6:#§&:-) is to indicate that the method call to the graphic is *product-specific*.

IF_CUSTOMIZING~LOAD_CUSTOMIZING

Use

The graphics proxy uses this method to load customizing bundles from the database. You can use this method instead of calling IF_CUSTOMIZING~LOAD for every customizing bundle.

Integration

IF_GRAPHIC_PROXY~LOAD_CUSTOMIZING can be regarded as the counterpart to the method IF_GRAPHIC_PROXY~SAVE_CUSTOMIZING. The latter saves customizing bundles to the database.

Prerequisites

Customizing bundles have already been saved to the database.

Features

IF_CUSTOMIZING~LOAD_CUSTOMIZING ensures that the default values set in IF_CUSTOMIZING~SET_DEFAULTS are not used.

A customizing bundle such as a drawing area can have several contexts, for example, a title context and a legend context. When the bundle is loaded from the database these contexts automatically get loaded as well.

When you call the method you can specify that the values are to be loaded from a client other than the current client. The default client is the current client.

Activities

IF_CUSTOMIZING~LOAD_CUSTOMIZING loads customizing settings bundle by bundle from the database. These are the customizing bundles that the application previously assigned to the graphics proxy using the method IF_GRAPHIC_PROXY~ADD_CU_BUNDLE.

IF_CUSTOMIZING~LOAD

Use

The application calls this method to load all the attribute values of a customizing bundle from the database. If no values are available for the current user the appropriate general settings are loaded.

Integration

For more information, see Customizing Objects in Database [Page 76].

Prerequisites

Customizing bundles have already been saved to the database.

Features

Parameter ADMIN = GFW_FALSE

The values that are saved under the name INSTANCE_ID are loaded first from the user-specific table. If one or more attributes do not exist there then they are loaded from the general table.

For more information, see Creating Customizing Bundles [Page 62].

Activities

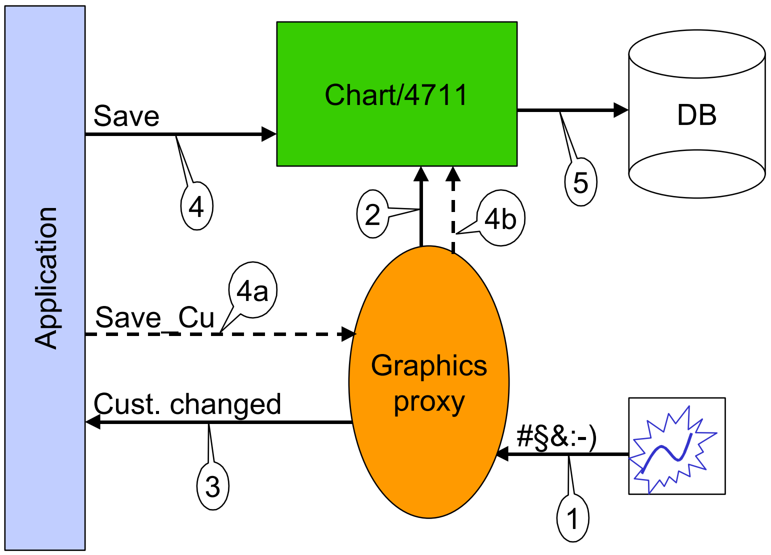
Customizing: From Graphic to Database

Purpose

You can change display attributes in the graphic interactively and these changes can then be saved to the database.

Process Flow

From the Graphic to the Database



The graphic shows the order of events from when the user changes the graphic to when the change is saved on the database.

1. A change takes place in the graphic and the graphic sends a GUI event to the graphics proxy. The label #§&:-) is to indicate that the event is product-specific. It is not a standard ABAP event.
2. The graphics proxy writes the changes to the customizing objects.
3. The graphics proxy sends an ABAP event to the application to inform it of the change.

There are two options for saving customizing changes:

4 + 5: The application can save customizing settings directly to the database by individually saving all the customizing objects.

```
CALL METHOD <drawing area> → IF_CUSTOMIZING~SAVE
    IMPORTING RETVAL = <return value>.
```

4a + 4b + 5: Alternatively, the application can use the method SAVE_CUSTOMIZING in IF_GRAPHIC_PROXY (4a) to tell the graphics proxy to save on the database all the customizing objects it is aware of.

```
CALL METHOD <graphics proxy> → IF_GRAPHIC_PROXY~SAVE_CUSTOMIZING
     IMPORTING RETVAL = <return value>.
```

IF_GRAPHIC_PROXY~SAVE_CUSTOMIZING

Use

You use this method to save (changed) customizing settings to the database for *all* the customizing bundles that the graphics proxy knows.

Integration

Calling IF_GRAPHIC_PROXY~SAVE_CUSTOMIZING calls the method IF_CUSTOMIZING~SAVE for each customizing bundle.

Alternatively, the application can save customizing settings by calling the method IF_CUSTOMIZING~SAVE for a specific customizing object.

Prerequisites

The customizing bundles have already been assigned to the graphics proxy.

Features

The graphics proxy automatically creates default customizing bundles for bundle types for which the application does not provide customizing bundles. The default customizing bundles are *not* saved.

Activities

The graphics proxy requests that *all* attribute bundles that it knows be saved to the database.

IF_CUSTOMIZING~SAVE

Use

You can use the method IF_CUSTOMIZING~SAVE to save changed attributes to the database. You can also include the changes in a transport request using the parameters TRANSPORT and TRANSPORT_REQ.

Features

If the bundle is used by the administrator (see the parameter ADMIN in the CONSTRUCTOR method of the bundle concerned) the general values are saved – those values that are valid for all users.For more information, see Creating Customizing Bundles [Page 62].

Including changes in a transport request

In addition to saving you can also include the changes in a transport request by using the parameter TRANSPORT. (However, the parameter TRANSPORT has no influence on saving).

You specify a transport task using the parameter TRANSPORT_REQ.

For more information, see the documentation on IF_CUSTOMIZING~SAVE in the class builder.

Deleting Attributes from Database

Use

There is a method you can use to delete attributes that are no longer needed from the database.

Features

IF_CUSTOMIZING~DELETE

This method deletes all the attribute values of the bundle from the database. If the bundle is used by the administrator (see parameter ADMIN in the constructor method of the bundle concerned) the general values, which are valid for all users, are deleted. In the other case the user-specific values are then deleted.

For more information, see the documentation on IF_CUSTOMIZING~DELETE in the class builder.

There is another method you can use to delete the graphic proxy's reference to the customizing bundle instead of deleting attributes on the database. For more information, see Assigning Customizing Bundles to Proxies [Page 63].

Changing Appearance of Graphic

Use

You can change the appearance of graphics in two ways:

You can change customizing in the graphic interactively and then save these changes on the database if required. For more information, see Customizing: From Graphic to Database [Page 81].

Alternatively you can use the ABAP Objects interface to change attributes. In this case the application creates a customizing object and changes attributes before the graphic is displayed. For more information, see Customizing: From Application to Graphic [Page 87].

Features

Customizing that can be changed

You can change customizing objects that are assigned to ports in a graphics proxy.

Customizing that cannot be changed

The graphics proxy creates default customizing objects if the customizing objects required have not been assigned to the ports. These default customizing objects cannot be changed.

IF_CUSTOMIZING~SET_MASTER

You can use this method to set a master flag for an attribute. If the master flag is set users cannot change this attribute. You can specify the attribute concerned and whether its master flag is to be set or deleted.

Interaction and property pages

If the application has not registered for the event GRAPHIC_CUSTOM_CHANGED then interaction is not possible. In this case the property pages are not displayed. Interaction is deactivated in the default setting.

To register for events the application specifies a list of events it is interested in using the parameter EVTCODE_LIST in IF_GRAPHIC_PROXY~INIT.

Customizing: From Application to Graphic

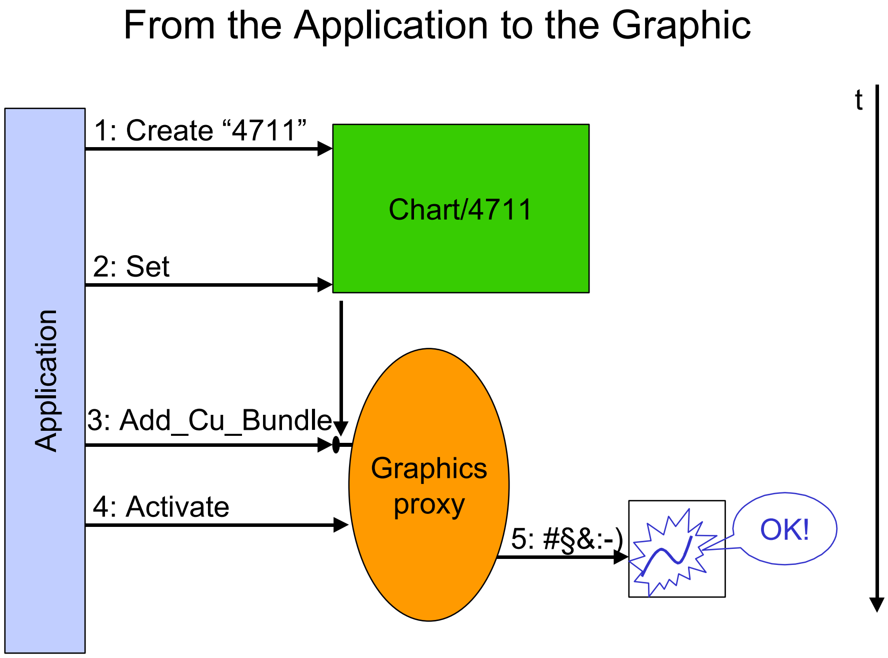
Purpose

The application can create a customizing object, then change various attributes in this object and assign the object to a port in the graphics proxy. Once the graphics proxy has been activated the graphic is displayed with the visualization attributes specified in the customizing object.

Prerequisites

In this case the customizing object is not loaded from the database and does not have to be saved to the database.

Process Flow



The graphic shows what happens when the application creates customizing objects, changes attributes and transfers the customizing data to the graphic. The order of events can be read vertically down the graphic:

7. The application creates the customizing object. In this example of a drawing area it has the ID 4711.

```
CREATE OBJECT <instance of CL_CU_DRAWING_AREA>
    EXPORTING INSTANCE_ID = "4711".
```

8. The application can change individual values, for example the title of the drawing area.

```
CALL METHOD <instance of CL_CU_DRAWING_AREA> → IF_CUSTOMIZING~SET
       EXPORTING    ATTR_ID = CL_CU_DRAWING_AREA => CO_TITLE
```

```
                                        VALUE = "This is the new title".
```

9. Using the method ADD_CU_BUNDLE the application notifies the graphics proxy about the customizing object and at which port it is to be used.

```
CALL METHOD <instance of graphics proxy >
 → IF_GRAPHIC_PROXY~ADD_CU_BUNDLE
      EXPORTING   PORT = IF_GRAPHIC_PROXY  => CO_PORT…
                            BUNDLE = <instance of
CL_CU_DRAWING_AREA>
      IMPORTING    RETVAL = <return value>
```

10. After all the customizing steps have been completed the graphics proxy has to be activated.

```
CALL METHOD <instance of graphics proxy>  → ACTIVATE
      IMPORTING RETVAL = <return value>
```

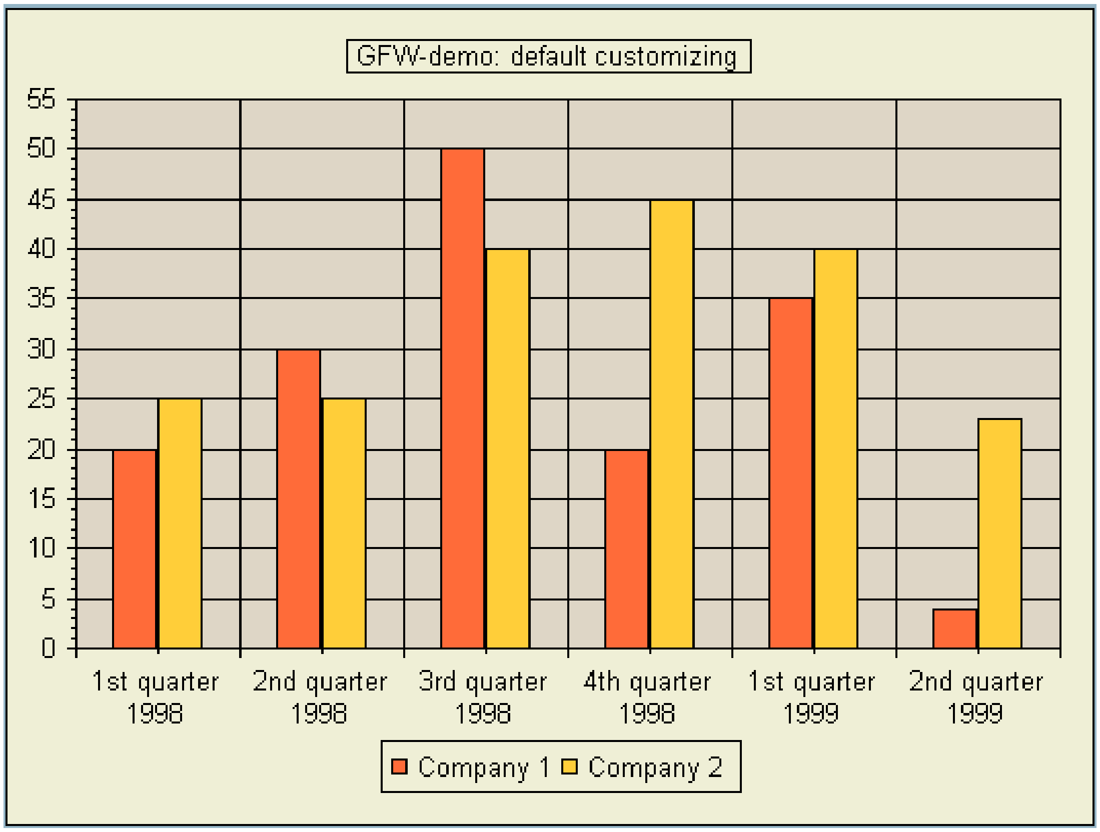
11. The graphic is displayed with the display attributes specified in the customizing object. The label 5:#§&:-) is to indicate that the method call is *product-specific*.

CREATE_CUSTOMIZING

Use

You can use the method CL_GUI_GP_PRES→CREATE_CUSTOMIZING to generate attractive customizing settings with a minimum of effort.

Graphic with Customizing generated using CREATE_CUSTOMIZING



Integration

There are three options with respect to Customizing with the Graphical Framework:

- If you make no Customizing settings the graphics proxy automatically generates default customizing. This involves a minimum of effort but the Customizing settings are rather bland.

  The application cannot access the normal default objects, for example using the methods mentioned in the section *Features*.

- You can use the method CL_GUI_GP_PRES→CREATE_CUSTOMIZING.
- You can make individual settings using the Graphical Framework's powerful Customizing concept. This provides great flexibility but is more time-consuming.
- A mixture of the last two point as in the example program GFW_PROG_CREATE_CUSTOMIZING.

Prerequisites

CREATE_CUSTOMIZING is only available for business graphics.

Features

Changing and saving Customizing settings

You can change the settings generated by CREATE_CUSTOMIZING using the methods:

- IF_GRAPHIC_PROXY~GET_CU_BUNDLE
- IF_GRAPHIC_PROXY~DEL_CU_BUNDLE

You can also save changes to / load settings from the database using the methods:

- IF_GRAPHIC_PROXY~LOAD_CUSTOMIZING
- IF_GRAPHIC_PROXY~SAVE_CUSTOMIZING

Parameters

INSTANCE_ID: You specify an instance ID and the graphics proxy ensures that if there are several instances of a class (for example, CL_CU_AXIS) they each have a different ID.

GRPIDS: You specify a list of group IDs and the method creates an instance for each entry in the list.

