Generic Business Tools for Application Developers (BC-SRV-GBT)

HELP.BCSRVGBT

Release 4.6C

Copyright

Icons

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

Contents

Generic Business Tools for Application Developers (BC-SRV-GBT)

The tools and how they are installed in the application is described in the following documentation:

Business Communication Interface - Sending from R/3 Applications

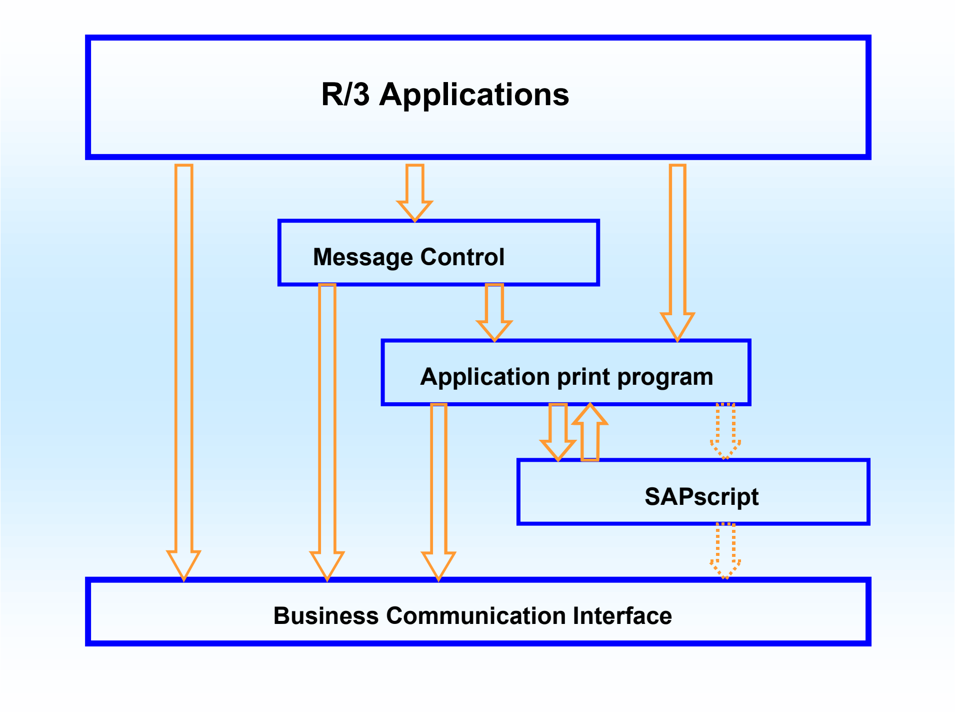
Purpose

This component allows R/3 application developers to integrate internal and external sending of documents in their applications without incurring great costs. The Business Communication Interface (BCI) is not only used for controlling the sending and receipt of documents, but also for extensive status handling and for making all the send information on an application object available.

In addition to interfaces without dialog, the BCI also offers dialogs that allow you to preselect data required for the sending of the document so that this can be stored temporarily or postprocessed.

Integration

Documents can be sent from the R/3 applications using the BCI directly. However, Message Control and SAPscript also support the BCI and, as a result, enable the applications that already send documents in this way to benefit from the enhanced functions without incurring the costs of modification.



When documents are sent externally, the BCI transfers the send objects to SAPconnect [Ext.].

Features

The BCI provides the following functions:

- Sending of documents with attachments and notes

- Sending of documents to internal and external recipients, including organizational units, address objects and other objects that support the interface IFRECEIVE, which provides the attribute RECEIVE.
- Sending of documents via all communication methods supported by the R/3 System (Fax, Internet, X.400, Remote Mail, Paging, Printing).

  When documents are sent to pagers or printers, the BCI presupposes that the paging address or the printer address of the recipient is maintained in the central address management.
- Sending of documents with all send attributes (for example, Express, Copy, Reply required)
- Automatic moving of the document sent to any folder
- Either asynchronous or synchronous sending of documents

  Synchronous sending of documents can be restricted for external communication.
- Linking of the application object to the documents

  The link is strongly recommended.
- Sending under another name
- Active status handling
- Receipt of faxes, Internet messages and IDocs, including possible inbound processing, for example, direct forwarding or starting of a workflow.

The BCI also provides the following dialog interfaces:

- Creation of a new document of any class with or without attachments
- Recipient determination, in which any address type can be used.

In addition, sample programs for the use of the BCI are available (RSSOKIF1 and RSSOKIF2).

Business Communication Interface

Definition

Standard interface for sending from the R/3 System and receiving in the R/3 System. The Business Communication Interface (BCI) controls the processing of documents to be sent from the R/3 System, including the status confirmations and the receipt of documents.

Structure

The BCI is implemented via objects from the Business Object Repository [Ext.] (BOR). A send process is split into steps that can be executed by calling methods. An object model, which is completely implemented, containing all the object types required in order to send, along with the corresponding methods, is available for this. This model is based on the two objects types MESSAGE and RECIPIENT.

Information Availability via Object Links

Use

Various types of information arise from the business processes of a company. Due to the increasing importance of the use of information, this information should be available quickly and at the right place. However, this will only be the case if the insertion of data can be carried out simply, quickly and at the right time or if this can be done automatically by the system.

To fulfill these requirements, the Business Communication Interface (BCI) enables all send processes for a business object to be automatically linked to it. This link provides the following advantages for the applications:

- Access to all send processes from the application object

  The user can call the send processes of the application object via the system menu. Further generic object services [Page 147] are available in the system menu.

- Status Confirmation
- The format in which the document left the R/3 System (for example, Postscript or PDF for faxes) is available.

  ⚠

  During reorganization all documents without links are deleted.

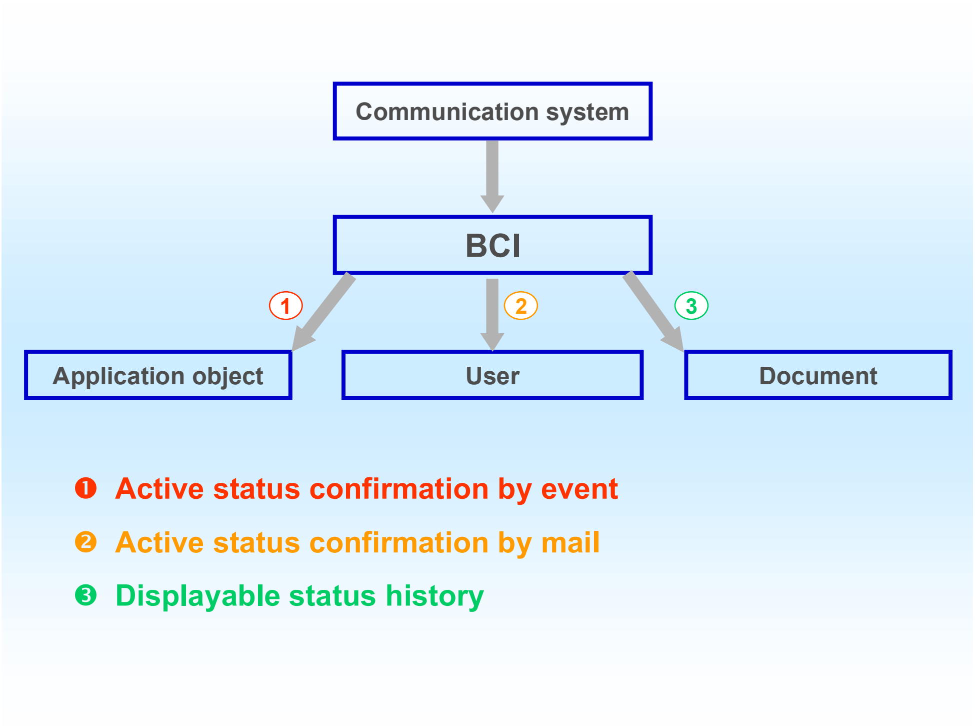
Activities

In order to link an application object with a document, the key of its BOR object has to be transferred by the application. If this is done, the link is automatically created when the document is generated. For further information on this link service [Page 137], see the documentation on BC - Object Links.

Status Confirmation

Use

The BCI offers various options for finding out the send status after a send process.



Communication system

BCI

1 2 3

Application object

User

Document

❶ Active status confirmation by event

❷ Active status confirmation by mail

❸ Displayable status history

For more information, see

- Send Process and Status Confirmation [Page 14]
- Active Status Confirmation by Event [Page 15]
- Active Status Confirmation by Mail [Page 16]
- Status History via Object links [Page 17].

Send Process and Status Confirmation

For **internal sending** the whole send process runs synchronously. The return codes for the Submit method are final in this case. Further status information is not obtained. No status event is triggered.

For **external sending** the status confirmation is more complex. Therefore, firstly the rough process of external sending will be presented:

1. The whole recipient list is broken down into individual recipients and a send order is placed in the queue for each individual recipient.
2. The next SAPconnect send process reads the send orders from the queue and determines the responsible nodes that represent the external communication systems in the R/3 System.
3. The send orders are transferred to the relevant responsible external communication system, for example, to a fax server.