POINTIDS: You specify a list of point IDs and the method creates an instance for each entry in the list.

For more information, see the class builder documentation.

Activities

```
* Define variable for list of group IDs

DATA: GROUPS TYPE GFW_GRPID_LIST.

* Add group IDs to list

APPEND <first group ID> TO GROUPS.

APPEND <next group ID> TO GROUPS.

* Call method

CALL METHOD <instance of graphics proxy>→CREATE_CUSTOMIZING

        EXPORTING    INSTANCE_ID  = '<any unique instance ID>'

                                  GRPIDS                = GROUPS

        IMPORTING    RETVAL                = RETVAL.
```

Setting Labels

Use

You use labels to display texts for data series or points in a graphic.

Labeling a data series

You label a data series using CL_CU_VALUES.

Labeling points

You label a point in a data series using CL_CU_POINT.

Integration

If the attribute CL_CU_VALUES=>CO_LABEL_AUTO = GFW_FALSE the text for the label is taken from the data container. If this attribute = GFW_TRUE then the label text depends on the setting in CO_LABEL_TYPE (for example, a value, percentage, category and so on).

Prerequisites

Data series: You have already created an instance of a data series (CL_CU_VALUES).

Point: You have already created an instance of a point (CL_CU_POINT).

Activities

To set a display context for the label use the attribute CO_LABEL_CONTEXT in CL_CU_VALUES or CL_CU_POINT as appropriate.

To specify the position of the label use the attribute CL_CU_VALUES=>CO_LABEL_POSITION. The position determines where the label appears in relation to the data series.

To align the label use the attributes CL_CU_VALUES=>CO_LABEL_H_ALIGNMENT / CO_LABEL_V_ALIGNMENT. The alignment determines where the label appears in relation to the small text field that contains the text.

Specific Customizing for Structure Graphics

Use

To customize the graphic as a whole use CL_CU_DIAGRAM.

To customize the main area of the control use CL_CU_DRAWING_AREA. With networks there can be several drawing areas for one control, for example a display area on the left and a navigation area on the right.

To customize nodes, also known as *points*, use CL_CU_POINT.

To customize sub-objects of nodes, known as *items*, use CL_CU_ITEM.

Prerequisites

The application has to assign customizing bundles to ports. For more information, see Assigning Customizing Bundles to Ports [Page 94].

Features

There are fewer attribute bundles than for business graphics.

For more information, see Point Customizing [Page 95].

For more information, see Item Customizing [Page 98].

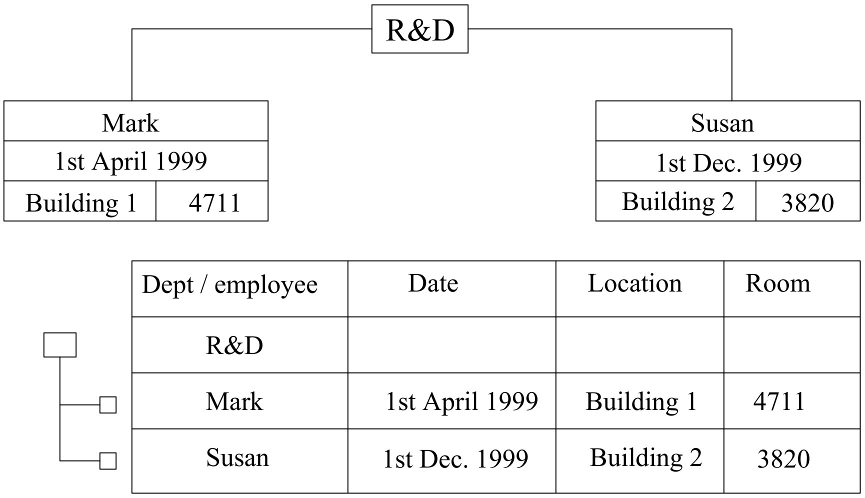
Relationship between Graphic and Data Container

Use

The data container is used, among other things, to store customizing information for the graphic.

Integration

Customizing: Relationship between Graphic and DC



R&D

Mark
1st April 1999
Building 1 | 4711

Susan
1st Dec. 1999
Building 2 | 3820

Dept / employee	Date	Location	Room
R&D			
Mark	1st April 1999	Building 1	4711
Susan	1st Dec. 1999	Building 2	3820

The top of the figure above displays a hierarchy graphic with three points (R&D, Mark and Susan). The bottom of the slide displays a section of the associated data container. The rows starting ”R&D”, “Mark” and “Susan” are points. The individual elements such as “Mark”, “1st April 1999” and “Building 2” are items. SET_DC_NAMES contains a list of items. The columns in the data container have numbers – item numbers.

Assigning Customizing Bundles to Ports

Use

You use the method ADD_CU_BUNDLE to assign customizing bundles to ports.

You use the parameter CUOBJ_LIST to assign a list of items to the port in addition to point customizing.

You can use variant keys to specify different customizing for different sets of points. For example, one group of customizing settings for a set of points representing departments and another group of customizing settings for a set of points representing employees.

Prerequisites

You should read the class documentation of CL_GUI_GP_HIER to find out about the possible combinations of customizing objects and ports.

Features

CUOBJ_LIST

The parameter CUOBJ_LIST is a list of additional customizing objects. This list is assigned to a CL_CU_POINT customizing object. With structure graphics it contains the CL_CU_ITEM objects that describe the individual items (for example, columns in a tree) for a node.
The number of the item in the table determines how it is used.

KEY

The parameter KEY (the variant key) allows you to assign several variants of a CU object to a port. You can define variants of point customizing objects whose variant keys are in the data container; for example, one customizing object for nodes as folders and one for file nodes in a directory hierarchy.

ADD_CU_BUNDLE *with* a variant key

ADD_CU_BUNDLE with a variant key determines the layout of the items for those points with the variant key assigned to the parameter KEY. Clearly, ADD_CU_BUNDLE can be called several times since there may be several variant keys, for example, it can be called once for points defined as folders and once for points defined as leaves.

ADD_CU_BUNDLE *without* a variant key

ADD_CU_BUNDLE without a variant key determines the layout of the items for those points with a variant key to which no extra customizing was assigned.

Default customizing

As with other graphics types, if ADD_CU_BUNDLE is *not* called default customizing is used.

Point Customizing

Use

You use the class CL_CU_POINT to make customizing settings for points (nodes).

Integration

A point (node) can have one or more sub-objects, known as items. For more information, see Item Customizing [Page 98].

Features

CO_ITEM_ORDER

You can specify the order of items using the attribute CO_ITEM_ORDER. For more information, see CO_ITEM_ORDER [Page 97].

CO_EXPANDABLE

A leaf is the last element in the hierarchy (= node without children). If CO_EXPANDABLE = GFW_TRUE then the node is a folder rather than a leaf even if it is the last node in a hierarchy. SAP Tree displays an expander icon for expandable nodes.

CO_ICON_LEAF, CO_ICON_NODE_EXPANDED / -COLLAPSED

The CO_ICON attributes are only supported by SAP Tree. If no CO_ICON_NODE_EXPANDED icon is provided then the CO_ICON_NODE_COLLAPSED icon will be used; if no CO_ICON_NODE_COLLAPSED icon is provided then the CO_ICON_LEAF icon will be used – the icon for a node that is expanded.

CO_ENABLE

This attribute determines whether a node is interactive. Examples of interaction at a node (its items) would be selecting a checkbox or clicking on a button \ link.

Default Customizing

If point customizing is not assigned to a row then default customizing is used. You can make use of the hierarchy of ports to set your own default customizing.

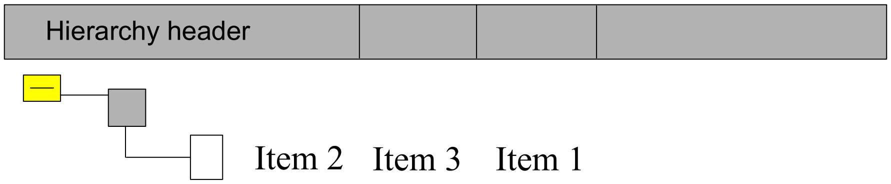
CO_ITEM_LAYOUT

You can specify the layout of the items, for example whether the fields are next to one another or above one another. This attribute is ignored by the tree control, which always places items next to one another in a line. For more information, see the documentation for this attribute in the class builder.

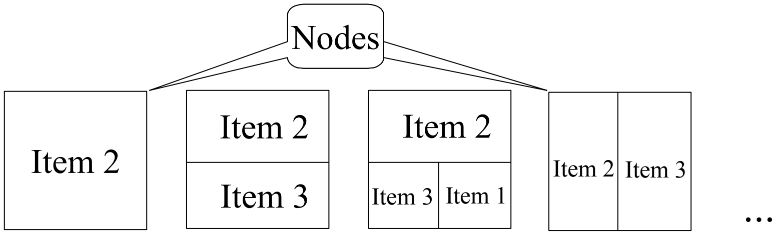
Layout of Items in a Node

CO_ITEM_ORDER = 2,|,3,1

A: Hierarchy graphics



B: Network graphics

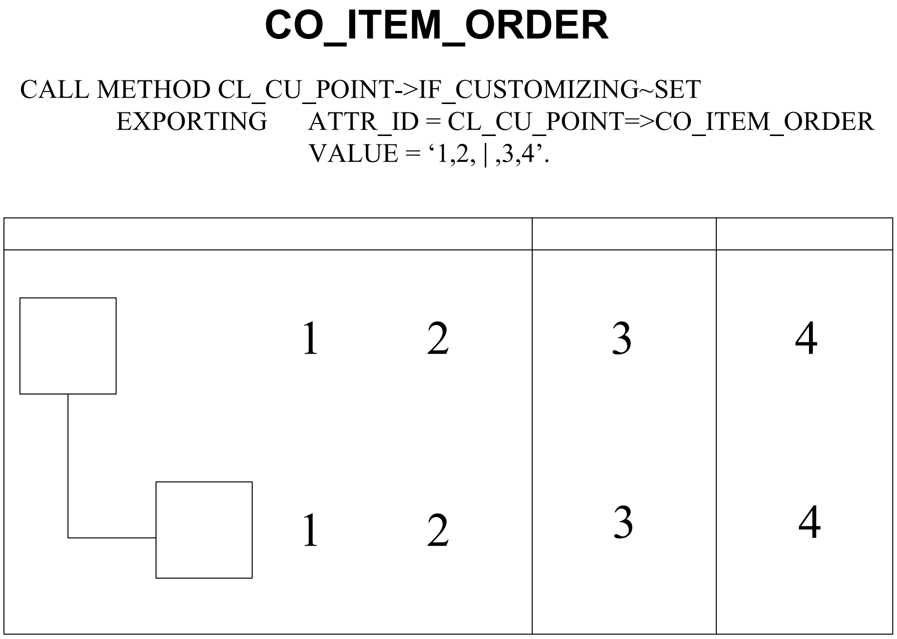


CO_ITEM_ORDER

Use

You can use this attribute to define the sequence of the items in all the points (nodes).

Integration



CO_ITEM_ORDER

```
CALL METHOD CL_CU_POINT->IF_CUSTOMIZING~SET
          EXPORTING      ATTR_ID = CL_CU_POINT=>CO_ITEM_ORDER
                         VALUE = ‘1,2, | ,3,4’.
```

You set the item order by calling the method IF_CUSTOMIZING~SET as in the coding above. The end of the hierarchy area is marked by a non-numerical character, "|" in the coding above. The items that follow are displayed in the table area of the control. The box below the coding represents the graphic. It has two points, one subordinate to the other, each of which has four items.

Features

The list of item customizing objects with the (item) numbers is sent to the graphics proxy as sub-objects for the default point customizing object (using the method ADD_CU_BUNDLE) in other words without a variant key.

For more information, see the documentation on CL_CU_POINT=>CO_ITEM_ORDER in the class builder.

Item Customizing

Use

Items are sub-objects of points (nodes). A single node can have several items. Customizing for items is in the class CL_CU_ITEM.

You can specify the following item types using CL_CU_ITEM=>CO_TYPE:

- Icons

  You write the icon in the format @**@ in the data container field – for example, @0A@ for a red traffic light.

- Checkboxes

  The first character for the value in the data container is reserved for the status of the checkbox – that is, whether or not it is selected.

- Links
- Texts

  You can add an icon in a text for the checkbox by adding the icon name to the text. The icon can be inserted either as a prefix or as a suffix to the text, but not in the middle of the text.
  Example: Leaf1@0A@

- Buttons

Integration

An item customizing object can only be assigned to a graphics proxy as a (numbered) sub-object of a point customizing object (see method ADD_CU_BUNDLE in the interface IF_GRAPHIC_PROXY).

Features

To define a ToolTip (a short text describing a function) use the attribute CO_TOOLTIP.

To specify whether an item is editable use the attribute CO_EDITABLE.

To define column headings in SAP Tree use the attribute CO_TITLE (without a variant key). For more information on column headings, see CO_ITEM_ORDER [Page 97].

For more information on item customizing, see the documentation in CL_CU_ITEM in the class builder.

For more information, see Creating and Parametrizing Items [Page 99].

For more information, see Item List and CUOBJ_LIST [Page 100].

For more information on specifying the order in which items are displayed, see CO_ITEM_ORDER [Page 97].

Creating and Parametrizing Items

Procedure

Creating an item

1. Create a reference variable for an item

```
DATA: CU_ITEM TYPE REF TO CL_CU_ITEM
```

2. Create an instance of the item

```
CREATE OBJECT CU_ITEM
EXPORTING INSTANCE_ID = <unique name>.
```

The INSTANCE_ID must be unique for all item customizing objects.

For more information, see Creating Customizing Bundles [Page 62].

Setting parameters for an item

Call the SET method in IF_CUSTOMIZING

```
CALL METHOD CU_ITEM -> IF_CUSTOMIZING~SET
        EXPORTING
        ATTR_ID = CL_CU_ITEM => CO_TYPE “type of item
        VALUE     = 1    “icon
        IMPORTING RETVAL = RETVAL.
```

You have to call the SET method every time you want to assign an attribute to an item.

Item List and CUOBJ_LIST

Procedure

Appending items to an item list

```
REFRESH CUOBJ_LIST.
    CUOBJ_SINGLE-NO = 1.
    CUOBJ_SINGLE-OBJ = CU_ITEM.
    APPEND CUOBJ_SINGLE TO CUOBJ_LIST.
```

Assigning the item list to the parameter CUOBJ_LIST in the method ADD_CU_BUNDLE

```
CALL METHOD GP -> IF_GRAPHIC_PROXY~ADD_CU_BUNDLE
    EXPORTING        PORT = CO_PORT_CHART
                 KEY = <variant key>
                 BUNDLE = <customizing bundle for point>
                 CUOBJ_LIST = CUOBJ_LIST
    IMPORTING        RETVAL = RETVAL.
```

There can be several item lists so the application can assign a different one each time it calls the method ADD_CU_BUNDLE. The application calls ADD_CU_BUNDLE once for each variant key.

Status Object

Definition

The status object contains status information on an object, for example that a node is "expanded". The status object may contain several items of information for an object.

Use

The status object is used by the application so that it can display a hierarchy graphic as required, for example with all the nodes expanded.

It is also used to synchronize the status of graphics.

Integration

CL_GFW_STATUS

The status object has to implement the interface IF_GFW_STATUS, in particular the COPY method, so that data containers can handle the status information as an attribute. The attribute for the status line in the data container references the class CL_GFW_STATUS which implements IF_GFW_STATUS provided by the Applications Graphics group.

SET_DC_NAMES / Data container

The parameter *status* in the method SET_DC_NAMES is the name of the status line in the data container. The status line contains information on the states of nodes, such as that they are “expanded”. If the name of the status line is specified then the required events (for example, “GRAPHIC_OBJ_EXPANDED”) are automatically registered. This guarantees that the status object in the DC always displays the current state of the object.
A status line is a reference to IF_GFW_STATUS. A status line for a structure graphic has to be a reference to IF_GFW_STATUS_HIER as well.