From the view of the sender, only the first step is synchronous. The rest of the process is asynchronous. The return codes that the Submit [Page 25] method returns can, therefore, only refer to the first, synchronous part. Furthermore, the fact that there are, for the most part, several recipients (for example, a distribution list) must also be taken into account. Thus, the Submit method returns the following return codes:

- SY-SUBRC after method call

  This is 0 as soon as the send order has been successfully placed in the queue for at least one recipient.

- Export parameter *SentToAll*

  This is 0 when the send orders have been successfully placed in the queue for all recipients.

- Export parameter *RecipientsInfo* (Table of structure SOSNDINF)

  This table contains the object keys and the return code for the broken down recipients. The return code for a recipient is 0 when the send order has been successfully placed in the queue for this recipient. Further status information is confirmed asynchronously by event. This exact object key is then reported in the event container [Ext.] under the parameter *Address*.

Active Status Confirmation by Event

Use

If a status confirmation generates an event [Ext.], this event can be processed further by a receiver. A single-step task or a workflow [Ext.] that the customer (or the application) has created can serve as a receiver for the event. Processing of status information then takes place within this receiver.

For further information on workflow, see the BC - SAP Business Workflow [Ext.] documentation.

Integration

The Business Communication Interface (BCI) triggers one of the following events, depending on the document sent, in the application object instance linked with the document:

- ErrorSendStatusReached
- WarningSendStatusReached
- InfoSendStatusReached

The following parameters are transferred via the event container [Ext.]:

- Document (key of the message object)
- Recipient (key of an address object; the same key that is returned to the table *RecipientsInfo* via the Submit [Page 25] method).
- Sender (key of an address object)
- Date of status
- Time of status
- Status information as a T100 message, that is with MSGID, MSGNO, MSGV1, MSGV2, MSGV3, MSGV4

The macro SX_GETREPTYPE from the type group SX classifies the status using the message number in to the categories

- Delivery Report
- Non-Delivery Report
- Read Notification
- Non-Read Notification
- Information.

Activities

If the application object supports the interface IFSENDSTAT, the event is then triggered by the BCI. Therefore, it does not have to be set from the application that the event is triggered.

Active Status Confirmation by Mail

Use

A further option for active status confirmation is to inform the sender of the status by mail. This function has the effect that the sender receives a message in their Workplace [Ext.] inbox informing them of the status of the document sent. In the process, you can set the type of status confirmation for which a message is to be sent. You can choose between the following values:

Value	Description
SPACE	No status confirmation at all
E	Message only if errors exist This is the standard setting.
W	Message only if errors exist and for warnings.
I	Message for every status

Activities

You can define whether a message is to be sent and for which status using the send attribute StatusinfoByMail [Page 69].

Status Display by Object Links

Use

This function has the effect that the status is confirmed passively. The users can display the status of a document sent to an application object in the application transaction. The procedure is always the same, irrespective of the application. It is carried out via the generic object services [Page 147] in the system menu. From the document display you can branch directly to the recipient list, in which the current status as well as the full status history is displayed for each recipient.

Activities

In order to create the object link, publish the key of your BOR object by calling the function module SWU_OBJECT_PUBLISH. For further information on this link service [Page 137], see the documentation on BC - Object Links.

Connection to the Interface

To connect to the interface, you call the methods of the BOR objects Message [Page 19] and Recipient [Page 45].

Indirect connection via Message Control [Page 71] or indirect connection via SAPscript [Page 72] are special cases.

In addition to the basic functions, the BCI enables documents to be sent under another name [Page 74].

The example reports RSSOKIF1 and RSSOKIF2 are available.

Message

Definition

A BOR object type that represents a document with recipient list, that is, the recipients for the next send process. The MESSAGE object represents the actual interface of the Business Communication Interface.

Use

Gradually, the document to be sent can be created, the recipient list can be maintained, and so on. At the end, the finished document is sent to the recipients via the *Submit* method.

⚠

> All changes to a specific instance of a Message object are at first buffered in the main memory only, in order to improve performance. The changes are copied to the database, or a newly created instance becomes persistent, only once the *Save* method has been called.

Structure

Attributes of the Message Object

Attribute	Description
Description	Title of the document
Recipients	Recipient list as a table of Recipient objects, meaning that it is to these Recipient objects that documents are sent in the next send process (with the MESSAGE.Submit method).
Asynchronous	Flag indicating that sending is to be asynchronous.
OutboxFlag	Flag indicating that a link to the document sent is to be stored in the outbox.
LinkFolderType, LinkFolderYear, LinkFolderNumber	ID of the folder in which (possibly in addition to the outbox) a link to the document sent is to be stored.
NextSender	Sender for the next send process of this document

Message Object Methods

Create [Page 21]

Attach [Page 23]

Find [Page 24]

Submit [Page 25]

Save [Page 26]

AddCopyOfRecipient [Page 27]

RemoveRecipient [Page 28]

Create

Definition

Method of the BOR object Message, either with or without dialog, that creates a document. When you create without dialog, the content, name and title as well as header data can be specified as import parameters. The document content can either be in the document itself or outside of it. In the latter case, the document itself is a link.

Structure

Import Parameters

Import Parameter	Description
DOCUMENTTITLE	Title of the document
DOCUMENTLANGU	Language of the document
DOCUMENTSORTFIELD	Sort field
DOCUMENTPRIORITY	Priority
DOCUMENTSENSITIVITY	Sensitivity
NO_DIALOG	Flag indicating that the document is to be created without dialog
PARENTFOLID	Folder in which the document is to be stored.
DOCUMENTNAME	Name of the document
DOCUMENTTYPE	Class of document
DocumentContent	Content of the document (table)
DocumentSize	Size of the document in bytes The size MUST be specified for PC documents.
DocumentHeader	Type-specific document header data (table) In general, the table does not have to be filled.
Originator	Owner of the document as an address object
ReferenceType	Type of reference if the document content is outside of the document Possible values: **F** - Reference to file on file server In this case, the file name with path specification must be entered in the first line of DocumentContent. **U** - Reference to Web site The URL must be transferred in DocumentContent **KEY** must appear at the start of every line used, followed by the URL. **O** - Reference to a business object The object ID must be transferred as structure SOXOBJ in the first line of the table DocumentHeader. **A** - Reference to an archive
FileImporte	Flag that controls whether the file selector box for selecting the file to be imported is started.

Export Parameters

Export Parameter	Description
DOCUMENTNAME	Name of the document

Attach

Definition

Method of the BOR object Message that appends an attachment to a document.

Structure

Import Parameters

Import Parameter	Description
ATTACHMENT	Document to be created
ATTACHMENTTITLE	Title of the attachment
AttachmentType	Attachment document class
AttachmentHeader	Type-specific header data of attachment (table) In general, the table does not have to be filled.
DocumentContent	Content of the document to be created (table)
DocumentSize	Size of the attachment in bytes The size MUST be specified for PC documents.

Export Parameters

None.

Find

Definition

Method of the BOR object Message with dialog that finds a document that has already been created. Documents located in folders that can be accessed via the interface can be searched for exclusively.

Structure

Import Parameters

None.

Export Parameters

Export Parameter	Description
DOCUMENTNAME	Name of the document
DOCUMENTTYPE	Class of document

Submit

Definition

Method of the BOR object Message without dialog that sends a document to a recipient list that has already been created. For external recipients, ‘sends’, in this case, means that the send order is transferred to the queue.

After sending, the selected recipients (list of Recipient objects) are removed. The broken down recipients of this send process are still available via the export parameter RecipientsInfo. If the document was successfully sent to all recipients, the general note is also deleted.

⚠

This reinitialisation is at first buffered in the main memory only. Therefore, the Save method should always be called after a Submit.

How can users find out whether the document was successfully sent to individual recipients?

See Send Process and Status Confirmation [Page 14].

Structure

Import Parameters

None.

Export Parameters

Export Parameter	Description
SentToAll	Flag indicating that the document was successfully sent to all recipients in this send process.
AllBindingDone	Flag indicating whether all object links were successfully created (compare the Save method).
RecipientsInfo	Table of broken down recipients each with return code.

Save

Definition

Method of the BOR object Message without dialog that saves the document to be sent with recipient list. If the object is not yet persistent, that is, if it does not yet exist in the database, an entry is created in the database. Otherwise, the changes are updated in the database.

Structure

Import Parameters

None.

Export Parameters

Export Parameter	Description
AllBindingDone	Flag indicating that all object links were successfully created. If not all of the object links could be created, no exception is triggered but AllBindingDone = Space is returned.

AddCopyOfRecipient

Definition

Method of the BOR object Message without dialog that adds a copy of the Recipient object specified to the recipient list. In the process, a copy is created and a handle is added to it.

Structure

Import Parameters

Import Parameter	Description
NewRecipient	New Recipient object to be added

Export Parameters

None.

RemoveRecipient

Definition

Method of the BOR object Message without dialog that removes a Recipient object from the recipient list.

Structure

Import Parameters

Import Parameter	Description
Recipient	The Recipient object to be removed

Export Parameters

None.

EditRecipientList

Definition

Method of the BOR object Message with dialog that allows the recipient list to be maintained. The dialog method EditRecipientList exists in addition to the AddCopyOfRecipient and RemoveRecipient methods, which are executed without dialog. The EditRecipientList method calls the send screen that offers, for example, the following functions:

- Enter recipient
- Create note
- Change settings for the folders in which links of the document sent are created (outbox and possibly another folder).

The note created in the send screen is the general note of the Message object. When sending to a specific Recipient object, this note is only included if the Recipient object does not contain a private note.

Structure

Import Parameters

None.

Export Parameters

None.

SetAsynchron

Definition

Method of the BOR object Message without dialog that sets whether sending is to be synchronous or asynchronous.

Structure

Import Parameters

Import Parameter	Description
Asynchronous	Flag that controls whether sending is asynchronous (**X**) or synchronous (**Space**).

Export Parameters

None.

AddApplObjectID

Definition

Method of the BOR object Message without dialog that adds a further application object ID. During sending a link is created for all added application objects with the document sent using the Submit method.

Structure

Import Parameters

Import Parameter	Description
Object ID	ID of the application object

Export Parameters

None.

SetOutboxFlag

Definition

Method of the BOR object Message without dialog that sets whether a link to the document sent is stored in the outbox.

Structure

Import Parameters

Import Parameter	Description
OutboxFlag	Flag that controls whether, in the outbox, a link to the document sent is stored (**X**) or not (Space).

Export Parameters

None.

SetLinkFolder

Definition

Method of the BOR object Message, either with or without dialog, that sets whether a link to the document sent is stored in a private folder. This can be performed either in addition to or instead of the link in the outbox. The SetLinkFolder method also determines which folder the link is to be stored in. If the folder ID is specified as an import parameter, it is copied without dialog. If this import parameter is not filled, a folder can be selected in dialog.

Structure

Import Parameters

Import Parameter	Description
FolderID	ID of the private folder in which the document sent is to be stored.

Export Parameters

None.

ClearLinkFolder

Definition

Method of the BOR object Message that deletes the settings for the link folder, which were set using the SetLinkFolder method.

Structure

Import Parameters

None.

Export Parameters

None.

AddGeneralNote

Definition

Method of the BOR object Message without dialog that allows a general note to be specified for the message object. The application specifies the text for the general note as a table and in this way a general note is created. The general note is sent to the recipients for whom no private note, which can be created for each Recipient object, exists.

Structure

Import Parameters

Import Parameter	Description
NoteText	Table with the text of the note

Export Parameters

Export Parameter	Description
NoteType	Type from the ID
NoteYear	Year from the ID
NoteNumber	Number from the ID

DeleteGeneralNote

Definition

Method of the BOR object Message without dialog that deletes a general note. For further information on general notes, see AddGeneralNote [Page 35].

Structure

Import Parameters

None.

Export Parameters

None.

SetProcGeneral

Definition

Method of the BOR object Message without dialog that sets the general execution parameters when a Message object is an executable document.

⚠

The execution parameters are firstly buffered in the main memory. The *Save* method must, therefore, always be called after the *SetProcGeneral* method so that the parameters are entered into the database.

⚠

The parameters of a document that has already been sent can no longer be changed. This also applies to the execution parameters.

Structure

Import Parameters

Export Parameters	Description
ProcessingType	Execution type that specifies what is to be executed. The following types are available: **D** - Dialog Module **F** - Function Module **R** - Report with Memory **S** - Report with Memory **T** - Transaction **U** - Transaction with Memory.
ProcessingElement	Execution element, for example, the name of the report
SkipFirstScreen	Flag that controls whether the selection screen is not to be displayed.

Export Parameters

None.

SetProcMemoryID

Definition

Method of the BOR object Message without dialog that sets the memory ID for the SAP memory. This is set if the SetProcGeneral [Page 37] method sets the execution type Report with Memory (**S**) or Transaction with Memory (**U**). In this case, the data is transferred via the SAP memory.

Structure

Import Parameters

Export Parameter	Description
MemoryID	Memory ID In this case, parameters for the SAP memory have to be set using the AddProcImportParam [Page 40] method.

Export Parameters

None.

AddProcSetGetParam

Definition

Method of the BOR object Message without dialog that adds Set/Get parameters in the form of **ID,Value** for executable documents (compare the SetProcGeneral [Page 37] method). By calling the method several times, several Set/Get parameters can be attached to a document.

⚠

The Set/Get parameters are firstly buffered in the main memory. The *Save* method must, therefore, always be called after the *AddProcSetGetParam* method so that the parameters are entered into the data base.

⚠

If Set/Get parameters are already assigned to a document, these are overwritten by calling the *AddProcSetGetParam* method and then the *Save* method.

Structure

Import Parameters

Export Parameters	Description
ID	ID of the Set/Get parameter
Value	Value that the parameter shall have when it is called.

Export Parameters

None.

AddProcImportParam

Definition

Method of the BOR object Message without dialog that adds a parameter in the form of Name, Value for executable documents (compare the SetProcGeneral [Page 37] method). By calling this method several times, several parameters can be added to the document. The parameters can be transferred directly (in the case of function modules, for example) or via the SAP memory (in the case of Transaction with Memory, for example), depending on the execution type (import parameter processing type of the SetProcGeneral method). If the parameters are transferred via the SAP memory, the memory ID must be set using the SetProcGeneral method.

⚠

If parameters are already assigned to a document, this method (followed by Save) overwrites them.

Use

Import Parameters

Export Parameters	Description
Name	Name of the parameter
Value	Value that the parameter shall have when it is called.

Export Parameters

None.

ImportRecipientList

Definition

Method of the BOR object Message without dialog that copies the recipient list of another Message object. The method creates a list with copies of the recipients of the document that serves as a template.

Structure

Import Parameters

Export Parameter	Description
MasterMessage	Handle for the Message object that serves as a template.

Export Parameters

None.

SetSensitivity

Definition

Method of the BOR object Message without dialog that sets the sensitivity of a document.

Structure

Import Parameters

Export Parameter	Description
Sensitivity	Sensitivity of the document The following values can be assigned: **F** - Functional **C** - Company confidential **O** - Standard **P** - Confidential

Export Parameters

None.

Copy

Definition

Method of the BOR object Message without dialog that copies the current object instance. As a result, a new object instance is created that adopts all the attributes, including the recipient list and the execution parameter, from the original. A handle for the new object instance is available as a result parameter.

Structure

Import Parameters

None.

Export Parameters

None.

SetNextSender

For information on the use of this method, see Sending Under Another Name [Page 74].

Recipient

Definition

BOR object type that represents a recipient to which a document (in other words a Message object) is to be sent. This may be, for example, an internal SAP user, an organizational unit or a contact person of a company to whom a standard communication method is assigned. The Recipient object must contain all the information required for sending the document. Besides the address, this also includes the send attributes, for example, send priority or express sending of documents.

⚠

All changes to a specific instance of a Recipient object are at first buffered in the main memory only, in order to improve performance. The changes are copied to the database, or a newly created instance becomes persistent, only once the Save method has been called.

Structure

Attributes of the Recipient Object

Attribute	Description
Value	Recipient address (possibly distributed over Value, Value2, Value3, Value4, Value5)
TypeDescription	Description of the address type
SendExpress	Send attribute *Express*
SendCopy	Send attribute *Send as copy*
SendBlindCopy	Send attribute *Send as blind copy*
NoForwarding	Send attribute *No forwarding*
NoPrinting	Send attribute *No printing*
SendPriority	Send attribute *Send priority*
ToDoByAllRecipients	Send attribute *ToDo by all recipients*
ToDoByOne	Send attribute *ToDo by one recipient* All the recipients of a document can be subdivided into groups. This send attribute has the effect that one member of a group has to process the document. The value of this attribute is the group number.
ReplyRequired	Send attribute *Reply required*
StatusInfoByMail	Send attribute *Status Confirmation by Mail*
NoteType, NoteYear, NoteNumber	ID of the private note that is to be sent to a recipient

Message Object Methods

CreateAddress

Definition

Method of the BOR object Recipient that creates the address part of a Recipient object.

Structure

Import Parameters

Import Parameter	Description
AddressString	Address in string form
Value	Address in structured format
TypeId	Flag for the address type The following address types are available: **B** - SAP User name **P** - Private distribution list **C** - Shared distribution list **F** - Fax number **U** - Internet address **R** - Remote mail address (within an R/3-R/3 System group) **X** - X.400 address **G** - ID of an organizational unit **H** - Name of an organizational unit

Export Parameters

None.

FindAddress

Definition

Method of the BOR object Recipient that finds an address, or better an Address object. The Address object corresponds to a Recipient object without send attributes. The address found is copied to the address part of the Recipient object. The other attributes of the Recipient object remain unchanged.

It is neither possible nor advisable to search for a Recipient object, in other words an address with send attributes.

Structure

Import Parameters

Import Parameter	Description
AddressString	Address in string form
Value	Address in structured format
TypeId	Flag for the address type The following address types are available: **B** - SAP User name **P** - Private distribution list **C** - Shared distribution list **F** - Fax number **U** - Internet address **R** - Remote mail address (within an R/3-R/3 System group) **X** - X.400 address **G** - ID of an organizational unit **H** - Name of an organizational unit

Export Parameters

None.

Copy

Definition

Method of the BOR object Recipient without dialog that creates a copy of the Recipient object. As a result, a handle of the copy is returned.

Structure

Import Parameters

None.

Export Parameters

None.

Edit

Definition

Method of the BOR object Recipient with dialog that allows a Recipient object to be edited.

Structure

Import Parameters

None.

Export Parameters

None.