Information on possible statuses (“able” information such as "expandable") – as opposed to current statuses ("ed" information such as expanded) – is specified in Customizing. For example, you would specify this sort of information for nodes in CL_CU_POINT.

For more information, see the documentation on IF_GFW_STATUS in the class builder.

IF_CUSTOMIZING~TRANSPORT

Use

The application calls this method to include all the attribute values of a customizing class in a transport request.

Integration

If this method is implemented by a customizing class with context attributes all the values of the attributes belonging to these contexts are included as well.

Features

This method has the following parameter:

TRANSPORT_REQ (changing)

> This parameter contains the transport task. If no transport task is specified the system prompts you to enter a task. The name of the task selected is returned. However, if you do not want to transport the customizing settings (if you press "Cancel" in the dialog box), the return value is CL_GFW=>E_CU_TRANSPORT_CANCELED.

Events

Definition

Graphical events are events/user actions reported by the graphic.

Use

They are supplied by the GUI for specific products. Most ActiveX graphics products have the same event codes for the standard Windows events (for example, mouse-click) but there is no guarantee that this is the case for all products. It must be assumed that some products can send other events as well.

The application uses only standardized internal events, that is, constants defined by SAP. The graphics proxy automatically converts external product events to internal events and vice versa.

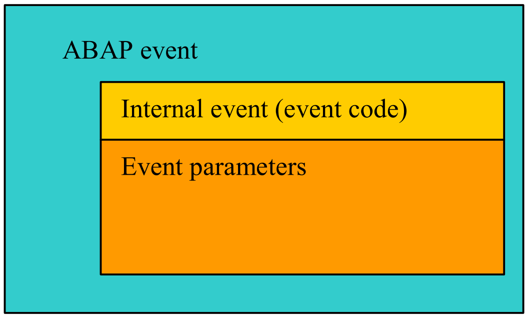
A graphics proxy converts an external GUI event it has received into an internal event and distributes it as an ABAP event.

The application implements appropriate event-handler methods so that it can receive the ABAP events in which it is interested that are sent to it by the graphics proxy. An application that wants to process all internal events in one place will implement its event handler for GRAPHIC_ANYEVT.

For more information, see Processing Events [Page 105].

Structure

Graphic showing the structure of an ABAP event



An ABAP event contains an internal event code and can contain event parameters.

The parameters for an event are supplied as an internal table with a name column and a value column. The parameters that can be expected for an event are described in the documentation for events in IF_GRAPHIC_PROXY.

Integration

The application specifies the events that are to reach the graphics proxy, that is, which events get through the front end/back end interface. This can be specified when the graphics proxy is initialized or using the method SET_EVENT_LIST by supplying a list of internal event codes. This ensures that communication between the front end and the back end is kept to a minimum and performance is optimized.

The internal event codes are defined as constants in IF_GRAPHIC_PROXY.

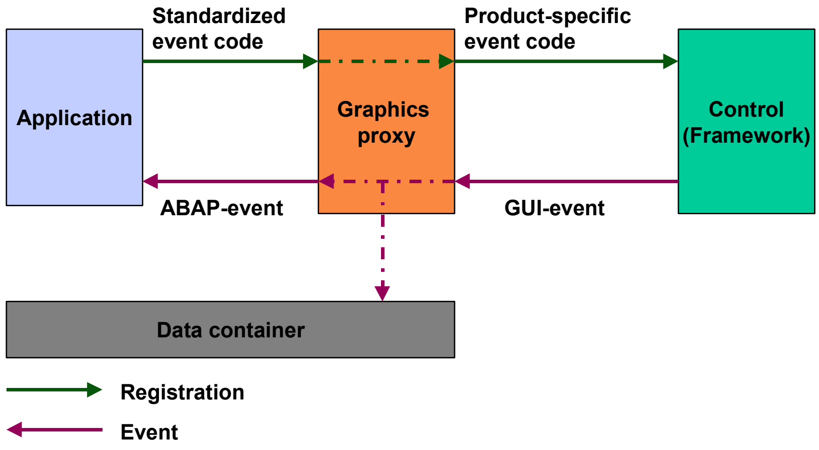
Processing Events

Use

Event processing deals with the conversion of standardized event codes to product-specific event codes and GUI events to ABAP events.

Integration

GUI-Event and ABAP- Event/Overview



If there is interaction in a graphic it creates and raises an event. This event has to be processed in the back end.

First, an event has to be registered. This is to minimize communication between the back end and the front end. The application registers the standardized (internal) event code with the graphics proxy. The graphics proxy registers the event code (converted to a product-specific code) with the Control Framework. For more information, see the documentation on the Control Framework.

If the graphic creates and raises this event the graphics proxy writes data changes to the data container (if necessary) and converts the product-specific event code to a standardized ABAP event. This ABAP event can then be handled by the application.

It is event codes that are sent from the application to the graphics proxy and from the graphics proxy to the Control Framework. It is events that are sent back in the other direction.

Features

Some ABAP events are sent by all graphics proxies (for example, GRAPHIC_ANYEVT). These events are defined and documented in the graphics proxy interface (IF_GRAPHIC_PROXY). Graphics proxies can also have their own event interface in their class definition.

Activities

Every graphics proxy distributes its internal event with a special ABAP event (for example, GRAPHIC_SELECTED) and also the ABAP event GRAPHIC_ANYEVT.

Events for Structure Graphics

Definition

Graphical events are events/user actions reported by the graphic.

Use

Structure graphics are highly interactive and for this reason events are very important for hierarchy graphics and network graphics.

Typical events include the following:

- GRAPHIC_BUTTON_CLICKED

  Items for a node can be displayed as pushbuttons. This event occurs when a pushbutton is pressed.

- GRAPHIC_LINK_CLICKED

  Items for a node can be displayed as links. This event occurs when a link is activated.

- GRAPHIC_CHECKBOX_CHANGED

  Items for a node can be displayed as checkboxes. This event occurs when the status of a checkbox is changed.

One of the most important functions involving events in structure graphics is Drag & Drop. You can use this function to drag one or more objects in one control and drop them onto an object in another control.

For more information, see Drag & Drop [Page 108].

See also: Drag & Drop [Ext.] in the Control Framework documentation.

Integration

Events and event codes are defined in the interface IF_GRAPHIC_PROXY as usual.

The events GRAPHIC_OBJ_EXPANDED and GRAPHIC_OBJ_COLLAPSED use the status attribute in the data container. For more information, see Status Object [Page 101].

Drag & Drop

Use

You can use the drag & drop function to drag one or more source objects and drop them on a target object.

Integration

CL_DRAGDROP

This class contains methods that describe the drag and drop behavior of a custom control.

CL_DRAGDROPOBJECT

Data is exchanged between the source and target objects by means of the so-called drag & drop object which is an instance of the class CL_DRAGDROPOBJECT

For more information, see Drag & Drop [Ext.] in the Control Framework documentation.

Prerequisites

IF_EVT_DRAGDROP

To implement drag & drop between controls the application must handle the events in the interface IF_EVT_DRAGDROP. For more information, see IF_EVT_DRAGDROP [Page 109]. *Every time* a Drag and Drop event occurs the events in IF_EVT_DRAGDROP have to be handled.

SET_BEHAVIOR_LIST

You have to specify the behavior for all objects. Note: you only have to do this once.

The application calls the method SET_BEHAVIOR_LIST to define the Drag & Drop behavior in the graphic. The importing parameter BEHAVIOR_LIST contains a list of behavior objects (CL_DRAGDROP). The behavior object contains one or more flavors and specifies whether the event is a copy event or a move event. You can specify a behavior object for each variant key.

IF_EVT_DRAGDROP

Definition

IF_EVT_DRAGDROP is the drag & drop event interface for controls encapsulated in a proxy using ABAP Objects.

Use

The application has to handle the following events in the order given below. They belong to IF_EVT_DRAGDROP:

- OBJ_FLAVOR_REQUESTED

  This event is sent by the target object's proxy to ask the application to resolve an ambiguity in the flavors - whether a text, a picture or a node is to be moved/copied. The application places the flavor in the parameter FLAVOR of the Drag & Drop object. For more information, see OBJ_FLAVOR_REQUESTED [Page 110]

- OBJ_DRAGGED

  This event is sent by the proxy containing the source object. The application must place the data object to be created for the source object in the parameter OBJECT of the Drag&Drop object.For more information, see OBJ_DRAGGED [Ext.]

- OBJ_DROPPED

  This event is sent by the proxy containing the target object. The application must analyze the data object (OBJECT) sent and insert it appropriately in the target object. For more information, see OBJ_DROPPED [Page 112]

- OBJ_DROP_COMPLETED

  This event is sent by the proxy containing the source object. The application can complete the drag and drop operation by carrying out tasks such as deleting data in the source object for a move operation. For more information, see OBJ_DROP_COMPLETED [Page 113]

Structure

The interface consists solely of the above-mentioned events.

OBJ_FLAVOR_REQUESTED

Definition

OBJ_FLAVOR_REQUESTED is an event in the interface IF_EVT_DRAGDROP. It is sent by the proxy containing the target object.

Use

The drag & drop service triggers this event if the drag object(s) and the drop object have more than one flavor in common. The appropriate handler routine in the application must then choose which flavor is to be used.

Structure

It has the following parameters:

- PROXY_OBJECT_DATA

  The parameter PROXY_OBJECT_DATA identifies the target object by means of a proxy-specific class (for the GFW an instance of the class CL_GFW_OBJECT).

- DRAG_DROP_OBJECT

  The parameter DRAG_DROP_OBJECT contains a reference to an instance of CL_DRAGDROPOBJECT – the so-called Drag & Drop object. The Drag & Drop object contains the data to be exchanged.

- FLAVORS

  The parameter FLAVORS contains a list of possible flavors.

OBJ_DRAGGED

Definition

OBJ_DRAGGED is an event in the interface IF_EVT_DRAGDROP.

Use

The event sends information as to the object(s) dragged. The application can use this information in its handler routine.

Structure

It has the following parameters:

- PROXY_OBJ_DATA_LIST

  The parameter PROXY_OBJ_DATA_LIST identifies the source object(s). It contains a list of proxy-specific objects.

- DRAG_DROP_OBJECT

  The parameter DRAG_DROP_OBJECT contains a reference to an instance of CL_DRAGDROPOBJECT.

OBJ_DROPPED

Definition

OBJ_DROPPED is an event in the interface IF_EVT_DRAGDROP. This event is sent by the proxy of the target object.

Use

The application's handler method can analyze the associated data object and insert it to match the target object.

Structure

The event has the following parameters:

- PROXY_OBJECT_DATA

  The parameter PROXY_OBJECT_DATA identifies the target object by means of a proxy-specific class.

- DRAG_DROP_OBJECT

  The parameter DRAG_DROP_OBJECT contains a reference to an instance of CL_DRAGDROPOBJECT.

OBJ_DROP_COMPLETED

Definition

OBJ_DROP_COMPLETED is an event in the interface IF_EVT_DRAGDROP. This event is sent by the proxy of the source object(s).

Use

The application's handler method can use information in the event parameters to complete any closing operations. In particular, the application has to adjust its data in connection with a move operation (deleting the source object and its data).

Structure

This event has the following parameters:

- PROXY_OBJECT_DATA_LIST

  The parameter PROXY_OBJ_DATA_LIST identifies the source object(s). It contains a list of proxy-specific objects.

- DRAG_DROP_OBJECT

  The parameter DRAG_DROP_OBJECT contains a reference to an instance of CL_DRAGDROPOBJECT.

Product Management

Definition

The graphics products organized by product management. All available graphics products that can be handled by the Graphical Framework are stored in product management along with some technical information and a priority.

The class for product management is CL_GFW_PRODUCTS.

Use

A graphics proxy can use product management to request a graphics product.

You can change the priority of the graphics products by maintaining the database table TGPM. It is based on the database table TGPN, maintained by SAP. You can maintain the settings in TGPN with transaction SM30, table/view V_TGPN.

You can find more information in the class documentation to CL_GFW_PRODUCTS in the class builder.

Structure

All products are entered in product management by graphics type and platform. The priority determines the search sequence. In the case of an ActiveX control the user should specify the ActiveX name from the registry as the *technical name*. The *usage* indicates the type of component (ActiveX, JavaBean or external program).

The wildcard * (asterisk) stands for any platform.

Example Programs

Definition

These programs are examples of the sorts of program that can be written using the Graphical Framework.

Use

The example programs can be used to display the sorts of graphics that can be generated using the Graphical Framework.

They should also help you to write your own graphics programs. You can use the example programs to check for programming errors or other errors (for example, concerning the installation or the platform used). If the reports function as required without any error messages you can assume that all the components of the Graphical Framework for the type of graphic concerned have been correctly installed. Therefore, if your new program does not function properly this is probably because of a programming error.

There is more detailed documentation for each program in the R/3 system. They all belong to the development class SGRB.

For more information on business graphics programs, see Business Graphics [Page 116].

Business Graphics

GFW_DEMO_PRES

This program gives an overview of the functionality of the Graphical Framework. It demonstrates the following:

Several graphics can use the same data container

Changes to the data container are visible in all the graphics

Interaction with graphics

Splitter and docking window

It can be used to display the following types of graphic:

In place in the screen

In the next screen

In a modeless window

In an external window/program

GFW_DEMO_PRES1

This program displays a Chart business graphic and is especially useful for demonstrating Customizing options.

DEMO_GFW_PRES_SHOW

This program displays a simple business graphic using the Graphical Framework´s function module.

DEMO_GFW_PRES_SHOW_MULT

This program displays two simple business graphics using the Graphical Framework´s function module. The graphics display the same data but with a different orientation - that is, the data is interpreted differently.

DEMO_GFW_SHOW1

This program shows how the Graphical Framework's function modules can be used with a splitter.

GFW_DEMO_HIER1

This program displays a hierarchy graphic using the Graphical Framework.

GFW_DEMO_HIER2

This program displays a hierarchy graphic in a splitter using the Graphical Framework.

GFW_DEMO_HIER3

This program displays both a hierarchy graphic and a presentation graphic in a splitter using the Graphical Framework.

Hierarchy Graphics

GFW_DEMO_HIER1

This report serves as a demonstration of a simple tree structure graphic.

GFW_DEMO_HIER2

The program provides an example of how to use interactive hierarchy/ structure graphics with the Graphical Framework (GFW).

GFW_DEMO_HIER3

The program provides an example of interactive hierarchy/structure graphics and business graphics using the Graphical Framework (GFW).

Tutorial for Graphical Framework

Contents

This document consists of the following sections:

Objectives of this Document

This document serves as an introduction ("Getting Started") for developers who want to use the Graphical Framework (GFW) with the ABAP objects interface.

If you only want to display graphics but do not need to use Customizing then the GFW function modules could well fulfill your requirements. In that case you do not need to know about the syntax of the GFW's ABAP objects interface. You should refer to the documentation on the Function Modules Supported [Page 207].

The aim of this tutorial is to create a simple application (SAPGFW1) with a graphic.

At the end of this document there is a section with possible reasons for errors for situations where the result of programming was not that which was intended.

To achieve a basic understanding of the GFW, in particular of the definition and interaction of the individual components, see the documentation on the Graphical Framework [Page 11].

Getting Started

The tutorial presupposes the following background knowledge:

- How to create a screen in R/3 with a very small menu (exit button)
- How the screen can be processed using PBO and PAI.
- ABAP objects syntax
- The following concepts current in object-oriented programming: *Interface*, *Class*, *Attribute*, *Method* and *Instance*. These concepts are briefly explained below.

After you have worked through this document you should be able to write your own applications using the GFW.

Overview: ABAP Objects, Interfaces, Classes, Attributes, Methods, Instances

This is a brief summary for those switching over from process-oriented programming to object-oriented programming.

The definitions should help you to understand the tutorial. They are not intended as an introduction to object-oriented programming!