Save

Definition

Method of the BOR object Recipient without dialog that saves Recipient object. If the object is not yet persistent, that is, if it does not yet exist in the database, an entry is created in the database. Otherwise, the changes are updated in the database.

Structure

Import Parameters

None.

Export Parameters

None.

ImportSendAttributes

Definition

Method of the BOR object Recipient without dialog that copies the send attributes of another Recipient object.

Structure

Import Parameters

Import Parameter	Description
Master	Handle for the Recipient object that serves as a template.

Export Parameters

None.

Delete

Definition

Method of the BOR object Recipient without dialog that deletes a persistent Recipient object.

Structure

Import Parameters

None.

Export Parameters

None.

Expand

Definition

Method of the BOR object Recipient without dialog that breaks down a Recipient object that consists of several recipients. Distribution lists and organizational units are examples of such Recipient objects. The individual recipients, in other words the ‘flat’ Recipient objects, are returned as a table.

⚠

If the Recipient object only contains one recipient, meaning that it is already ‘flat’, a copy of this instance is created using the *Expand* method and returned. If the original Recipient object is no longer required, you should delete it after the Expand using the *Delete* method and continue to work with the copy. In the process, you must check the return code of the *Expand* method.

Structure

Import Parameters

None.

Export Parameters

Export Parameters	Description
ResultTable	Table with the handles for the ‘flat’ Recipient objects

AddNote

Definition

Method of the BOR object Recipient without dialog that creates a private note. In contrast to a general note [Page 35], a private note only applies to the Recipient object, meaning that it is sent to the recipients that the Recipient object groups together. The text for the note is entered as a table and automatically inserted into the database as a document.

Structure

Import Parameters

Import Parameter	Description
NoteText	Table with the text of the note This parameter is mandatory.

Export Parameters

Export Parameter	Description
NoteType	Part of the note ID that specifies the type of the note created
NoteYear	Part of the note ID that specifies the year in which the note was created.
NoteNumber	Part of the note ID that specifies the number of the note created

DeleteNote

Definition

Method of the BOR object Recipient without dialog that deletes the private note that was created using the AddNote [Page 55] method.

Structure

Import Parameters

None.

Export Parameters

None.

SetExpress

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *Express*.

Structure

Import Parameters

Import Parameter	Description
SendExpress	Value for switching on or off the send attribute *Express*

Export Parameters

None.

SetCopy

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *Send as copy*.

Structure

Import Parameters

Import Parameter	Description
SendCopy	Value for switching on or off the send attribute *Send as copy*

Export Parameters

None.

SetBlindCopy

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *Send as blind copy*.

Structure

Import Parameters

Import Parameter	Description
BlindCopy	Value for switching on or off the send attribute *Send as blind copy*

Export Parameters

None.

SetNotForwarding

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *No forwarding.*

Structure

Import Parameters

Import Parameter	Description
NoForwarding	Value for switching on or off the send attribute *No forwarding*

Export Parameters

None.

NoPrinting

Definition

Method of the BOR object Recipient without dialog that can be used to set the send attribute *No printing*.

Structure

Import Parameters

Import Parameter	Description
NoPrinting	Value for switching on or off the send attribute *No printing*

Export Parameters

None.

SetSendPriority

Definition

Method of the BOR object Recipient without dialog that is used to set the send priority.

Structure

Import Parameters

Import Parameter	Description
SendPriority	Value of the send priority

Export Parameters

None.

SetToDoByAll

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *ToDo by all recipients*.

Structure

Import Parameters

Import Parameter	Description
ToDoByAll	Value for switching on or off the send attribute *ToDo by all recipients*

Export Parameters

None.

SetToDoByOne

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *ToDo by one recipient of the group.* All the recipients of a document can be subdivided into groups. This send attribute has the effect that one member of a group has to process the document. The value of this attribute is the group number.

Structure

Import Parameters

Import Parameter	Description
ToDoByOne	Value of the send attribute *ToDo by one recipient of the group*

Export Parameters

None.

SetReplyRequired

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *Reply required*.

Structure

Import Parameters

Import Parameter	Description
ReplyRequired	Value for switching on or off the send attribute *Reply required*

Export Parameters

None.

SetDeliver

Definition

Method of the BOR object Recipient without dialog that is used to set the send status *Received*. The send status is assigned when the send status Received is confirmed. The prerequisite, when sending documents externally, is that the external mail system returns the status. Users can display the send status using the recipient list of the document sent.

Structure

Import Parameters

Import Parameter	Description
Deliver	Value of the send status *Received*

Export Parameters

None.

SetNotDeliver

Definition

Method of the BOR object Recipient without dialog that is used to set the send status *Not received*. The send status is assigned when the send status *Not received* is confirmed. The prerequisite, when sending documents externally, is that the external mail system returns the status. Users can display the send status using the recipient list of the document sent.

Structure

Import Parameters

Import Parameter	Description
NotDeliver	Value of the send status *Not received*

Export Parameters

None.

SetRead

Definition

Method of the BOR object Recipient without dialog that is used to set the send status *Read.* The send status is assigned when the send status *Read* is confirmed. The prerequisite, when sending documents externally, is that the external mail system returns the status. Users can display the send status using the recipient list of the document sent.

Structure

Import Parameters

Import Parameter	Description
Read	Value of the send status *Read*

Export Parameters

None.

SetStatusInfoByMail

Definition

Method of the BOR object Recipient without dialog that is used to set the send status *Mailstatus*. This method can be used to define the type of send status for which the sender receives a message informing them of the status.

Structure

Import Parameters

Import Parameter	Description
StatusInfoByMail	Value of the send status *Mailstatus* The following values are possible: *Space* - never **E** - only if errors exist **W** - for warnings and errors **I** - always

Export Parameters

None.

SetResend

Definition

Method of the BOR object Recipient without dialog that is used to set the send attribute *Resend*. By default, it is not possible to send a document several times to the same recipient. If this send attribute is set, this restriction is removed for the Recipient object

Structure

Import Parameters

Import Parameter	Description
Resend	Value for switching on (**X**) or off (Space) the send attribute *Resend*

Export Parameters

None.

Indirect Connection using Message Control

Use

If your application already sends documents using Message Control [Ext.], you can connect to the Business Communication Interface (BCI) indirectly.

Features

If you connect your application to the BCI using Message Control, not all of the BCI functions are available. You can use the following functions:

- Linking of the Application Object with the Document Sent [Page 12]

  Activities

  The application object has to be a BOR object. A NAST object is not adequate.

  The application BOR object must support the IFNAST interface so that the *ConvertKey* method can be used.

  If the BOR object and the NAST object have different keys, you have to redefine the *ConvertKey* method with the actual conversion. The interface of the function module MESSAGING has been extended to include the BOR object type and the key of the NAST object for this purpose. They are transferred via the following import parameters:

Import Parameters	Description
PI_OBJTYPE	Type of application BOR object
MSG_OBJKY	NAST object key

  Within Message Control, the BOR object key is then determined using the *ConvertKey* method and the object link is created using the BCI.

- Status Confirmation by Mail [Page 16]

  Activities

  No further activities are necessary.

- Status Confirmation by Events in the Application Object [Page 15]

  Activities

  The application object has to have been linked with the document sent (see above).

  The application object must support the interface for the event triggered by the status confirmation.

Indirect Connection using SAPscript

Use

If your application already sends documents using SAPscript, you can connect to the Business Communication Interface (BCI) indirectly.

Activities

In order to access the BCI using SAP*script,* the device type **MAIL** (that is, import parameter DEVICE = **MAIL**) must be set when you open form printing using the function module OPEN_FORM. The formatted document is then automatically transferred to the BCI and sent from there. The recipient is determined by the import parameter MAIL_RECIPIENT. This is (analogous to direct access to the BCI) a Recipient [Page 45] object.

Constraints

Only individual recipients can be specified as Recipient objects for the connection using SAPscript. Therefore, distribution lists, organizational units and so on, cannot be specified. The reason for this is that the formatting of documents by SAPscript is generally recipient-dependent.

Therefore, before calling SAPscript, break down the Recipient object using the Expand [Page 54] method, and call SAPscript for each recipient individually.

Features

If you connect your application to the BCI using SAPscript, not all of the BCI functions are available. You can use the following functions:

- Linking of the Application Object with the Document Sent [Page 12]

  Activities

  Transfer the ID of the application BOR object in the parameter MAIL_APPL_OBJECT.

- Status Confirmation by Mail [Page 16]

  Activities

  Status confirmation by mail is set using the SetStatusInfoByMail [Page 69] method.

- Status Confirmation by Events in the Application Object [Page 15]

  Activities

  Transfer the ID of the application BOR object in the parameter MAIL_APPL_OBJECT.

  The application object must support the interface for the event triggered by the status confirmation.

- Sending Under Another Name [Page 74]

  Activities

  Specify the sender in the import parameter MAIL_SENDER of the function module OPEN_FORM. The ID of a BOR object is expected.

Sample Report

The sample report RSSOKIF2 shows, in a simple example, which calls are necessary in order to send a fax to a fax number with the connection using SAPscript and the BCI. This includes the creation of a Recipient object as a recipient, the break down of the Recipient object into individual recipients using the *Expand* method and the specification of a sender as well as an application object ID.

Sending Under Another Name

Use

With this function it is possible to prevent the current user (SYUNAME) automatically being entered as the sender of a document. A different sender can be specified instead.

The following terms can be differentiated:

- Owner

  This is the user who is responsible for a document. If, for example, a secretary creates a document as a substitute for his or her boss, the boss is the owner and the secretary the creator.

- Sender

  This is the user who is responsible for a sending a document. As a rule, the sender and the owner are the same person. However, this does not have to be the case. The sender is displayed in the transmission information after the document has been sent.

Activities

When creating a document, the required owner must be specified using the import parameter *Owner* of the Create [Page 21] method.

Before the document is sent, the required recipient must be specified using the import parameter *Sender* of the *SetNextSender* method (Sender = Handle for a BOR object, for example, a Recipient object).

The sender can only be set for the next send process, that is, the next *Message.Submit*. The sender has to be set again for further send processes.

Sending under a different internal user (in this example, the model user)

```
* Creating a Recipient object that represents the SAP model
user:
SWC_CREATE_OBJECT recipient ‚RECIPIENT‘ space.
SWC_SET_ELEMENT container ‚AdressString‘ ‚MODEL‘.
SWC_SET_ELEMENT container ‚TypeID‘ ‚B‘.
SWC_CALL_METHOD recipient ‚CreateAddress‘ container.
PERFORM error_handling.
SWC_CLEAR_CONTAINER container.
SWC_CALL_METHOD recipient ‚Save‘ container.
PERFORM error_handling.
```

```
* Creating a document with Owner = ‘Model‘
SWC_CREATE_OBJECT message ‚MESSAGE‘ space.
SWC_CLEAR_CONTAINER container.
SWC_SET_ELEMENT container ‚Originator‘ recipient.
…    "Specify further document attributes (for example, title,
content, …)
SWC_CALL_METHOD message ‚Create‘ container.
PERFORM error_handling.

* Set sender
SWC_CLEAR_CONTAINER container.
SWC_SET_ELEMENT container ‚Sender‘ recipient.
SWC_CALL_METHOD message ‚SetNextSender‘ container.
PERFORM error_handling.
```

➡

The Form routine *error_handling* is responsible for the processing of errors. For example, it issues error messages. After the method call, SY-SUBRC, SY-MSGID, SY-MSGNO, SY-MSGV1, … are set and can be evaluated.

Creating Express Messages in Applications

Purpose

This component allows application developers to create express messages in their R/3 applications. The express messages are displayed to the recipient in a dialog box, irrespective of the transaction he or she is currently working in. Processing can be started directly from the express message.

Features

Extensive express handling with the following functions is available to the applications:

- Any user in any client can receive information via a dialog box.
- You are free to create the title and the text in the dialog box. It is administered via T100. Variable text sections (for example, document titles, and so on) are possible, as with messages.
- An object from the Business Object Repository (BOR) can be executed directly from the dialog box.
- The processing that was started from the dialog box can be executed in an amodal window so that the current work is not disrupted. This means that a new session is started for the processing and the session with the current work remains open parallel to this new session.
- A link to the inbox of the Business Workplace can be offered.
- The appearance of the dialog box can be integrated so that either the dialog box immediately appears the next time the user executes an action or so that it does not appear until the next time the user executes a new logon. You can set this option for each user.
- If a user is not logged on when an express message is created for them, the dialog box appears the next time this user logs on.
- When a user logs on to the R/3 System, express messages are integrated into the dialog box that informs the user about unread documents in his or her Workplace inbox.
- If several messages with the same text accumulate for one user, you can issue a collective message or list the messages individually. If the text is the same but the titles of the individual messages are different, a neutral title is displayed for the collective message.

> Example of a use of the express message:
>
> An object has been created or changed. A group of users is to be informed of this immediately. The next time you execute an action you receive an express message that you can execute directly in order to display the object. If several messages have accumulated, these can be listed in the dialog box and gradually processed by the users.

Constraints

An express message can only be displayed once. Afterwards, it is deleted directly from the database.

Connection to the Interface

Purpose

In order to integrate the express message with the required functions into the application, the function module SO_EXPRESS_FLAG_SET [Page 79] has to be called.

Process Flow

The function module SO_EXPRESS_FLAG_SET breaks down the recipients (for example, several individual recipients, distribution lists or organizational units) into a list of individual users, sets the express flag for each user and stores the necessary data temporarily.

When an action is executed by a user, the task handler checks whether the express flag is set. If this is the case, the task handler starts the processing that then displays the dialog box and makes the other functions available. The application developer, who wants to use the extended express handling, does not have to do this himself.

SO_EXPRESS_FLAG_SET

Definition

Function module from the function group *SOB5* of the development class *SO*. The function module can be called from R/3 applications in order to create express messages and to initiate the required express handling.

Structure

Import Parameters

Import Parameter	Default	Description
client		Client in which the express message is to be triggered. The parameter is optional. If it is not set, the express message is triggered in the current client. ⚠ If a client other than the current client is specified, the recipients have to be specified using the SAP user name (compare table parameter rec_tab).
text_info	*SPACE*	Text that is to be displayed The parameter consists of a structure with the fields - *msgid* (message class) - *msgno* (message number) - *msgv1* (variable text section 1) - *msgv2* (variable text section 2) - *msgv3* (variable text section 3) - *msgv4* (variable text section 4) You have to maintain the texts in the table T100 that is defined for messages. A text is uniquely described by the message class (*msgid*) and the message number (*msgno*). The fields *msgv1* to *msgv4* refer to the text to be output in the express message. As with messages, you have the option of setting variables, for which the placeholders **&1** to **&n** are used. ➡ If the structure *text_info* is not maintained but a default attribute for the object to be executed exists, this is output.

process_param	*SPACE*	Transfer of identification and method of the object to be executed. The parameter consists of a structure with the fields *logsys* (logical system) *objtype* (object type) *objkey* (object key) *method* (method) The object identified in this parameter is executed using the method specified.
procdirect	**X**	Flag indicating whether the object can be executed directly from the dialog box of the express message. If this flag is set, the Execute symbol is displayed in the dialog box. The prerequisite for this is that the structure *process_param* is maintained.
singlentry	*SPACE*	Flag indicating whether a collective message is issued or whether the express messages are displayed individually, if several express messages with the same text have accumulated for one user.
inbox	**X**	Flag indicating whether the express message is to contain a link to the Business Workplace inbox. The link is started using a pushbutton.
popup_title	*SPACE*	Title of the dialog box

Export Parameters

Export Parameter	Description
sent_to_all	Flag indicating whether the express message was successfully sent. Possible values: - **X** all recipients have received the express message - *SPACE* at least one recipient has not received the express message

Internal Table

Internal Table	Description
rec_tab	Internal table for transfer of a list of recipients.

Fields of the internal table *rec_tab*:

Field	Description	Data element	Type	Length
recnam	Recipient name	SO_REC_EXP	CHAR	130

recesc	Recipient type Possible values: **B** - SAP name **P** - Private distribution list **C** - Shared distribution list **H** Organizational unit. The four fields of the internal table form the structure SOOS7.	SO_ESCAPE	CHAR	1
rcode	Return code	As SY-SUBRC		
no_express	Flag indicating whether the recipient is not to receive the express message until he or she executes a new logon.	EXPR_FLAG	CHAR	1

Exceptions

Exceptions	**Description**
NO_RECEIVER_EXIST	None of the specified recipients exist.
OFFICE_NAME_ERROR	No Business Workplace is assigned to the caller of the function module.

Business Appointment Services (4.6B)

Purpose

This component enables developers to integrate the R/3 appointment management in their applications. The Business Appointment Services, as a central data storage service for appointments, also offer applications the integration of other functions, for example, Internet presence, synchronization with external calendar clients or with a free/busy analysis.

Features

You can use the following appointment management functions in your applications by using the Business Appointment Services:

- The basic functions of the R/3 appointment calendar:

  Creating, displaying, changing and deleting appointments (individual and group appointments as well as dominants), searching for existing appointments, searching for free times and reading user defaults for the calendar.

- The authorization concept of the R/3 appointment calendar:

  Individual and general assignment of differentiated authorizations (for example, display authorization or substitute authorization) as well as classification of appointments (public, private, secret).

- Event handling:

  Calling function modules depending on appointment type and action.

- Linking of appointments to the corresponding application object:

  Access to all appointments from the application object / Access to the corresponding application object from the appointment calendar using object relationships [Page 137]

- Various user interfaces:

  SAP GUI, Web browser using the Internet Application Component [Ext.], third-party products such as Microsoft Outlook using calendar integration [Ext.].

Terms

Appointment

An appointment [Ext.] has a start time and an end time in the form of date and time specifications. It is possible to have appointments with a duration of zero. A group appointment [Ext.] encompasses individual participant appointments. Individual appointments are appointments that only one user participates in.

Dominants

A dominant [Ext.] can be defined as a main activity for a specific day. It therefore does not have any explicit time specifications. Only one dominant can be specified for each day. Dominants have a start and an end in the form of date specifications and run from 00:00:00 hrs on the start date to 23:59:00 hrs on the end date. Group dominants do not exist.

Appointment type

The appointment type classifies an appointment or a dominant according to its business purpose, for example, "Customer appointment", "Training" or "Vacation".