Interface: Description of which subprograms (methods) can be called.

- Class: Collection of variables (attributes) and subprograms (methods) – if a class (for example, the class CL_CLASSNAME ...) provides an interface ("implements the interface"), this means that it has at least those methods defined in the interface (marked as "public").
- Instance: Object belonging to a class. Every instance of a class offers users all the methods of its class. The instance attributes can only be used by this instance; other instances of the same class have their own instance attributes. However, the local attributes in methods are available for every instance. You can only access methods and (public) attributes of a class via instances (for example, DATA C_REF TYPE REF TO CL_CLASSNAME.).
- Attribute: Variable; an attribute is local for all methods in the class for which it is defined; programs/methods belonging to another class can only access attributes marked "public".
- MYVAL = C_REF->PUBLICVAL.
- Method: Subroutines; can be called in all others methods belonging to the class; programs/methods belonging to another class can only call methods marked "public":
- CALL METHOD C_REF->INIT.
- Class Attribute, Class Method: These are exceptions to the rule that every instance has its own attributes: class attributes and class methods are called without an instance and are the same for all instances, that is attributes and static variables have the same contents. => is used to call them, for example:

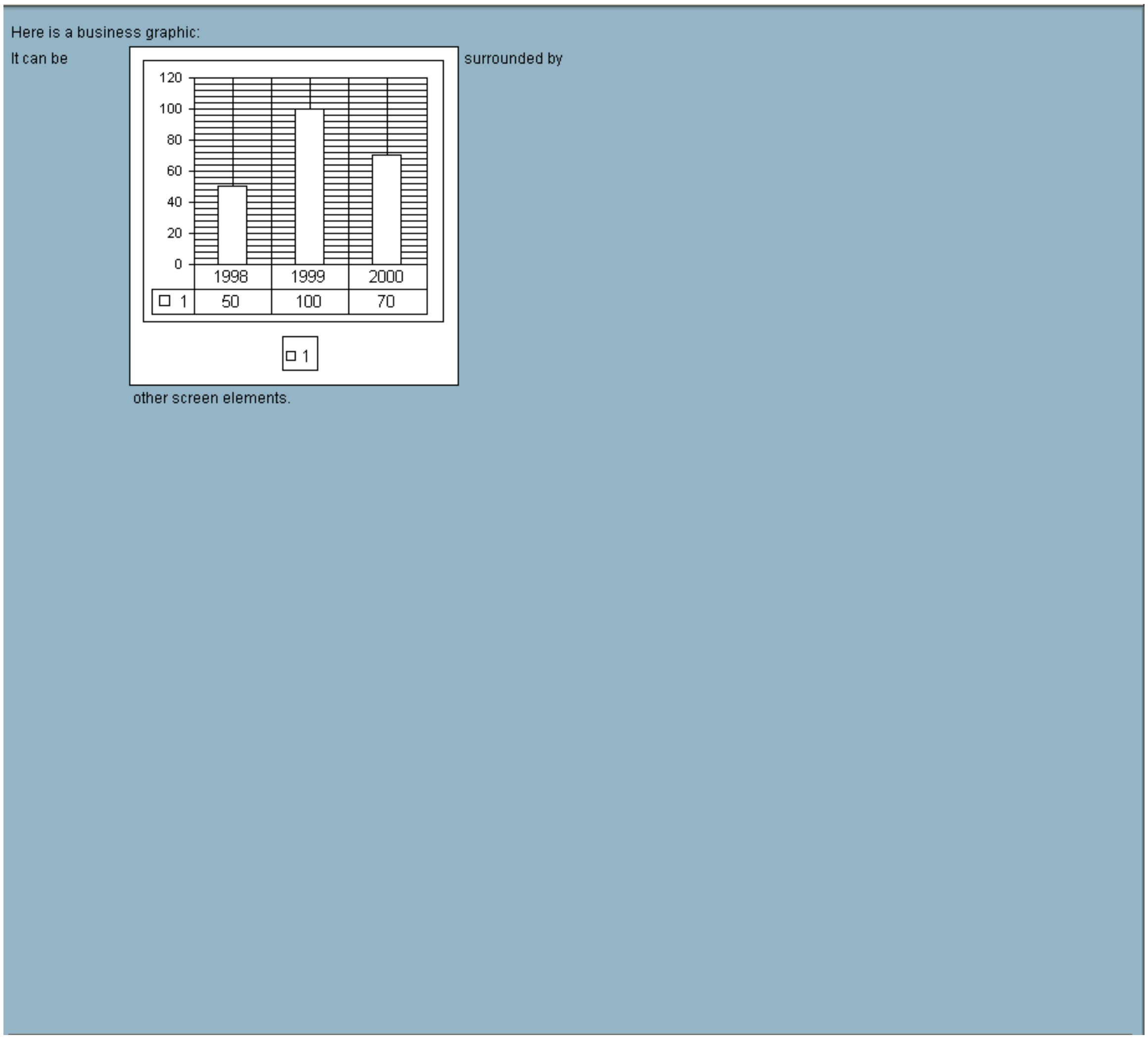
```
CALL METHOD CL_CLASSNAME=>ACTIVATE.
```

You can find clear examples of the syntax in the mail folder OOROLLOUT in the document "Status information of OO issues ..."

Tutorial for Graphical Framework

Tutorial

Our objective is to create a report with a window containing a business graphic. There should be text placed around the graphic.



Here is a business graphic:

It can be

surrounded by

other screen elements.

Sample of a Graphic created using the Graphical Framework

The graphic should be visible in the window directly after the application has been called.

Step 1: Creating the Report with a Screen

Create your example report "My Report" with the screen "100" with a menu and texts as usual.

Place a custom control element (call it GRAPHIC) where you want the graphic to appear.

Tip: It is easy to create a custom control element using the graphical fullscreen editor. Choose the custom control from the toolbox and draw a box on the screen using the mouse. When you release the mouse button you can specify the name of the custom control.

Do not forget to save and activate the screen and menu!

Test your application. Does it correspond to the initial screen in this section except that is does not have a graphic? If necessary, insert the program call (`CALL SCREEN “100“`).

Step 2: Creating the Framework for the GFW

Now program the framework of the GFW.

To process return values from the GFW the application needs to know the data types used:

```
TYPE-POOLS gfw.
```

To use the global classes of the GFW you need to write the following commands:

```
CLASS cl_gui_cfw DEFINITION LOAD.

CLASS cl_gfw_mux DEFINITION LOAD.
```

You have to include the required data container as a local class:

```
* data container

INCLUDE gfw_dc_pres.
```

Caution: These Includes contain executable ABAP so they have to be added AFTER the screen has been called or the command "START-OF-SELECTION" must stand before the screen call.

All GFW methods send return codes. Therefore you have to define a variable in your program to receive the return codes:

```
DATA: retval type symsgno.
```

Insert the multiplexer call in PBO:

```
**** activate mux (handling of external graphics and synchronizations)

 CALL METHOD cl_gfw_mux=>activate IMPORTING retval = retval.
```

Error analyses for the return codes can consist in simply displaying the corresponding message:

```
 IF retval <> cl_gfw=>ok.

   CALL METHOD cl_gfw=>show_msg EXPORTING msgno = retval.

 ENDIF.
```

Insert the call of the dispatcher for events in PAI:

```
ok_code = sy-ucomm.

* activate event analysis of object-oriented control framework

  CALL METHOD cl_gui_cfw=>dispatch.
```

Step 3: Providing Data

What data should your graphic display?

To supply the data you must create an instance of the data container for business graphics (LCL_DC_PG). The object is accessed using an interface variable. Your application must also recognize an ID which it uses in future to register with the data container.

You must define the following variables:

```
DATA: dc_inst TYPE REF TO lcl_dc_pres,        "// dc instance
      dc_manage TYPE REF TO if_dc_management, "// dc interface handle
      my_id_at_dc TYPE i, retval TYPE symsgno."// dc id
```

Create the data container in PBO before the multiplexer is activated:

```
retval = cl_gfw=>ok.
  IF dc_manage IS INITIAL.
*   create and initialize data container
    CREATE OBJECT dc_inst.
    IF sy-subrc <> 0.
      CLEAR dc_inst.
    ELSE. "//create dc ok
      dc_manage = dc_inst.
      CALL METHOD dc_manage->init IMPORTING id = my_id_at_dc
                                            retval = retval.
      IF retval <> cl_gfw=>ok.
        CLEAR dc_inst.
        CLEAR dc_manage.
      ELSE.
*       PLACEHOLDER: Fill with data
      ENDIF.
    ENDIF. "// create dc ok
  ENDIF. "//dc_manage IS INITIAL.

  IF retval <> cl_gfw=>ok.
    CALL METHOD cl_gfw=>show_msg EXPORTING msgno = retval.
  ENDIF.
```

An empty data container now exists.

You must tell the data container to distribute any existing changes so that your graphic automatically receives data from the data container (especially changes). This is why you should insert an interface method call before the multiplexer is activated:

```
* **** distribute changes (to all registered graphics proxies)
  IF NOT dc_manage IS INITIAL.
    CALL METHOD dc_manage->distribute_changes
                          IMPORTING retval = retval.
    IF retval <> cl_gfw=>ok.
      CALL METHOD cl_gfw=>show_msg EXPORTING msgno = retval.
    ENDIF.
  ENDIF.
```

Filling the data container with data can be a very large task so it cannot occur directly in PBO but instead in an application method (for example, FILL_DC for the instance THIS_APPLICATION).

```
*       fill DC with application data
        IF NOT this_application IS INITIAL.
         CALL METHOD this_application->fill_dc
                                  EXPORTING dc_manage = dc_manage
                                  IMPORTING retval = retval.
        ENDIF.
```

Enter the definition of the application class before PBO:

```
* application code
  CLASS lcl_this_application DEFINITION.
    PUBLIC SECTION.
      METHODS: fill_dc
                      IMPORTING dc_manage TYPE REF TO if_dc_management
                      EXPORTING retval TYPE symsgno.
  ENDCLASS.
  DATA this_application TYPE REF TO lcl_this_application.
```

The implementation of the application class with the frame for the fill method can be written at the end of the report:

```
* --------------------------------------------------------------------
-
* APPLICATION CLASS implementation
* --------------------------------------------------------------------
-
CLASS lcl_this_application IMPLEMENTATION.
METHOD fill_dc.
  retval = cl_gfw=>ok.
ENDMETHOD. "//fill_dc
```

Implement the method FILL_DC with the help of the data container method SET_VALUE or SET_OBJ_VALUES.

Hint: The data container attributes 'OBJID', 'X_VAL' and 'Y_VAL' must be filled with data to provide a meaningful display. To provide data for the example GRPID is also filled.

Is your program free of syntax errors?

Step 4: Creating the Graphics Proxy

Once you have programmed the report with the screen and the framework of the GFW and supplied the data the graphic has to be generated.

For this purpose an instance of a business graphics proxy is created. The graphics proxy is accessed (for example, for parametrization) using an interface reference variable.

```
DATA: gp_inst TYPE REF TO cl_gui_gp_pres.        "// gp instance
```

- Create and initialize the graphics proxy in PBO.

Note: Use SAP's BUSG graphic (cl_gui_gp_pres=>co_prod_sapocx) as the graphics product.

Note: Create a container object using the name of the subscreen in the dynpro that you defined. Specify this container object as a placeholder (parameter PARENT).

```
DATA: CUSTOM_CONTAINER TYPE REF TO CL_GUI_CUSTOM_CONTAINER.
      CREATE OBJECT CUSTOM_CONTAINER
                    EXPORTING CONTAINER_NAME = '...'.
```

- Tell the graphics proxy the names of the attributes in the data container.

Note: Use GP_INST->SET_DC_NAMES, a method in CL_GUI_GP_PRES.

- If up to this point no error has occurred then the graphics proxy (in PBO as well) can be activated. Otherwise you should clear the proxy and the reference:

```
IF retval <> cl_gfw=>ok.
      CALL METHOD gp_inst->if_graphic_proxy~free
           IMPORTING retval = retval2. "//ignore return value(keep 1st)
      CLEAR gp_inst.
    ELSE.
      CALL METHOD gp_inst->if_graphic_proxy~activate
                                 IMPORTING retval = retval.
    ENDIF.
    IF retval <> cl_gfw=>ok.
      CALL METHOD cl_gfw=>show_msg EXPORTING msgno = retval.
    ENDIF.
```

Now that you have finished you should test your program. Does it give the right result?

Yes? Congratulations, now go to step 5.

No? Have you instantiated your application class (before "THIS_APPLICATION->FILL_DC") ?

```
      * check existence of application instance
        IF this_application IS INITIAL.
          CREATE OBJECT this_application.
        ENDIF.
```

Solution: You can find the solution in the report GFW_PROG_TUTORIAL in B20.

Step 5: Suggestions for Self-Study

You can extend your program yourself. For example, try the following:

- Incorporating a button for activating/deactivating the graphic (graphics proxy methods DEACTIVATE/ACTIVATE)
- Dynamic placing of the graphic (without custom control)
- Using another graphics product (for example, PROD_ID = CL_GUI_GP_PRES=>CO_PROD_CHART)
- Making the graphic interactive

Reasons for Errors

Description of situation	Possible reasons for errors	Solution
All calls end with CL_GFW=>OK but the graphic is still not displayed, not even example data.	CL_GFW_MUX=>ACTIVATE-Call is missing in PBO (the graphics proxies are optimized, they do not carry out any flushes; it is the multiplexer that sends the front end a flush).	In PBO: Complete call of CL_GFW_MUX=>ACTIVATE
The graphic ignored my changes; no error is displayed	The graphics product used/the graphics proxy used does not support the desired operation.	1) Ensure that programming is correct by testing with another product 2) Check the capability of the desired product (product catalog, description of GP used) 3) Choose another product or development request or error message
After an Include strange errors occur	The ABAP interpreter cannot find the start of the program.	Extend START-OF-SELECTION before the first call screen or the actual report source.

Printing Graphics

Use

The current view of a window can always be printed using the clipboard. The Windows function ALT+PRINT copies the contents visible in the current window into the clipboard. Some graphics products allow the user to copy the current graphic into the clipboard.

The clipboard can then be inserted in a text editing program and printed.

The method PRINT in IF_GRAPHIC_PROXY can be used in situations that are more direct and more complex.

Objects used in the Graphical Framework

Definition

The Graphical Framework consists of the following objects (some of them in the class browser):

- Error class APPLG
- All interface definitions
- Multiplexer (CL_GFW_MUX)
- Superclass for graphics proxy (CL_GUI_GP)
- Superclass for data container (CL_GFW_DC)
- Data container wizard (Report GFW_DCWIZARD)
- For business graphics:

  Data container for business graphics (LCL_DC_PRES)

  Graphics proxy with ActiveX and JavaBean (CL_GUI_GP_PRES)
- For structure graphics:

  Data container for structure graphics (LCL_DC_HIER)

  Structure graphics graphics proxy with ActiveX and JavaBean (CL_GUI_GP_HIER)

Error Handling

Use

The error handling function can be used by programmers to analyze programming errors. It is not intended for use by people using the Graphical Framework.

Features

The class CL_GFW defines all the error constants for the Graphical Framework and provides some routines to handle errors.

Error numbers and error texts are stored in the error class APPLG created for the Graphical Framework (=> autom. translation). Constants in the global class CL_GFW are used in the program for the error numbers.

Activities

To display the message log call the method SHOW_MSG_PROTOCOL in CL_GFW.

To delete any existing messages from the log call the method CLEAR_MSG_PROTOCOL in CL_GFW.

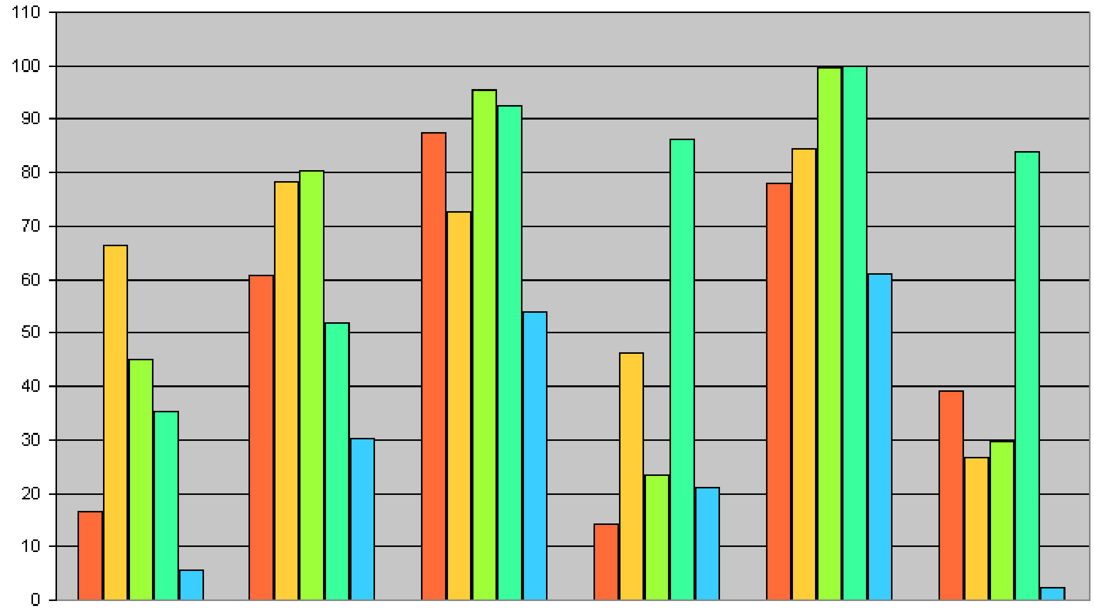
Every method returns the number of the first error to occur.

Business Graphics

Displaying data on a worksheet is often not the best way to present it to users. A page full of numbers, even if formatted attractively, can be hard to understand and perhaps a little boring. To present business data more effectively and to allow users to change data interactively business graphics are often used to display data in the form of a chart.

A chart is a visual representation of selected data. A well-designed chart draws the reader´s attention by illustrating significant relationships between numbers.

A Simple Business Graphic



User Documentation

Chart

The objective of the Chart user documentation is to provide you with detailed procedures for working with this business graphics product. The procedures contain no information on programming Chart or other business graphics products.

Although the information is concrete and detailed it is assumed that you are familiar with basic handling functions dealt with in Getting Started. For example, basic functions such as selecting a graphic element or making printer settings are not covered here.

Chart is tailored to the requirements of users of business graphics. A familiarity with EXCEL is a great advantage but not a prerequisite for using Chart.

For more information, see Chart [Page 136].

SAP BUSG

The graphics product SAP BUSG is available both as an external program graphic and as an ActiveX control.

For more information, see SAP Business Graphics [Ext.].

Chart

Definition

An R/3 application component can use the Chart business graphic to display data in a graphic in the SAP front end.

The Chart business graphic is a graphics component with a programming interface and a user interface.

Use

You can use Chart as a 32-bit graphics component with the following operating systems:

- Windows 95
- Windows NT
- Windows NT 4.0

Chart offers a wide range of chart types , for example vertical bars or pie charts. You can use time scale charts with one or more time scales for milestone trend analyses and control charts. For more information, see Chart Types [Page 145].

You can work flexibly with the charts using the property pages. You can format every chart element individually. The texts in the property pages can be configured by the calling application. For more information, see Property Pages [Page 143].

You can carry out some functions by manipulating a graphical element with the mouse, for example changing the value of a data point.

When you make changes in the chart, for example selecting objects or changing values these changes are sent to the calling function using the interface. The application can use this interface to monitor what is happening in the chart.

Integration

Chart is integrated in the SAP front end as an ActiveX control and called up using the R/3 application component.

You can call only those settings/functions in Chart that are supported by the calling R/3 application.

Making Settings for Printing a Chart

Use

You can print the chart, specify the page setup for printing and select a print preview.

Activities

Printing a chart

1. Select the chart using the right-hand mouse button.
2. Choose *Print.*
3. Choose *OK.*

Specifying the page setup for printing

Carry out the first two steps above and then choose *Page Setup.*

You can make a variety of settings by choosing *Page, Page Borders* and *Options.*

Selecting a print preview

1. Select the chart using the right-mouse button.
2. Choose *Print Preview.*

Formatting Chart Area and Plot Area

Use

You can change the border attributes and fill attributes of the chart area.

You can change the border attributes, fill attributes and line attributes of the plot area.

Procedure

Changing border attributes and fill attributes of the chart area

1. Double-click the chart area outside the plot area.
2. Change the attributes as required.

Changing border attributes and fill attributes of the plot area

1. Double-click the plot area outside the data series.
2. Choose *Pattern*.
3. Change the attributes as required.

Changing line attributes of the plot area

1. Double-click on the plot area outside the data series.
2. Choose *Line*.
3. Change the attributes as required.

Copying Charts

Procedure

1. Select the chart using the right-hand mouse button.
2. Choose *Copy*.

Result

The chart is saved in the clipboard in the following formats:

- ASCII (unformatted text)
- BMP (bitmap)
- EMF (enhanced metafile)

You can insert it into other documents.

Changing Size of Charts

Procedure

1. Select the plot area.
2. Place the cursor on a selection point.

   The cursor changes into a double arrow.
3. Depress the left-hand mouse button.

   The cursor changes into a cross.
4. Drag the chart keeping the left-hand mouse button depressed until the chart is the required size.

Result

When you release the mouse button the chart is resized.

Chart Element

Definition

This is an item in a chart.

There are two types of chart element:

- Graphical chart elements, such as axes
- Text chart elements, such as headings for axes

Use

You can specify the elementary chart elements to be displayed in the chart, for example the axes, using the chart options on the property pages. Data points or data series are automatically inserted in the chart. Chart elements such as the plot area appear in the chart automatically.

You can use the property pages to format every chart element individually, for example changing the color. You can use chart element-specific formatting functions to insert additional chart elements depending on the chart type, for example data labels for a data series.

Selecting Chart Elements

Procedure

Selecting a chart element

Select the chart element using the left-hand mouse button.

Several handles are displayed. Handles are small black boxes that appear around the perimeter of a selected object, indicating that you can move, resize, copy or delete the object.

Selecting an individual data point or an individual data heading in a data series.

Click twice in succession using the left-hand mouse button on the chart element (not a double-click!).

The first click selects all similar elements in the data series, for example all data points.

The second click makes all the selections disappear apart from the selection of the chosen chart element, for example the data point.

Property Pages

Use

You can use property pages to do the following:

- Call up all the settings functions for the graphic
- Reverse settings
- Copy the chart

Changing settings

For example, you can select a chart type such as lines or columns.

Reversing settings

You use this function to reverse the last setting. You can reverse all the settings you have made since you called up the chart one after the other.

Copying

You use this function to copy the chart onto the clipboard. The chart is saved on the clipboard in the following formats:

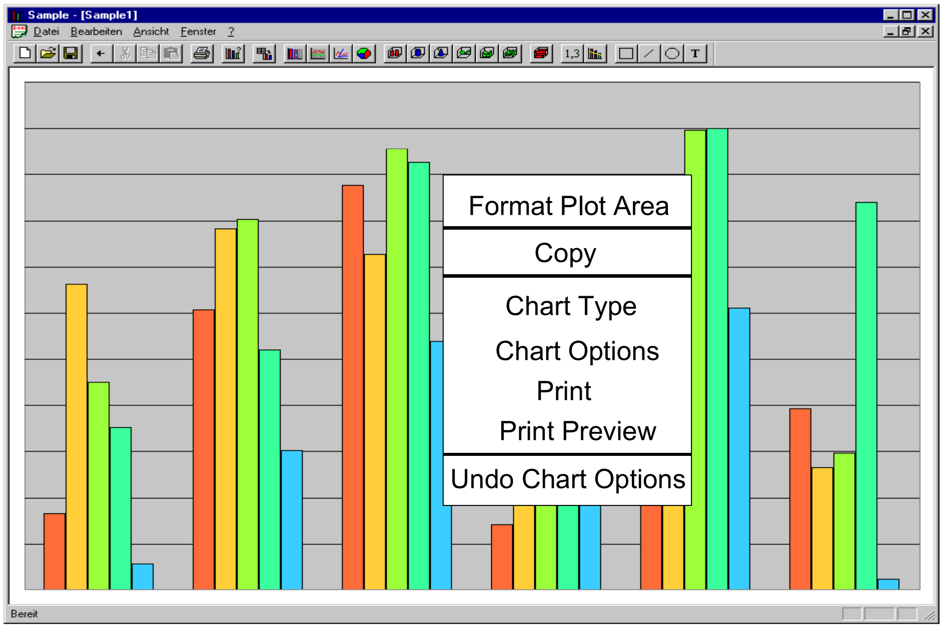
- ASCII (unformatted text)
- BMP (bitmap)
- EMF (enhanced metafile)

You can insert the copied chart into other documents.

The functions offered by the property pages depend on the context, that is, the selected chart element and the chart type determine which of the formatting functions are offered on the property pages.

You can only use functions that are allowed in the application that calls the graphic.

Property Pages in Chart



Activities

You can call up the property pages by selecting a chart element using the right-hand mouse button.

If you want to make settings for an individual data point or for an individual data heading you have to select them before you call up the property pages.

If you double-click on a chart element using the left-hand mouse button you go directly to the dialog box for formatting the chart element.

Chart Types

Definition

The chart type determines how your data is represented in the graphic. You can select a chart type and a chart subtype.

Use

The following chart types are available:

- Column chart (clustered, stacked and 100% stacked, 3-D visual effect, cylinder, cone and pyramid).

  For more information, see Column Charts [Page 147].
- Bar chart (clustered, stacked and 100% stacked, cylinder, cone and pyramid).

  For more information, see Bar Charts [Page 148].
- Line chart

  For more information, see Line Charts [Page 149].
- Pie chart.

  For more information, see Pie Charts [Page 150].
- XY scatter chart (dots can also be joined with lines).

  For more information, see XY (Scatter) Charts [Page 151].
- Area chart (clustered, stacked and 100% stacked)

  For more information, see Area Charts [Page 152].
- Doughnut chart.

  For more information, see Doughnut Charts [Page 153].
- Radar chart

  For more information, see Radar Charts [Page 154].
- Surface chart

  For more information, see Surface Charts [Page 155].
- Cone, cylinder and pyramid charts

  For more information, see Cone, Cylinder and Pyramid Charts [Page 156].
- Milestone charts and step charts

  For more information, see Milestone Charts [Page 157] or Step Charts [Page 158].
- Histograms (allow you to display a standard distribution as well)

  For more information, see Histograms [Page 159].

You can specify the chart type using the property pages as described for the individual chart types.

Selecting a Different Chart Type/Subtype

Use

For most charts you can change the chart type either for a data series or for the entire chart. For bubble charts you can change only the type of the entire chart.

Procedure

1. To change the chart type of a data series select the data series using the right-hand mouse button. To change the chart type of the entire chart select the chart area using the right-hand mouse button.
2. Choose *Chart type* and then select the chart type and the chart subtype.

   If you set the indicator *Assign to all data rows* the chart type is changed for the entire chart, even if a single data series is selected.

Column Charts

Definition

A column chart displays data in the form of vertical columns.

Use

You can use column charts to illustrate comparisons between categories. You can use the XY scatter charts to display changes through time.

Structure

Categories are organized horizontally and values vertically.

Stacked column charts show the relationship of individual items to the whole.

Bar Charts

Definition

A bar chart displays data in the form of horizontal bars.

Use

A bar chart illustrates comparisons between individual items.

Structure

Categories are organized vertically and values horizontally to focus on comparing values and to place less emphasis on time.

Stacked bar charts show the relationship of individual items to the whole.

Line Charts

Definition

A line chart displays data in the form of lines.

Use

A typical use would be to plot categories (for example, months) along the X axis and values (for example, revenue in dollars) along the Y axis.

Structure

Line charts display data at equal intervals.

Pie Charts

Definition

A pie chart shows the size of elements that make up a data series proportionate to the sum of the items.

Use

It shows only one data series and is useful when you want to emphasize a significant element.

Structure

You can select subtypes that separate the segments from one another.

XY (Scatter) Charts

Definition

An XY (scatter) chart either shows the relationship between the numeric values in several data series or plots two groups of numbers as one series of XY coordinates.

Use

It shows uneven intervals (or clusters) of data and is commonly used for scientific data.

The X axis must be set as a time axis in contrast to other chart types, which display categories along the X axis.

Area Charts

Definition

An area chart corresponds to the line chart type only the area under the line for a data series is filled.

Use

By displaying the sum of plotted values an area chart can show the relationship of parts to a whole.

Doughnut Charts

Definition

A doughnut chart looks similar to a pie chart but it can contain more than one data series.

Use

Like a pie chart a doughnut chart shows the relationship of parts to a whole.

Structure

Each ring of the doughnut chart represents a data series.

One subtype shows the parts of a ring of the doughnut separately, the other shows them as one continuous ring.

Radar Charts

Definition

In a radar chart each category has its own value axis radiating from the center point. Lines connect all the values in the same series.

Use

You can use radar charts to compare data series: the data series with the highest values covers the largest area.

Surface Charts

Definition

A surface chart plots sets of values in the form of a surface. It is basically a line chart where the points for a data series for a category are displayed behind one another.

Structure

As in a topographic map, colors and patterns indicate areas that are in the same range of values.

Cone, Cylinder and Pyramid Charts

Definition

Each of the three chart types has several subtypes some with vertical cones/cylinders/pyramids, some with horizontal cones/cylinders/pyramids.

Milestone Charts

Definition

Milestone charts plot two sets of values against each other. There are two subtypes.

Use

You can use milestone charts to compare planned values with actual values for project analysis.

Step Charts

Definition

A step chart plots values in the form of steps rather than curves or sloped lines. There are two subtypes.

Use

Step charts can be used to show loads for a resource material over a period of time, available capacity at a work center over a period of time or stepped fixed costs.

The X axis must be set as a time axis in contrast to other chart types, which display categories along the X axis.

Histograms

Definition

A histogram displays categories/value ranges along the X axis and their values on the Y axis. There are two subtypes. One displays a transparent normal distribution curve over the histogram.

Use

You can use histograms to compare statistical values such as the number of measurement readings within a certain range.

Displaying Standard Distribution

Use

You can display a transparent normal distribution curve over the histogram along with a title.

Prerequisites

You must have selected the chart type *Histogram* and the appropriate chart subtype.

Procedure

Specifying the mean value of the curve

1. Double-click the standard distribution curve.
2. Choose *Standard distribution*.
3. Specify the mean value.

Specifying the standard deviation

1. Carry out steps 1 and 2 as in the first procedure.
2. Specify the standard deviation.

Specifying a title for the curve

1. Carry out steps 1 and 2 as in the first procedure.
2. Deselect the indicator *Automatic*.
3. Specify a title under *Distribution name*.
4. Select the indicator *Display name on chart*.

Formatting the curve

1. Double-click the standard distribution curve.
2. Choose *Pattern*.
3. Specify the required attributes.

Axes and Gridlines

Definition

The axes consist of the vertical Y axis and the horizontal X axis.

Gridlines are vertical and horizontal guidelines that appear behind a chart.

Use

The Y axis shows the values for the data series in the chart. The X axis shows either values or categories for the data series. There is a wide range of functions for the axes including the following:

General axis settings

Time scale settings (for more information, see Making Time Scale Settings [Page 169])

Gridline settings (for more information, see Displaying or Hiding Gridlines in a Chart [Page 180])

The gridlines make it easier to read data from the graphic.

Formatting Axes

Use

You can individually format each axis in a chart.