It represents a way of grouping the appointments and can be closely linked to application functions in a business application scenario so that the user branches to an application transaction when they double-click on an appointment, for example. Furthermore, access to such appointments from the R/3 appointment calendar interface and external calendar clients can be controlled using Appointment type Customizing (transaction osc1).

Appointment Calendar

The appointment calendar [Ext.] is a central visualization tool for displaying all the appointments of a calendar user. It represents a portal via which the user gains central access to appointment-relevant application data. The data contained in the appointment management is displayed. The calendar can be accessed using transactions ssc0 (other users) and ssc1 (own). The calendars of other users are displayed in display mode if the user calling the calendar does not have maintenance authorization for the appointments in it.

Appointment type Customizing enables you to restrict the actions that are possible on appointments of a certain type. You can, for example, disallow the deletion of an appointment created by the application, from the R/3 calendar interface.

In addition to the appointments from the appointment management, it will also be possible in the future to display external data. However, due to performance reasons, this variant will only be recommended for use in exceptional cases. Furthermore, the current interface will be replaced by a calendar control.

Object link

The object links [Page 137] service is used within the address management to link application objects, for example, sales activities (SD) or events (TEM), to appointment objects. The prerequisite is that the application object exists as a BOR object. The GUIDs of appointments and application objects are then linked. The roles and role relationships used in this process are generic, although internal appointment management default values exist. If the applications' own roles and role relationships are to be used, these have to be maintained beforehand in transaction sm34 (Viewcluster VRBINRELATION). It must be noted that for every newly-defined application role, an additional role relationship must be defined between this role and the appointment role 'APPOINTMNT'. The purpose of these links is to enable the linked objects to both be identified at any time.

From the appointment calendar, the application within the exit function module, in which it has full control, can identify its object that is linked with the appointment, from the interface of the object link service.

If the appointment interface is called from the application, this application will generally select the appointments that are linked with the application object beforehand from the appointment interface, in order to be able to address them specifically.

External calendar client

In addition to the R/3 interface of the appointment calendar, external programs, such as Microsoft Outlook or a Web browser, also have access to the appointment data via RFC calls from the external API. Different appointment type attributes in Appointment type Customizing (transaction osc1) can be used to control the access of these external programs, for example, Non-deletability of appointments of a certain type for external calendars. For further information on the use of external calendar clients, see Appointment Calendar in the Internet [Ext.] and Calendar Integration [Ext.].

Authorizations

Use

The appointment calendar has a differentiated authorization concept [Ext.]. Every user of the appointment calendar can assign read authorization and maintenance authorization to other users for appointments of a certain classification (public, private). If the other user does not have maintenance authorization, they can only call the appointment calendar of the first user in display mode. It is also possible to lock the whole calendar display for certain users or for all users so that they cannot display it either.

In the case of group appointments, it is also possible to choose whether the participants are allowed to edit the description or not. If not, only the organizer of the appointment can change the text. Since there is only one description for each group appointment, changes made to it can immediately be seen by all participants. This feature can be used for discussion papers for a meeting, for example.

Appointment Types and Event Handling

Use

In the R/3 calendar, appointments have an appointment type, which classifies the appointment, for example, "Consultation". Communication between the appointment management and the application is controlled in appointment type Customizing (transaction OSC1) via its attributes. As a result, it is possible to influence the behavior of the calendar interface when actions are executed on appointments of a certain type. For example, deleting or copying can be forbidden. These settings only affect the calendar user interface. The interface works independently of this.

Interaction and data exchange with other applications occur via function modules defined in Customizing for each appointment type, function module exits. They have to suit a specific interface, via which data exchange finally occurs. Appointment types can also exist for which no function module exit is defined.

In specific situations within a calendar scenario, for example when creating or copying an appointment, the function module exit is called with an event. In this case, the internal application data changes then normally occur before the appointment management, depending on a possible returned control command, continues its processing after this call. An application example of event handling can be found in Event Handling for the Appointment Type 'Service Order' [Page 87].

Activities

To integrate event handling in your application, proceed as described in Use of Event Handling [Page 125].

Event Handling for the Appointment Type 'Service Order'

The ordering department has created an appointment of the type *Service order* for a service employee who is working outside the office. The employee concerned selects this appointment in their calendar because they have to move it. By doing this, they access the maintenance interface for service orders. Here, they move their order and, by doing so, they also automatically move their appointment. The advantage of this is that another processor is informed of the change and can notify customers, for example. The service employee exits the application for service orders and returns to their calendar, which automatically displays the changed data.

From a program technical point of view, the application assigned an exit module to the appointment type "Service order" in appointment type Customizing. When the appointment was selected by the service employee, this exit module was automatically called with the corresponding event and caused the jump to the application for "Service orders". After the employee made changes there, these changes, along with the corresponding appointment changes, were executed by the program. The function module then issued a control command to its interface so that no further processing of the appointment could take place via the appointment management because the appointment had already been changed in the background. The calendar only has to refresh the display.

Connection to the Interface

Use

The interface consists of a set of function modules that allow you to create, read, change and delete appointments and to read the participants of an appointment and the fixed values of calendar users. An appointment that has been created is given a globally unique ID, the GUID. The appointment can be identified and manipulated via this GUID. It is possible to read appointments either by specifying a time period and users or by specifying a GUID.

Authorization checks for appointments can be suppressed by a parameter of the interface.

In order that appointments can be identified as belonging to an application object, they can, if required, be linked to the generated object via the interface. This is carried out via the object links [Page 137] service. As a result, it is possible to find appointments linked to the application object at any time via the appointment management interface.

A detailed description of the structures and data types of the individual parameters can be found in the repository.

The interface provides the following function modules:

APPT_CREATE_INTERNAL [Page 89]

APPT_MODIFY_INTERNAL [Page 92]

APPT_DELETE_INTERNAL [Page 96]

DAY_COVERING_GET [Page 98]

DAY_COVERING_AND_PATTERN_GET [Page 101]

SC_GET_APPT_NEIGHBOR [Page 103]

APPOINTMENT_GRP_PARTICIPAN_GET [Page 105]

SCHEDULE_OWNER_ATTRIBUTES_GET [Page 106].

APPT_CREATE_INTERNAL

Definition

Function module that creates appointments and dominants with certain attributes for one or more users. Several-day appointments and dominants are possible. Appointments are created with a unique ID (GUID). If an application object is also specified, the appointment created is linked to it and can then be addressed again at any time by the application so that it can be changed, for example.

Structure

Import Parameters

Parameter	Description
DATE_FROM, mandatory	Date on which appointment starts.
DATE_TO, mandatory	Date on which appointment ends.
TIME_FROM, mandatory	Time at which appointment starts on the day DATE_FROM.
TIME_TO, mandatory	Time at which appointment ends on the day DATE_TO.
TYPE	Appointment type
DESCRIPTION, mandatory if set in Customizing of appointment type attributes [Page 126].	Title of appointment
ROOM	Room in which the appointment takes place.
VISIBILITY	Classification: 1 = public, 2 = private, 3 = secret (not for group appointments)
BODY_ACCESS_ID	Access authorization for description of group appointment. 1 = changeable by all participants 2 = only changeable by organizer.
APPOINTMENT_IS_PLANNED	Not supported
ACTIVATE_SUBSTITUTE	Activation of substitution in HR organization model for appointment time period.
SUPPRESS_MAIL	No sending of notification mails for group appointments This flag should only be used if a group appointment is created for users for whom the organizer has calendar maintenance authorization (for appointments classified with VISIBILITY) because otherwise the participants do not find out about the appointment arranged. It makes sense to use this flag in conjunction with the flag NO_CALENDAR_AUTHORITY_CHECK.

NO_CALENDAR_AUTHORITY_CHECK	The calendar authorization checks are suppressed.
TYPE_SPECIFIC_DATA	Application-specific (technical) information, which is not processed within the calendar but which is made available via the exit functions [Page 125] of the application.
CUSTOMER_NUMBER	Not supported
DOMINANT	Flag indicating whether it should be a dominant. If so, the fields TIME_FROM and TIME_TO are not included.
APPOINTMENT_RULE	Internal
APPL_NEIGHBOR	Application object from the BOR with specifications for at least object type and object key. A role type can also be specified (optional). A link between the appointment to be created and the application object specified via this parameter is created using the object links service. If no role type is explicitly specified for the application, a default role type is used.

Export Parameters

Parameter	Description
ERROR_MESSAGE	Error structure An error output can be displayed using the command message.

Table Parameters

Parameter	Description
PARTICIPANT_LIST	Import parameter This is the participant list for the appointment. At least one user name must be specified. If more than one user name is entered, a group appointment is created for the participants specified, even if the organizer does not have maintenance authorization for the calendars of the participants. This group appointment can be found again at any time by the application, even though it does not appear in any calendar. It is possible to specify private and shared distribution lists.
APPOINTMENT	Export parameter Return of the individual appointment created. The GUID is found in the field APPT_ID. If a group appointment was created, this table should not be included.
APPOINTMENT_PATTERN	Export parameter Return of the group appointment created. The GUID is found in the field GENAPPT_ID.

APPOINTMENT_BODY	Import parameter Appointment description in the form of any long ASCII text.
APPOINTMENT_DATES	Internal
APPOINTMENT_CONFLICT_DATES	Internal

Exceptions

None.