The chart type determines the formatting functions you can use for the axes. The formatting functions include the following:

- Specifying patterns (colors, thickness of lines and so on)
- Formatting tick marks
- Specifying number formats
- Specifying fonts
- Specifying alignments of axes
- Selecting value ranges

  For the value range you can specify values for an upper limit, a lower limit and an optimal value. Within the chart you can change these values as you change values for a data point - by moving the lines.

Prerequisites

You specify the primary axes to be displayed in the chart using the chart options on the property pages.

You can also specify that a secondary axis is to be displayed. This is possible even if there is no data series for this axis.

Chart assigns an axis to a data series automatically; this is not something that the user has to do.

Activities

You can call up the formatting functions for an axis by double-clicking it.

Displaying Tick Marks

Use

You can display major and/or minor tick marks inside, outside or across an axis.

Procedure

1. Double-click an axis.
2. Choose *Line*
3. Specify the *Major/Minor tick mark type*:

    None: No tick marks

    Outside: Tick marks outside the line

    Inside: Tick marks inside the line

    Cross: Tick marks both inside and outside the line

Displaying or Hiding Axes in a Chart

Use

You can specify whether the axes along with their labels and tick marks are to be displayed or hidden.

Pie charts and doughnut charts do not have axes.

Procedure

1. Select the chart using the right-hand mouse button.
2. Choose *Chart options*.
3. Choose *Axes.*
4. Select the indicators for the axes you want to display. Clear the indicators for the axes you want to hide.

Changing Number Formats for Axes

Prerequisites

The relevant axis must be visible.

Procedure

1. Double-click the axis for which you want to change the number format.
2. Choose *Number.*
3. Select the required *Category* (for example All, Date, Time)
4. Select the required *Number format* within that category.

Defining the Intersection of Axes

Use

You can specify the point of intersection of the two axes. You can make this specification for both axes as described below.

Prerequisites

The scale style must be *Time* (see Changing the Scale Style of an Axis [Page 168]).

Procedure

Defining the intersection point type for the X axis

1. Double-click the X-axis.
2. Choose *Scale*.
3. Specify the required type of intersection:

   Intersection at value: Enter the required value in the field *Value axis crosses at category number*.

   Intersection at minimum: Clear the indicator *Value axis crosses at maximum category*.

   Intersection at maximum: Select the indicator *Value axis crosses at maximum category*.

Defining the intersection point type for the Y axis

1. Double-click the Y-axis
2. Choose *Scale*.
3. Specify the required type of intersection:

   Intersection calculated automatically: Set the indicator to the left of *Category axis crosses at:*

   Intersection at value: Clear the indicator to the left of *Category axis crosses at* and enter the required value in the field *Category axis crosses at:*

   Intersection at minimum: Clear the indicator *Category axis crosses at maximum scale*

   Intersection at maximum: Select the indicator: *Category axis crosses at maximum scale.*

Specifying a value for the intersection point for the category axis

1. Double-click the X-axis.
2. Choose *Scale*.
3. Specify the value in *Value axis crosses at category number:*

Specifying a value for the intersection point for the value axis

1. Double-click on the Y-axis.
2. Choose *Scale.*
3. Make sure that the scale type is not *Time.*
4. Enter the required value in the field *Category axis crosses at.*

Specifying a time for the intersection point for a value axis

1. Double-click the Y-axis.
2. Choose *Scale.*
3. Select the scale type *Time.*
4. Enter the required value in the field *Category axis crosses at.*

Specifying whether the Y-axis crosses the X-axis between labels or at labels

1. Double-click the X-axis.
2. Choose *Scale.*
3. To specify that the value axis is to cross the category axis between category labels select the checkbox *Value axis crosses between categories*. To specify that the value axis is to cross the category axis at the category labels clear the checkbox.

    If the indicator is selected then the major tick marks are placed between the labels. If the indicator is cleared the major tick marks are placed directly under the labels.

Changing the Scale Style of an Axis

Use

You can specify one of the following three scale styles:

- Linear
- Logarithmic
- Time

The scale style determines what other formatting options are available for a values axis.

Prerequisites

You can change the scale style of a values axis but not of a categories axis. This is regardless of whether it is the X axis or the Y axis.

Procedure

1. Double-click the axis.
2. Choose *Scale*.
3. Select the scale style required.

Making Time Scale Settings

Use

There is a range of settings available when you have specified the scale style *Time* that are not available for the scale styles *Linear* and *Logarithmic.*

Prerequisites

The axis must be a value axis rather than a categories axis. The scale style for the value axis must be set to *Time*.

Features

The following formatting functions are available for time scales:

Specifying the gap between the time scale and the graphic

For more information, see Specifying Gap Between Axis and Graphic [Page 170]

Specifying a time interval value or format

For more information, see Defining a Time Interval Value [Page 174] and Defining a Time Interval Format

[Page 175]Specifying the width of a day

For more information, see Specifying the Width of a Day [Page 171]

Specifying the date at which the view starts

For more information, see Specifying the Date at which the View Starts [Page 172]

Specifying fill attributes for the time axis

For more information, see Specifying Fill Attributes for Time Axis [Page 173]

Specifying Gap Between Axis and Graphic

Prerequisites

The scale style for the value axis must be set to *Time*.

Procedure

1. Double-click on the value axis
2. Choose *Scale*
3. Set the required value for the *Gap width*

Result

If the *Gap width* is set to 0 the time axis is directly next to the graphic.

If the *Gap width* is greater than 0 the time axis is the specified number of 1/100th of a mm away from the graphic.

Specifying the Width of a Day

Use

You can specify the display width for a day on a value axis.

Prerequisites

You can specify the width of a day only for a value axis, not for a category axis. The value for the width of a day is taken into account only if the indicator under the column *Auto* and to the left of *Width of a day* is cleared.

Procedure

4. Double-click on the axis.
5. Choose *Scale*.
6. Specify the required value in the field to the right of *Width of a day*.

Result

You specify the width of a day in 1/100th of a mm.

Specifying the Date at which the View Starts

Prerequisites

The scale style for the value axis must be set to *Time*.

Procedure

7. Double-click on the value axis
8. Choose *Scale*
9. Set the required value for *View starts at.*

Specifying Fill Attributes for Time Axis

Use

You can specify the following fill attributes for the time axis:

Color

Pattern

Texture

Gradient

Transparency

Procedure

10. Double-click on the value axis
11. Choose *Pattern*
12. Make the settings you require.

Defining a Time Interval Value

Procedure

1. Double-click on an axis.
2. Choose *Scale*.
3. Clear the indicator to the right of *Major unit*.
4. Select *Time* for the *Value axis scale*.
5. Select a time interval value in the second field to the right of *Major unit*.

Defining a Time Interval Format

Procedure

6. Select the axis using the right-hand mouse button.
7. Choose *Format axis.*
8. Choose *Scale*
9. Select *Time* for the *Value axis scale*.
10. Clear the indicator to the right of *Major unit*.
11. Select a time interval format in the field to the right of *Format*.

Defining an Interval for the Value Axis

Use

You can define a major interval and/or a minor interval for a value axis.

Prerequisites

You can define an interval only for a value axis, not for a category axis.

The indicators to the left of *Major unit* and/or *Minor unit* have been cleared.

Procedure

Specifying the major/minor unit automatically:

12. Double-click the axis.
13. Choose *Scale*.
14. Select the indicator to the left of *Major unit* and/or *Minor unit*.

Specifying the major/minor unit manually:

1. Carry out steps 1 and 2 above.
2. Clear the indicator to the left of *Major unit* and/or *Minor unit* respectively.
3. Specify the major/minor interval in the fields *Major unit*/*Minor unit.*

Setting Value Ranges

Use

You can set minimum, optimum and maximum value ranges, specify whether they are to be displayed in the graphic and give them a title.

Prerequisites

If the scale style is *Time* the values will be expressed as dates. If the scale style is *Linear* or *Logarithmic* the values will be expressed as linear or logarithmic values respectively.

Procedure

Specifying a value range

1. Select the values axis using the right-hand mouse button.
2. Choose *Add Value Range*.
3. Specify a minimum, optimum and/or maximum value in the field(s) to the right of *Value*.

Specifying whether a value is to be displayed

1. Carry out steps 1 and 2 above.
2. To show the value range on the screen clear the indicator *Invisible*. To hide the value range set the indicator *Invisible.*

Specifying a title for a value

1. Carry out steps 1 and 2 as in the first procedure.
2. Specify a title for one or more of the value ranges.

Formatting Value Ranges

Use

You can format the titles for the minimum, optimum and maximum values as well as the lines representing them.

Prerequisites

You must already have specified values and titles for the values ranges.

Procedure

Formatting the titles for values ranges

Double-click the value range title you want to change:

To change the title choose *Value Range* and make the required settings.

To change the fill/border attributes choose *Pattern* and make the required settings.

To change the font choose *Font* and make the required settings.

To change the alignment choose *Alignment* and select the required alignment.

Formatting the value ranges

Double-click the relevant value range line:

To change the value range choose *Value Range* and make the required settings.

To change the fill/border attributes choose *Pattern* and make the required settings.

To change the line attributes choose *Line* and make the required settings.

Changing Minimum/Maximum Values

Use

You can set the minimum and maximum values for a value axis either manually or automatically.

With the scale type *Time* you can specify both the date and the time of day.

Prerequisites

You can change these values only for a value axis, not for a category axis.

If you want to set the values manually you must make sure that the indicator for setting the values automatically is cleared.

Procedure

Setting the values automatically

1. Double-click the value axis.
2. Choose *Scale.*
3. To set the minimum value automatically set the indicator in the column headed *Auto* to the left of *Minimum.* To set the maximum value automatically select the indicator in the column headed *Auto* to the left of *Maximum.*

Setting the values manually

1. Carry out steps 1 and 2 above
2. To set the values manually clear the indicator to the left of *Minimum* and/or *Maximum* respectively.
3. Enter a value in the field to the right of *Minimum* and/or *Maximum* respectively.

   With the scale type *Time* you can change the date by selecting the icon to the left of the field *Minimum*/*Maximum*. To change the time of day select the icon to the right of the field *Minimum*/*Maximum*.

Displaying or Hiding Gridlines in a Chart

Use

You can display major gridlines and/or minor gridlines for both the X axis and the Y axis.

Prerequisites

You must select a chart type other than pie charts or doughnut charts.

Procedure

1. Select the chart using the right-hand mouse button.
2. Choose *Chart options*.
3. Set the indicators for the gridlines you want to display. Clear the indicators for the gridlines you want to hide.

Data Series and Data Points

Definition

A data series is a range of related data points in a chart such as bars, columns, or pie slices.

A data point is a point in a data series. It is denoted by a data marker, which is a chart object such as a circle, dot or square.

Use

You can carry out the following procedures relating to data series and data points:

Reversing Order of Categories or Values [Page 182]

Changing Values of Data Points [Page 183]

Formatting Data Series and Data Points [Page 184]

Smoothing the Angles of Line Charts [Page 185]

Reversing Order of Categories or Values

Use

You can reverse the plotting order of categories or values for most charts. You cannot reverse the plotting order of values in a radar chart.

Procedure

1. Double-click the X or Y axis.
2. Choose *Scale*.
3. Select *Categories in reverse order* or *Values in reverse order*.

Changing Values of Data Points

Use

You can change values in charts directly by dragging a data marker in the chart or by using the property pages.

Procedure

Directly in the chart

1. Select a data series using the left-hand mouse button.
2. Click on a data point in the data series and, keeping the left-hand mouse button depressed, drag the data point to the required position.

Using the property pages

1. Double-click the data point using the right-hand mouse button.
2. Choose *Format data point*.

   The dialog box *Format data point* appears.

   If you double-click on the selected data point using the left-hand mouse button you go directly to this dialog box.

3. Choose *Value.*
4. Enter the value required.

Formatting Data Series and Data Points

Use

You can format every data series in a chart and every point in a data series individually.

You can change either the border and fill attributes *or* the line and marker attributes depending on the chart type.

Prerequisites

Before you set an attribute for a data series/data point you must first select either a data series or a data point as follows:

Formatting a data series:

1. Double-click the data series.

Formatting a data point in a data series:

1. Select the data series using the left-hand mouse button.
2. Click to select the data point.
3. Click the right-hand mouse button and choose *Format data point*.

Procedure

Changing border and fill attributes

Choose Pattern.

Change the border attributes and/or the fill attributes as required.

Changing line and marker attributes

Choose *Line.*

Change the line attributes and/or marker attributes as required.

Smoothing the Angles of Line Charts

Use

An interpolated curve is drawn between the points specified by the application. You use this procedure to soften the jagged edges of a line chart. However, your data is not affected.

Prerequisites

You must have selected *Lines* as a chart type.

Procedure

1. Double-click the data series you want to smooth.
2. Choose *Line*.
3. Select the indicator *Smooth.*

Titles, Data Labels and Legends

Definition

There are optional titles for the chart, for the Y axis and for the X axis.

A data label is a text providing additional information, for example about a data series or a data point.

A legend is a key that explains the colors, patterns or symbols in a chart.

Use

You can add, delete and edit the chart and axis titles. For more information, see

Maintaining Chart of Axis Titles [Page 187]

Formatting Chart and Axis Titles [Page 188]

There is a wide range of editing and formatting functions for axis labels and data labels. See:

Specifying Position of Axis Labels [Page 190]

Editing and Formatting Data Labels [Page 191]

Adding Data Labels to a Chart [Page 192]

Changing the Position of Data Labels [Page 193]

You can add/delete legends and format the legend index and the legend key. See:

Adding and Deleting Legends [Page 194]

Formatting Legend Index/Key [Page 195]

Maintaining Chart or Axis Titles

Use

You can maintain a chart title, a title for the X axis and a title for the Y axis. You can enter all, some or none of them.

Prerequisites

To display an axis title you have to specify that the axis itself is to be displayed.

Procedure

Specifying that an axis is to be displayed

1. Select the chart using the right-hand mouse button.
2. Choose *Chart options*.
3. Choose *Axes*.
4. To display the X axis set the indicator *Category (X) axis.* To display the Y axis set the indicator *Value (Y) axis*.

Maintaining titles

1. Select the chart using the right-hand mouse button.
2. Choose *Chart options*.
3. Choose *Title*.
4. Enter the required titles in the fields *Chart title, Category (X) axis* and *Value (Y) axis*.

Formatting Chart and Axis Titles

Use

You can specify a range of display attributes for chart and axis titles including:

- Fill attributes
- Border attributes
- Font attributes
- Alignment

Prerequisites

You must have created a chart title or axis title before you can format it. For more information, see Maintaining Chart or Axes Titles [Page 187].

Procedure

Changing the fill attributes

1. Double-click the text for which you want to change the fill attributes.
2. Choose *Pattern* and select the fill attributes required.

Changing the border attributes

1. Double-click the text for which you want to change the border attributes.
2. Choose *Pattern* and select the border attributes required.

Changing the font attributes

1. Double-click the text for which you want to change the font attributes
2. Choose *Font* and select the font attributes required.

Aligning text

1. Double-click the text for which you want to change the alignment.
2. Choose *Alignment* and select the type of alignment required.

Changing Text of a Text Chart Element

Use

You can change the texts of the following text chart elements:

- Chart title
- Legend entries
- Headings of data points

Procedure

1. Select the text chart element.
2. Click in the selected area using the left-hand mouse button.

   The selection points disappear and the cursor turns into a vertical line.
3. Edit the text.

To leave the text processing mode click anywhere in the chart using the left-hand mouse button.

You can also change the texts for the chart title and axes using the chart options on the property pages.

Specifying Position of Axis Labels

Use

You can specify the position of axis labels using both the property pages and drag and drop.

Prerequisites

You must have specified that the axis is to be displayed. If the axis is not displayed then neither are the axis labels. For more information on displaying an axis, see Maintaining Chart or Axis Titles [Page 187].

Procedure

Specifying the position of axis labels using the property pages

1. Double-click the required axis
2. Choose *Line*
3. Specify in *Tick mark labels* whether the labels are to be displayed *High, Low, Next to axis* or not displayed at all (*None).*

Specifying the position of axis labels using drag and drop

1. Select the axis label.
2. Drag the axis label to the required position by keeping the left-hand mouse button depressed.
3. When the label is at the required position release the mouse button.

Editing and Formatting Data Labels

Procedure

Editing data labels

1. Select on the data label you want to change.

   A small black box appears on either side of each of the data labels in the data series.

2. Click on the data label again.

   A small black box appears at each of the four corners of the data label.

3. Click on the data label a third time.

   A vertical cursor appears. You can move the cursor using the arrows and can edit the text.

Aligning data labels

1. Double-click the data label.
2. Choose *Alignment.*
3. Select the required *Text alignment* and the *Orientation.*

Positioning data labels

1. Double-click the data label.
2. Choose *Alignment.*
3. Select the required *Position.*

Specifying the number format of data labels

1. Double-click the data label.
2. Choose *Number.*
3. Select the required number format.

Specifying the text format of data labels

1. Double-click the data label.
2. Choose *Font.*
3. Select the required text format.

Specifying the border and fill attributes of data labels

1. Double-click the data label.
2. Choose *Pattern.*
3. Specify the border and fill attributes.

Adding Data Labels to a Chart

Use

You can add data labels to a data series or data points.

Procedure

Adding a data label to a data series

1. Double-click on the data series.
2. Choose *Data labels*
3. Select *None, Show value* or *Show label*

Adding a data label to a data point

1. Double-click the data series.
2. Click on the data point in the data series.

   The data point is selected.
3. Click on the data point using the right-hand mouse button.
4. Choose *Format Data Point*.
5. Choose *Data Labels*.
6. Select *None, Show value* or *Show label*.

Changing the Position of Data Labels

Use

You can change the position of titles, legends and data labels for data points. You can reposition them manually or automatically.

Prerequisites

You must have created a data label already. For more information, see Adding Data Labels to a Chart [Page 192].

Procedure

Positioning data labels automatically

1. Select the data label using the right-hand mouse button.
2. Choose *Format Data Labels*.
3. Choose *Alignment* and make the required selections.

Positioning data labels manually

4. Select the data label.
5. Drag the data label to the required position by keeping the left-hand mouse button depressed.
6. Release the mouse button when the data label is at the required position.

Adding and Deleting Legends

Use

You can add a legend and specify its position in the chart. You can also delete an existing legend.

Procedure

Adding a legend

1. Select the chart using the right-hand mouse button.
2. Choose *Chart Options*.
3. Choose *Legend*.
4. Select the position for the legend.

Deleting a legend

1. Double-click on the legend.
2. Choose *Legend.*
3. Select *None.*

Result

The legend is displayed in the chart in the position you have specified. You can use the mouse to drag the legend to a different position and/or change its size/shape.

Formatting Legend Index/Key

Use

You can change the font of the legend index and change fill attributes and border attributes of the legend key.

Prerequisites

There must be a legend displayed in the chart already. For more information, see Adding and Deleting Legends [Page 194].

Procedure

Formatting the legend index

1. Double-click the legend index.
2. Select the font attributes required.

Formatting the legend key

1. Double click the legend key.
2. Select the required display attributes.

Data Tables

Definition

A data table is a table containing the values underlying the data points in the data series.

Use

You can display the data table below the graphic and format it as required.

It makes sense to use a data table only with chart types that have a category axis.

For information on displaying a data table, see Hiding/Showing a Data Table in a Chart [Page 197].

For information on formatting a data table, see Formatting a Data Table [Page 198].

Hiding/Showing a Data Table in a Chart

Use

You can hide/show a data table in some types of chart. You can also hide/show the legend keys for the data table.

Prerequisites

You can display a data table only with the following chart types:

Line charts

Column charts

Area charts

Bar charts

Procedure

To hide/show the data table:

1. Select on the chart using the right-hand mouse button.
2. Choose *Chart Options*.
3. Choose *Data Table*.
4. To show the data table set the indicator *Show data table.* To hide the data table clear the indicator *Show data table.*

   The data table does not replace an axis of the chart but is aligned to the chart.

To hide/show legend keys for the data table carry out the first two steps above and then:

To display the legend keys set the indicator *Show legend keys*. To hide the legend keys clear the indicator *Show legend keys*.

The legend keys are shown at the side of the data table.

Formatting a Data Table

Use

You can change the line attributes and the font attributes of the data table.

Procedure

Changing the line attributes of the data table:

1. Double-click the data table and choose *Line*.

   Change the line attributes as required.

Changing the font attributes of the data table:

1. Double-click the data table and choose *Font*.

   Change the font attributes as required.

SAP Business Graphics

Definition

This is a graphics product for displaying business graphics. It is often referred to in documentation as SAP BUSG or simply BUSG.

Use

You can use BUSG to display business graphics as an external program graphic.

You can also use BUSG with the Graphical Framework to display business graphics as an ActiveX control.

Structure

You can find more information on using BUSG without the Graphical Framework in SAP Business Graphics [Ext.]. You can find more information on the programming interfaces for BUSG in Calling SAP Business Graphics [Ext.].

Programming Interfaces

The programming interfaces are of interest to developers programming graphics. More information is available on the following areas:

- Data Containers Supported [Page 201]

  This section contains information on the data container provided by SAP and the data container wizard you can use to generate your own data container. It also gives details on the interfaces to the application and to the graphics proxies that you need to implement in the data container.

- Graphics Proxy Supported [Page 202]

  This section contains information on the business graphics proxy; in particular on declaring the names of attributes in the data container and assigning customizing bundles to the ports in the graphics proxy. It also mentions the method for setting the names of attributes in the data container.

- Graphics Products Supported [Page 204]

  This section contains information on the business graphics products supported by the Graphical Framework.

- Function Modules [Page 207]

  This section contains information on function modules used for business graphics.

Data Containers Supported

Definition

The data containers supported are those data containers that the Graphical Framework can use to supply the business graphic with data.

Use

You can use a data container provided by SAP if you do not want to program your own data container.

The standard data container for business graphics is LCL_DC_PRES and is generated using the following parameters:

- DDIC structure: GFWDCPRES
- Key attribute: OBJID
- Filter attribute: FILTER
- ABAP Objects class: LCL_DC_PRES
- Name of include: GFW_DC_PRES

  For more information, see Data Container Wizard [Page 39].

A data container must implement the following interfaces:

- IF_DC_ACCESS

  This is a nested interface contained in IF_DC_MANAGEMENT and IF_DC_SUBSCRIPTION. It has several methods for interacting with objects in the data container.

- IF_DC_MANAGEMENT

  This is the data container´s interface to the application.

- IF_DC_SUBSCRIPTION

  This is the data container´s interface to the graphics proxies so that they can register and deregister with the data container.

Graphics Proxy Supported

Definition

There is a global class CL_GUI_GP_PRES for business graphics. It contains a constructor method and the method SET_DC_NAMES.

Use

CL_GUI_GP_PRES

Declaring the names of attributes in the data container

There are some attributes that must be declared and others that are optional. For more information, see the class documentation for CL_GUI_GP_PRES in the class builder.

Assigning customizing bundles to ports in the graphics proxy

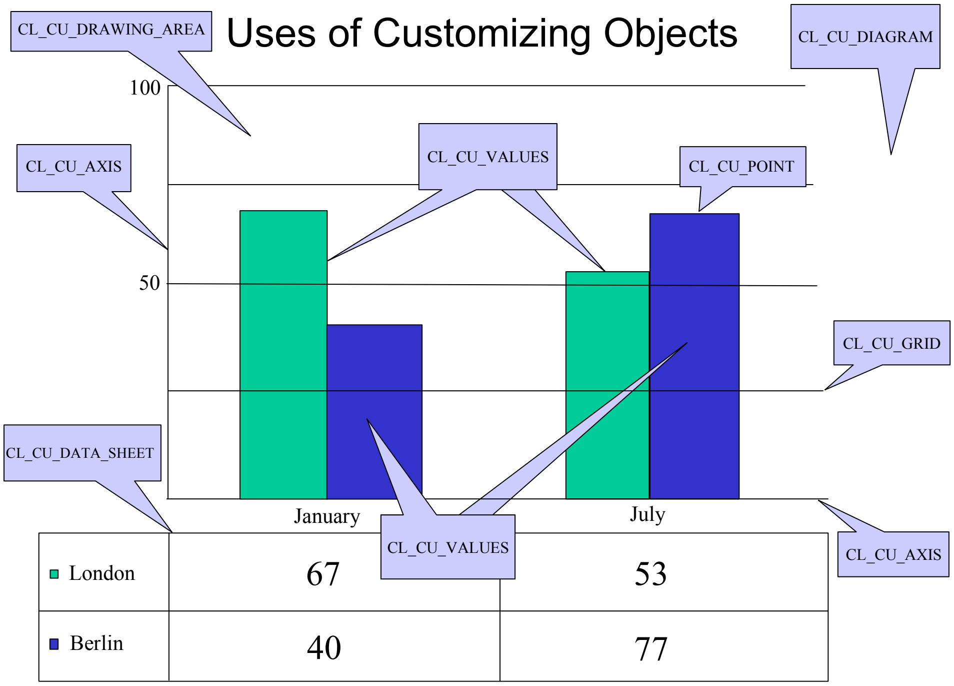
Not all combinations of customizing bundles and ports are allowed. For more information, see the class documentation for CL_GUI_GP_PRES in the class builder.

For information on the automatic generation of customizing objects, the display context hierarchy and other details on programming the product-specific part of the graphics proxy, see the class documentation for CL_GUI_GP_PRES in the class builder.

SET_DC_NAMES

You use this method to set the names of the attributes in the data container. There are a number of importing parameters all of which are documented in SET_DC_NAMES.

Integration



	January	July
🟩 London	67	53
🟦 Berlin	40	77

The graphic above shows uses for the various customizing objects. For example, CL_CU_GRID can be used to customize gridlines.

Graphics Products Supported

Definition

A graphics product is a graphical component such as Chart.

Use

The Graphical Framework can use various graphics products to display graphics on a wide range of frontend platforms.

It is important to distinguish between graphics displayed as controls and those displayed as external program graphics.

The graphics products are organized in product management. For more information, see Product Management [Page 114].

You can use the following graphics products with the Graphical Framework:

Graphics displayed as controls

- Chart
- BUSG

Graphics displayed as external program graphics

- BUSG

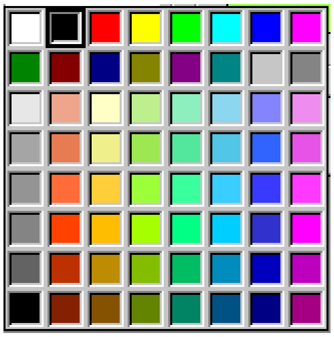
Color Palettes

Chart:

Table showing Graphical Framework values for colors in Chart

2	1	7	4	5	8	3	6
13	15	11	12	14	16	18	9
17	27	49	33	19	36	43	57
10	26	50	35	34	46	48	62
21	25	53	52	38	41	42	45
20	28	54	51	22	37	47	24
23	29	58	30	31	44	32	63
55	60	59	40	39	56	61	64

The Graphical Framework (GFW) values range from 1 to 64. Reading from the table above, the top left cell tells you that the GFW value 2 corresponds to the color white (corresponding cell in the color palette below), and reading to the right the GFW value 1 corresponds to the color black (in the color palette below) and so on.



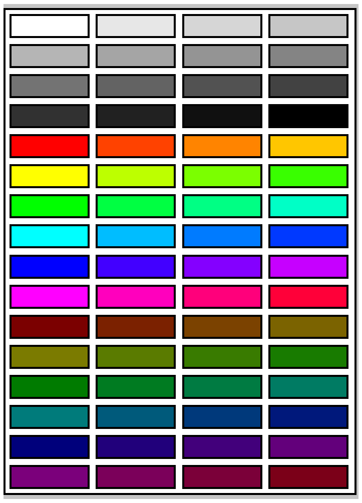
BUSG

Table showing Graphical Framework values for colors in BUSG

2, 49	17, 62	36	18
	10, 27	21	9, 20
	23		
47			1, 55
7	28	26	53, 54
4, 50	33, 51	35, 52	
5		22, 34, 38	19
8	37, 41, 46	48	42
3		43	
6, 24, 45, 57	63		25
15	29, 60		59
12, 30, 58	40		
13			31, 39
16, 44	56		
11, 32, 61			
14, 64			

The GFW values range from 1 to 64. Reading from the table above, the top left cell tells you that the GFW values 2 and 49 are used for the color white (corresponding cell in the color palette below), and reading to the right the GFW values 17 and 62 correspond to the color light gray and so on.

On the basis of the RGB values (for the GFW colors) the most suitable colors for each product are selected.



Function Modules Supported

Definition

GFW_PRES_SHOW is the function module for displaying a single business graphic.

GFW_PRES_SHOW_MULT is the function module for displaying several business graphics at the same time.

Use

It is estimated that in a large proportion of cases where graphics are used (approximately 80-90 percent of business graphics) users require only a basic, standardized display of simple data with fixed customizing.

You can use GFW_PRES_SHOW to display one ActiveX control or JavaBean. It contains an alternative strategy for those platforms that do not support controls. In such cases the graphic is displayed as an external program (SAPBUSG).

The graphics product used by the function module is specified in product management. For more information, see Product Management [Page 114].

The function modules should cover only the basic case; that is sufficient in most situations. The object-oriented interface of the Graphical Framework should be used for situations requiring more complex functionality, in particular interaction and stipulating a particular graphics product.

For more information on function modules for the Graphical Framework, see the documentation on GFW_PRES_SHOW and GFW_PRES_MULT.

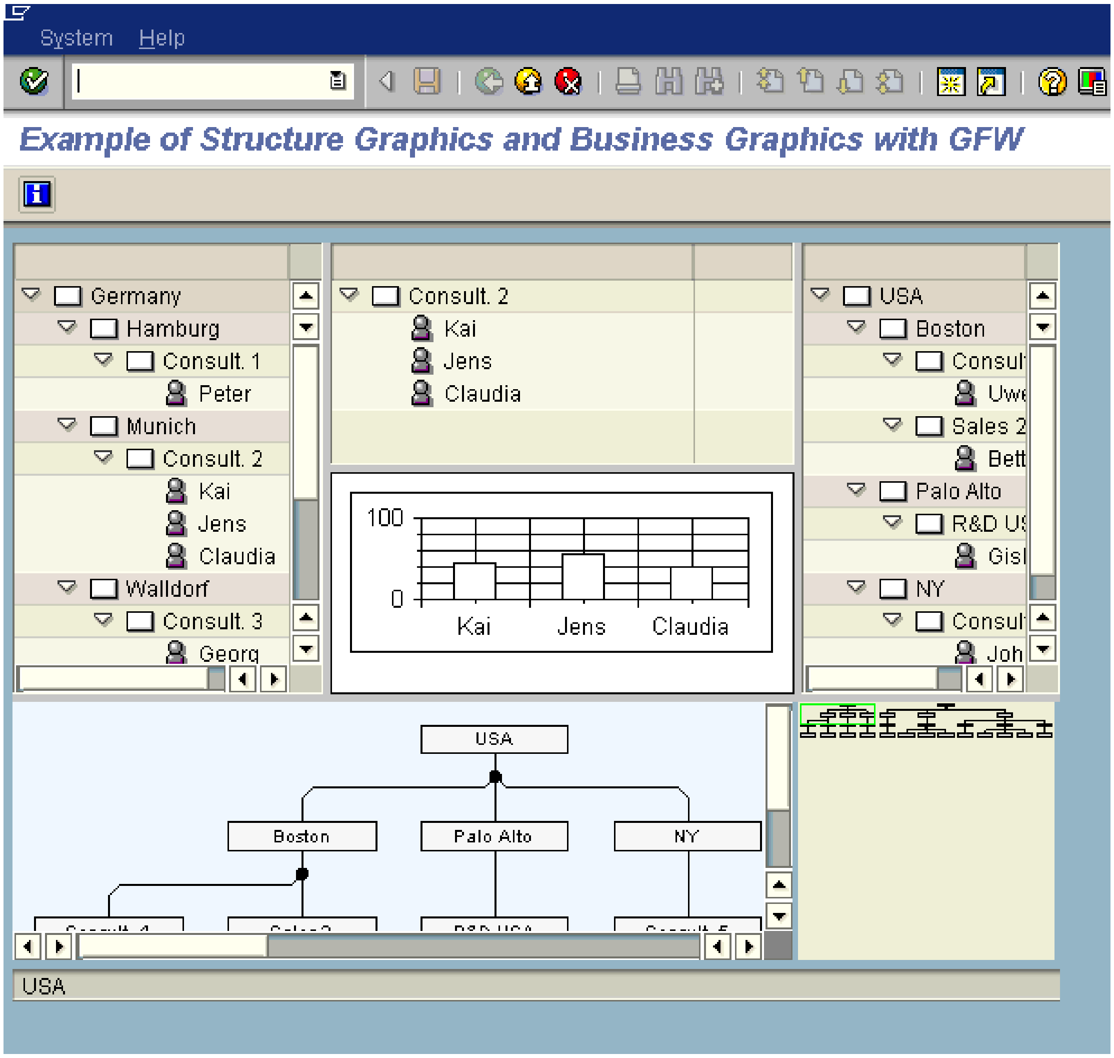
Structure Graphics

Structure graphics are graphics that display data in the form of a structure. There are the following types of structure graphic:

- Tree graphics
- Gantt charts
- Networks

Example of a Hierarchy Graphic using the Graphical Framework

The graphic shows an interactive hierarchy graphic created using the Graphical Framework. The lower sections display all of the data in the data container. The upper three sections show subsections of the data in the form of a tree.