Example call

```
call function 'APPT_CREATE_INTERNAL'
     exporting
          date_from        = sy-datum
          date_to          = sy-datum
          time_from        = '120000'
          time_to          = '130000'
          type             = space
          description      = 'Lunch'
     importing
          error_message    = error_message
     tables
          participant_list = participant_list
          appointment_body = appointment_body.
```

APPT_MODIFY_INTERNAL

Definition

Function module that changes appointments via their unique ID. All parameters specified overwrite the old parameters in principle. Therefore, it is advisable to read the appointment attributes beforehand. Examples for reading appointment attributes can be found in Examples of the Use of Function Modules [Page 107].

Structure

Import Parameters

Parameter	Description
OWNER, mandatory	Owner of the individual appointment to be changed or organizer of the group appointment to be changed.
ID, mandatory	GUID of an individual or group appointment created with APPT_CREATE_INTERNAL.
DATE_FROM	Date on which the appointment that was moved is to start.
DATE_TO	Date on which the appointment that was moved is to end.
TIME_FROM	Time at which the appointment is to start on the day DATE_FROM.
TIME_TO	Time at which the appointment is to end on the day DATE_TO.
TYPE	New appointment type
DESCRIPTION	New appointment title
ROOM	New room in which the appointment is to take place.
TXT_PUB_ID	Internal
VISIBILITY, mandatory	New classification 1 = public, 2 = private, 3 = secret (not for group appointments)
BODY_ACCESS_ID	Access authorization for description of group appointment. 1 = changeable by all participants 2 = only changeable by organizer.
STATUS_TYPE	Not supported
ACTIVATE_SUBSTITUTE	Activation of substitution in HR organization model for appointment time period.

SUPPRESS_MAIL	No sending of notification mails for group appointments This flag should only be used if a group appointment is created for users for whom the organizer has calendar maintenance authorization (for appointments classified with VISIBILITY) because otherwise the participants do not find out about the appointment arranged. It makes sense to use this flag in conjunction with the flag NO_CALENDAR_AUTHORITY_CHECK.
NO_CALENDAR_AUTHORITY_CHECK	The calendar authorization checks are suppressed.
TYPE_SPECIFIC_DATA	Application-specific (technical) information, which is not processed within the calendar but which is made available via the exit functions of the application.
CUSTOMER_NUMBER	Not supported
DOMINANT	Flag indicating whether it should be a dominant. If so, the fields TIME_FROM and TIME_TO are not included.
CHANGED_MODE	Internal
DAY_COVERING_RULE_NEW	Internal
DAY_COVERING_RULE_OLD	Internal

Export Parameters

Parameter	Description
ERROR_MESSAGE	Error structure. An error output can be displayed using the command message.

Table Parameters

Parameter	Description
PARTICIPANT_LIST	Import parameter New participant list
DAY_COVERING_DATES	Internal
DAY_COVERING_DELETE_DATES	Internal
DAY_COVERING_CONFLICTS	Internal
DAY_COVERING_DELETE_CONFLICTS	Internal

APPT_MODIFY_INTERNAL

DAY_COVERING_BODY	Import parameter New appointment description in the form of any long ASCII text.
APPOINTMENT_CREATE	Export parameter Return of the individual and group appointments created. If a group appointment was created by adding participants to an individual appointment, its GUID is found in the field GENAPPT_ID. If an individual appointment was created from a group appointment by removing all the participants except one, its GUID is found in the field APPI_ID.
APPOINTMENT_MODIFY	Export parameter Internal
APPOINTMENT_DELETE	Export parameter Return of the deleted individual and group appointments. The comments under the table parameter APPOINTMENT_CREATE apply here.

Exceptions

None.

Example call

```
call function 'APPT_MODIFY_INTERNAL'
     exporting
         owner                  = 'R_GERE'
         id                     = guid
         date_from              = sy-datum
         date_to                = sy-datum
         time_from              = ‘130000‘
         time_to                = ‘140000‘
         type                   = ‘MEETING‘
         description            = ‘Enjoy Presentation‘
         room                   = ‘Munich‘
         visibility             = ‘1‘
    tables
         participant_list  = participant_list
         day_covering_body = appointment_body.
```

APPT_DELETE_INTERNAL

Definition

Function module that deletes appointments via their unique ID.

Structure

Import Parameters

Parameter	Description
OWNER, mandatory	Owner of the appointment Owner of the individual appointment to be changed or organizer of the group appointment to be changed.
ID, mandatory	GUID of an individual or group appointment created with APPT_CREATE_INTERNAL.
CHANGED_MODE	Internal
SUPPRESS_MAIL	No sending of notification mails for group appointments This flag should only be used if a group appointment is created for users for whom the organizer has calendar maintenance authorization (for appointments classified with VISIBILITY) because otherwise the participants do not find out about the appointment arranged. It makes sense to use this flag in conjunction with the flag NO_CALENDAR_AUTHORITY_CHECK.
NO_CALENDAR_AUTHORITY_CHECK	The calendar authorization checks are suppressed.

Export Parameters

Parameter	Description
ERROR_MESSAGE	Error structure. An error output can be displayed using the command message.

Table Parameters

Parameter	Description
APPOINTMENT	Export parameter Return of the deleted individual appointment. The GUID is found in the field APPT_ID. If a group appointment was deleted, this table should not be included.
APPOINTMENT_PATTERN	Export parameter Return of the deleted group appointment. The GUID is found in the field GENAPPT_ID.

Exceptions

None.

Example call

```
call function 'APPT_DELETE_INTERNAL'
     exporting
         owner             = 'M_DOUGLAS'
         id                = guid.
```

DAY_COVERING_GET

Definition

Function module that reads appointments within a time period. The function module reads the attributes of appointments for one or more users within a specific time period, which is defined by two date entries. All the appointments that fall within the specified time period, including those that just start or end within it, are returned.

Structure

Import Parameters

Parameter	Description
DATE_FROM, mandatory	Date from which the appointments are to be returned.
DATE_TO, mandatory	Date up to which the appointments are to be returned.
NO_CALENDAR_AUTHORITY_CHECK	The calendar authorization checks are suppressed. If this parameter is not set, information, such as the title of the appointment, is hidden, if necessary, according to the authorizations that the inquirer has for the calendar of the other user.

Export Parameters

None.

Table Parameters

Parameter	Description
APPOINTMENTS	Export parameter Return of the individual and group appointments found. The ID of the group appointments is found in the field GENAPPT_ID. In the case of individual appointments, this field is empty and the ID is found in the field APPT_ID.
DOMINANTS	Export parameter Return of the dominants found.
BODY_LIST	Export parameter All the descriptions of the appointments found are returned together in this table. Information as to which lines of this table refer to the description of the appointment for an individual user can be found in the table BODY_PKLIST.

BODY_PKLIST	Export parameter Positioning table for the descriptions returned in the table BODY_LIST. The description of an appointment within APPOINTMENTS or DOMINANTS that can be identified by its GUID can be found in the table BODY_LIST using the information on the start line and the number of lines that is contained in the table BODY_PKLIST. This table also contains information on the number of bytes that a description requires.
OWNER_OR_DISTR_LIST	Import parameter List of users for whom appointments are to be read. At least one user name must be specified. It is possible to specify private and shared distribution lists.

Exceptions

Parameter	**Description**
DATE_HAS_INVALID_FORMAT	Invalid format in the date fields DATE_FROM and/or DATE_TO. Leap years are considered, but 29.02.1999, for example, does not exist.
DATE_INCONSISTENCY	Error: DATE_TO < DATE_FROM

Example call

```
date_from = sy-datum.
date_to   = sy-datum+10.

call function 'DAY_COVERING_GET'
     exporting
          date_from           = date_from
          date_to             = date_to
     tables
          appointments        = appointments
          dominants           = dominants
          body_list           = body_list
          body_pklist         = body_pklist
          owner_or_distr_list = owners.
```

DAY_COVERING_AND_PATTERN_GET

Definition

Function module that reads individual appointments. This function module returns the attributes of one appointment only, which is addressed by its ID.

Structure

Import Parameters

Parameter	Description
OWNER, mandatory	Owner of the individual appointment to be changed or organizer of the group appointment to be changed.
ID, mandatory	GUID of an individual or group appointment created with APPT_CREATE_INTERNAL.

Export Parameters

Parameter	Description
DAY_COVERING	Return of the individual appointment found.
DAY_COVERING_PATTERN	Return of the group appointment found.

Table Parameters

Parameter	Description
DAY_COVERING_BODY	Appointment description in the form of any long ASCII text.

Exceptions

Parameter	Description
DAY_COVERING_NOT_FOUND	No individual appointment found under the ID.
NO_AUTHORITY	Inquirer does not have authorization to read the attributes of the appointment of the other user.
OWNER_NOT_FOUND	User OWNER not found
DAY_COVERING_PATTERN_NOT _FOUND	No group appointment found under the ID.

Example call

```
call function 'DAY_COVERING_AND_PATTERN_GET'
     exporting
         owner             = 'M_STREEP'
         id                = guid
     importing
         day_covering      = appointment
     tables
         day_covering_body = appointment_body.
```

SC_GET_APPT_NEIGHBOR

Definition

Function module that reads linked appointments. This interface module returns the IDs of the appointments that are linked to a specific application object.