Types of Structure Graphic

Definition

Structure graphics displayed as a hierarchy are called hierarchy graphics while those displayed as networks are called networks. It is important to distinguish between graphics products on the one hand and type of structure graphic on the other. Some graphics products, such as Netronic Chart can display both hierarchy graphics and structure graphics.

Structure

Hierarchy graphics have the following characteristics:

- An object can have only one predecessor
- An object can have one or more successors
- Cycles are not permitted

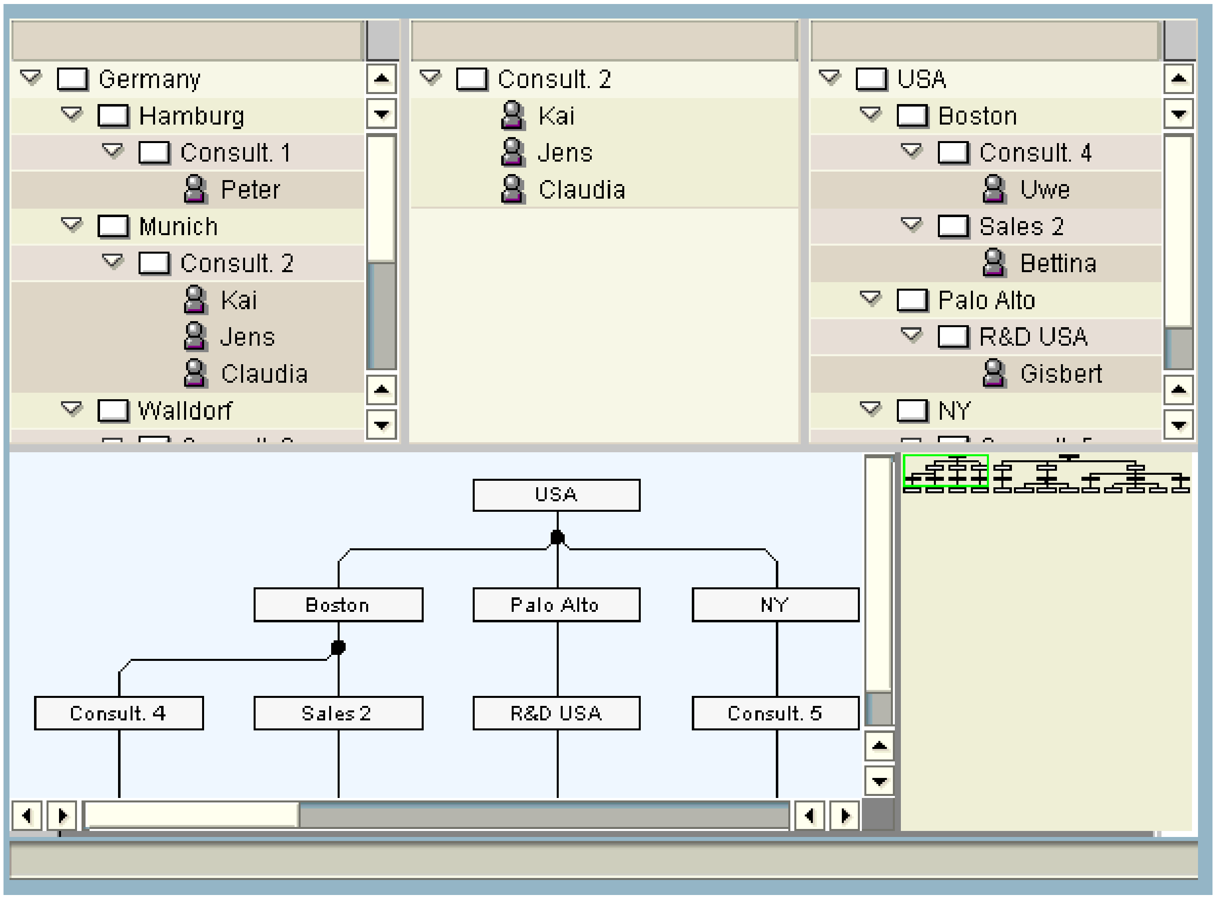
  This means that an object cannot be the successor of one of its own successors.

Networks have the following characteristics:

- An object can have several predecessors
- An object can have one or more successors
- Cycles are permitted

Example of a Hierarchy Graphic

Hierarchy graphic consisting of several controls in a splitter



The graphic above comes from the demo program GFW_DEMO_HIER2 and shows several GFW hierarchy graphics in a splitter.

- The window is split up into several sections. The lower graphic (SAP Network) has a display area and a navigation area. It always displays all of the data in the data container. The upper three (SAP Tree) each show subsections of the data.

All of the graphics are based on a single data container. The data container has a filter function that can be used to generate the required subsections of the data. The graphic on the left displays data with the filters 1 and 2, the graphic in the middle displays data with filter 2 and the graphic on the right displays data with filter 3. The lower graphic does not have a filter which means that there are no restrictions to the data it displays.

User Documentation

The objective of the user documentation on structure graphics is to provide you with detailed procedures for working with these graphics products or links to documentation where such information is provided.

SAP Tree

A tree graphic is a graphic that displays objects in a tree structure. SAP Tree is a tree graphic in the form of an ActiveX control. For more information, see SAP Tree [Page 213]. This product can display only hierarchies, not networks.

Gantt Chart

A Gantt chart is a graphical display that shows one or more processes over a period of time. For more information, see Gantt Chart [Page 214]. This product can display all sorts of structure graphics including networks.

SAP Tree

Definition

A tree graphic is a graphic that displays objects in a tree structure.

Use

The tree control is an ActiveX control that fulfills the basic requirements of a tree control but is not tailored to the needs of individual applications.

Structure

The tree control can be implemented in three different ways:

- Simple tree structure: A simple tree structure with the nodes arranged in a single column
- List structure: A tree with nodes that have several items. The items are displayed from left to right
- Column structure: A tree structure with freely-definable columns

Gantt Chart

Definition

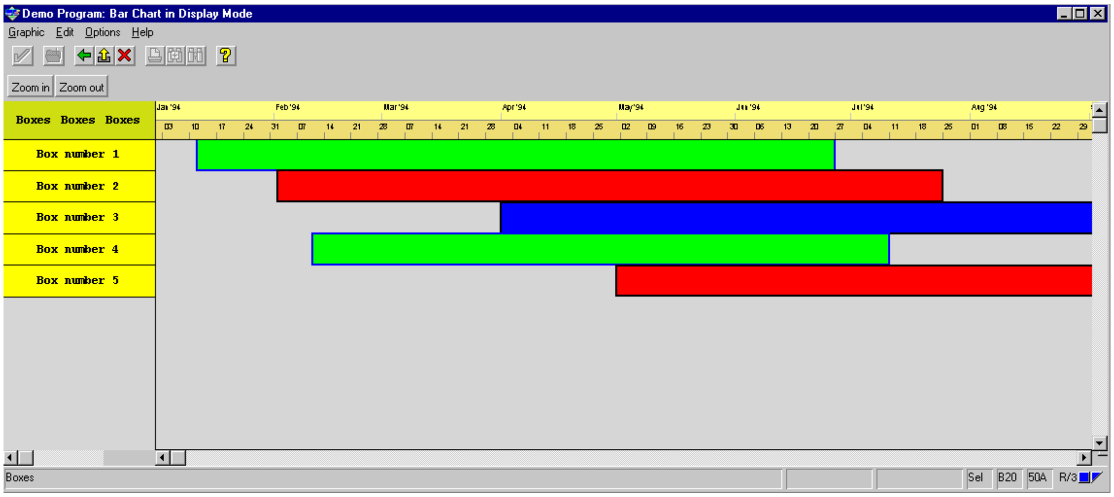
A Gantt chart is a graphical display that shows one or more operations or activities over a period of time.

Use

You can use a Gantt chart to plan operations or activities over a period of time.

Structure

The graphic shows a typical Gantt chart. The horizontal bars represent operations or activities whose duration is shown by the scale on the X-axis.



Programming Interfaces

The programming interfaces are of interest to developers programming graphics. More information is available on the following areas:

- Data Containers Supported [Page 216]

  This section contains information on the data container provided by SAP and the data container wizard that you can use to generate your own data container. It also gives details on the interfaces to the application and to the graphics proxies that you need to implement in the data container.

- Graphics Proxy Supported [Page 217]

  This section contains information on the structure graphics proxy; in particular on declaring the names of attributes in the data container and assigning customizing bundles to the ports in the graphics proxy. It also mentions the structure graphics interface and the method for setting the names of attributes in the data container.

- Graphics Products Supported [Page 220]

  This section contains information on the structure graphics products supported by the Graphical Framework.

Data Containers Supported

Definition

The data containers supported are those data containers that the Graphical Framework can use to supply the structure graphic with data.

Use

You can use a data container provided by SAP if you do not want to program your own data container.

The standard data container for structure graphics is LCL_DC_HIER and is generated using the following parameters:

- Name of include: GFW_DC_HIER
- ABAP Objects class: LCL_DC_HIER
- DDIC structure: GFWDCHIER

    OBJID (key attribute - *mandatory*)

    FILTER

    CU_REFOBJ

    OBJTYPE (node, link)

    PRED (mandatory)

    SUCC

    NODE_TEXT0…3 (*mandatory*)

    STATUS_OBJ (for example, “expanded”)

    The fields marked *mandatory* are the minimum attributes required for a data container for structure graphics.

    For more information, see Data Container Wizard [Page 39].

A data container must implement the following interfaces:

- IF_DC_ACCESS

    This is a nested interface contained in IF_DC_MANAGEMENT and IF_DC_SUBSCRIPTION. It has several methods for interacting with objects in the data container.

- IF_DC_MANAGEMENT

    This is the data container´s interface to the application.

- IF_DC_SUBSCRIPTION

    This is the data container´s interface to the graphics proxies so that they can register and deregister with the data container.

Graphics Proxy Supported

Definition

There is a global class CL_GUI_GP_HIER for structure graphics. It contains a constructor method, the method SET_DC_NAMES and the interface IF_GP_HIER along with its methods GET_SELECTED_NODES and SET_SELECTED_NODES.

Use

CL_GUI_GP_HIER

Declaring the names of attributes in the data container:

> There are some attributes that must be declared and others that are optional. For more information, see the class documentation for CL_GUI_GP_HIER in the class builder.

Assigning customizing bundles to ports in the graphics proxy

> Not all combinations of customizing bundles and ports are allowed. For more information, see the class documentation for CL_GUI_GP_HIER in the class builder.
>
> For information on the automatic generation of customizing objects, the display context hierarchy and other details on programming the product-specific part of the graphics proxy, see the class documentation for CL_GUI_GP_HIER in the class builder.

Defining object types:

> To define a node in the data container use the attribute CO_OBJTYPE_NODE. To define a link in the data container use the attribute CO_OBJTYPE_LINK. You can do without links by using the AUTO_LINK function. For information on AUTO_LINK, see AUTO_LINK [Page 219].

IF_GP_HIER

In addition to the interface IF_GRAPHIC_PROXY available to all graphics proxies, this interface is used by the structure graphics proxy. It contains two methods:

GET_SELECTED_NODES: The application calls this method to find out which nodes were selected.

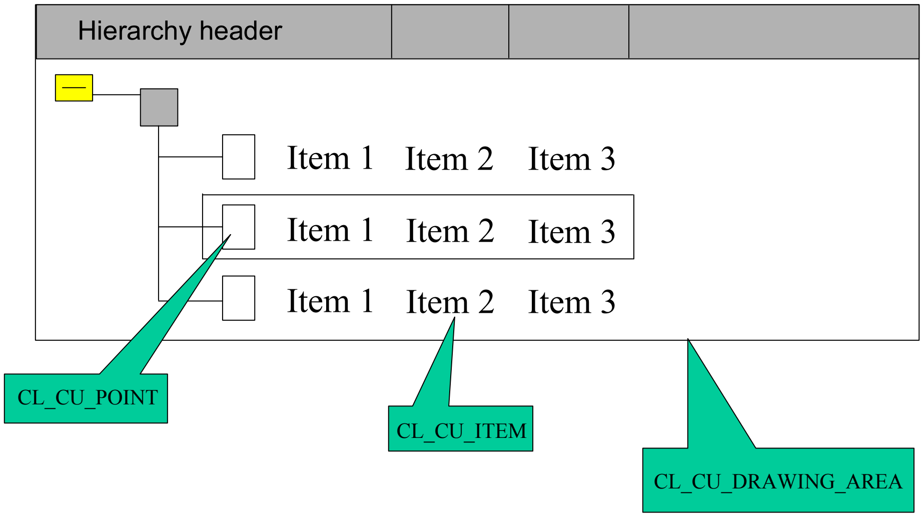
SET_SELECTED_NODES: The application calls this method to select nodes.

SET_DC_NAMES

You use this method to set the names of the attributes that are in the data container. There are a number of importing parameters all of which are documented in SET_DC_NAMES.

Integration

Uses of Customizing Objects



CL_CU_POINT is relevant for the whole line (inside the rectangle) but it is especially important for customizing the node symbol (the square inside the rectangle).

AUTO_LINK

Use

AUTO_LINK is a parameter of the structure proxy's constructor method. When the parameter AUTO_LINK = GFW_TRUE the graphics proxy automatically generates links from the nodes in the data container.

Using the AUTO_LINK function improves performance.

You can only use this function with hierarchy graphics. For network graphics AUTO_LINK must be set to GFW_FALSE.

Prerequisites

The parameter AUTO_LINK must be set to GFW_TRUE. If it is set to GFW_FALSE then the links have to be maintained in the data container as usual. In the latter case the nodes and links are independent objects in the data container.

Features

The nodes define the links between each other using the attributes SUCC and PRED.

Graphics Products Supported

Definition

A graphics product is a graphical component such as SAP Tree.

Use

The Graphical Framework can use various graphics products to display graphics on a wide range of frontend platforms.

The graphics products are organized in product management. For more information, see Product Management [Page 114].

You can use the following graphics products with the Graphical Framework:

Graphics as controls

- SAP Tree
- SAP Net (Gantt chart)