Structure

Import Parameters

Parameter	Description
APPL_NEIGHBOR	Application object from the BOR with specifications for at least object type and object key. A role type can also be specified (optional), otherwise a default role type is used.

Export Parameters

None.

Table Parameters

Parameter	Description
APPT_ID	Export parameter Return of the appointment ID found.

Exceptions

Parameter	Description
INTERNAL_ERROR	Internal error
NO_LOGSYS	Own logical system is not defined
NOT_FOUND	Object relationship not found.
UNKNOWN	Unknown error

Example call

```
data: contact type neighbor.

* application object data
contact-objkey   = vbeln.     "ID of the contact
contact-roletype = 'CONTACT'. "predefined in TA sm34, optional

* read related appointments
call function 'SC_GET_APPT_NEIGHBOR'
     exporting
          appl_neighbor  = contact
     tables
          appt_id        = appt_id
     exceptions
          internal_error = 1
          no_logsys      = 2
          not_found      = 3
          unknown        = 4
          others         = 5.
```

APPOINTMENT_GRP_PARTICIPAN_GET

Definition

Function module that reads participants of group appointments using their ID.

Structure

Import Parameters

Parameter	Description
GENAPPT_ID, mandatory	GUID of a group appointment created with APPT_CREATE_INTERNAL.

Export Parameters

None.

Table Parameters

Parameter	Description
PARTICIPANT	Export parameter Appointment participant list

Exceptions

None.

Example call

```
call function 'APPOINTMENT_GRP_PARTICIPAN_GET'
     exporting
         genappt_id  = guid
     tables
         participant = participant_list.
```

SCHEDULE_OWNER_ATTRIBUTES_GET

Definition

Function module that reads the fixed values for a calendar user. If a user has not maintained any fixed values, default values are returned.

Structure

Import Parameters

Parameter	Description
OWNER, mandatory	Owner of an appointment calendar.

Export Parameters

Parameter	Description
OWNER_ATTRIBUTES	Attributes of the calendar (user defaults [Ext.]), for example display time period [Ext.] or factory calendar set [Ext.].

Table Parameters

None.

Exceptions

Parameter	Description
OWNER_NOT_FOUND	User OWNER not found
ATTRIBUTES_NOT_FOUND	Obsolete

Example call

```
call function 'SCHEDULE_OWNER_ATTRIBUTES_GET'
     exporting
         owner            = ‘S_MCQUEEN‘
     importing
         owner_attrubutes = owner_attributes.
```

Examples of the Use of Function Modules

Use

The following examples contain excerpts of programs in which the function modules are called:

Creating a Group Appointment with a Title and a Description [Page 108]

Creating Dominants for Several Days [Page 110]

Creating an Individual Appointment with a Link to an Application Object [Page 111]

Moving an Individual Appointment [Page 113]

Moving a Group Appointment for all Participants [Page 116]

Deleting Appointments [Page 119]

Scenario Meeting Arrangement [Page 120].

Creating a Group Appointment with a Title and a Description.

```
* create a group appointment

* assumption: according to the customizing of the appointment
* type "space" the parameter "description" (the title) is
* obligatory

wa_participant_list = 'M_ROURKE'.
append wa_participant_list to participant_list.
wa_participant_list = 'K_BASINGER'.
append wa_participant_list to participant_list.
wa_appointment_body = '9 1/2 weeks'.
append wa_appointment_body to appointment_body.

* create appointment
call function 'APPT_CREATE_INTERNAL'
     exporting
          date_from        = sy-datum
          date_to          = sy-datum
          time_from        = '120000'
          time_to          = '130000'
          type             = space
          description      = 'Lunch'
     importing
          error_message    = error_message
     tables
          participant_list = participant_list
          appointment_body = appointment_body.
```

Creating Dominants for Several Days

```
* create a dominant for 1 week

date_from = sy-datum.
date_to   = date_from + 6.
time_from = sy-uzeit. "will be overwritten internally by 00:00:00
time_to   = sy-uzeit. "will be overwritten internally by 23:59:00
wa_participant_list = 'J_BOND'.
append wa_participant_list to participant_list.

* create appointment
call function 'APPT_CREATE_INTERNAL'
     exporting
          date_from        = date_from
          date_to          = date-to
          time_from        = time_from
          time_to          = time_to
          type             = 'HOTLINE'
          dominant         = 'X'
     importing
          error_message    = error_message
     tables
          participant_list = participant_list.
```

Creating an Individual Appointment with a Link to an Application Object

```
* create an appointment and relate it to a SD contact

data: contact type neighbor.

wa_participant_list = 'J_FOSTER'.
append wa_participant_list to participant_list.

* application object data
contact-objkey   = vbeln.      "ID of the contact
contact-roletype = 'CONTACT'. "predefined in TA sm34

* create appointment
call function 'APPT_CREATE_INTERNAL'
     exporting
          date_from        = sy-datum
          date_to          = sy-datum
          time_from        = '170000'
          time_to          = '180000'
          type             = 'CONTACT'
          appl_neighbor    = contact
     importing
          error_message    = error_message
     tables
          participant_list = participant_list.
```

Moving an Individual Appointment

```
* change a single appointment of 'R_GERE'
* assumption: the appointment has the id 'guid'

* read the appointment attributes
call function 'DAY_COVERING_AND_PATTERN_GET'
     exporting
         owner             = 'R_GERE'
         id                = guid
     importing
         day_covering      = appointment
     tables
         day_covering_body = appointment_body.

* move the appointment one day forward
appointment-date_from = appointment-date_from + 1.
appointment-date_to   = appointment-date_from.

* modify the appointment
call function 'APPT_MODIFY_INTERNAL'
     exporting
         owner             = 'R_GERE'
         id                = guid
         date_from         = appointment-date_from
         date_to           = appointment-date_to
         time_from         = appointment-time_from
         time_to           = appointment-time_to
         type              = appointment-appt_type
         description       = appointment-txt_short
         room              = appointment-room
         visibility        = appointment-class_id
    tables
         day_covering_body = appointment_body.
```

Moving a Group Appointment for all Participants

```
* change a group appointment
* assumption: the appointment has the id 'guid' and was
* organized by 'M_STREEP'

* read the appointment attributes
call function 'DAY_COVERING_AND_PATTERN_GET'
     exporting
         owner             = 'M_STREEP'
         id                = guid
     importing
         day_covering      = appointment
     tables
         day_covering_body = appointment_body.

* read the participants
call function 'APPOINTMENT_GRP_PARTICIPAN_GET'
     exporting
         genappt_id  = guid
     tables
         participant = participant_list.

* move the appointment one day forward
appointment-date_from = appointment-date_from + 1.
appointment-date_to   = appointment-date_from.

* modify the appointment
call function 'APPT_MODIFY_INTERNAL'
     exporting
         owner             = 'M_STREEP'
         id                = guid
         date_from         = appointment-date_from
         date_to           = appointment-date_to
         time_from         = appointment-time_from
         time_to           = appointment-time_to
         type              = appointment-appt_type
         description       = appointment-txt_short
         room              = appointment-room
        description      = appointment-txt_short
```

Deleting Appointments

```
* assumption: the appointment has the id 'guid'. It could be a single,
* a participant or a group appointment. In the last case the
* corrisponding participant appointments are deleted as well.
* 'M_DOUGLAS' is the owner resp. organizator in the group case

* delete the appointment
call function 'APPT_DELETE_INTERNAL'
     exporting
         owner             = 'M_DOUGLAS'
         id                = guid.
```

Scenario Meeting Arrangement

In the following example, a group appointment is created at midday with two participants. The appointment has a title and there is further information in the description, which can be changed by all participants.

If the organizer has maintenance authorization for the calendars of the other participants, a participant appointment is also created for them. This participant appointment also appears in the return table for created appointments.

```
include ssc_top.

data: participant_list           type standard table of scsparinc,
      appointment_body           type standard table of soli,
      appointment_create         type standard table of scsappt,
      appointment_pattern_create type standard table of scsappt,
      appointment_modify         type standard table of scsappt,
      appointment_delete         type standard table of scsappt,
      appt                       like line of appointment_create,
      organizator                like appt-owner,
      wa_participant_list        like line of participant_list,
      wa_appointment_body        like line of appointment_body,
      wa_appointment_create      like line of appointment,
      error_message              like scserror.

* organizator of the meeting
organizator = 'R_GERE'.

* appointment properties
appt-date_from      = sy-datum.
appt-date_to        = appt-date_from.
appt-time_from      = '120000'.
appt-time_to        = appt-time_from + 3600.
appt-txt_short      = 'Lunch'.
appt-room           = 'MC Donalds'.
appt-bd_accs_id     = c_body_access_part.
wa_participant_list = organizator.
append wa_participant_list to participant_list.
wa_participant_list = 'M_ROURKE'.
append wa_participant_list to participant_list.
wa_appointment_body = 'Bill Gates will also attend'.
append wa_appointment_body to appointment_body.

* create the appointment
call function 'APPT_CREATE_INTERNAL'
     exporting
          date_from           = appt-date_from
          date_to              = appt-date_to
          time_from            = appt-time_from
          time_to              = appt-time_to
```

Unfortunately, the appointment has to be moved. M_ROURKE cannot attend at the new time. He asks Mrs. J_ROBERTS to stand in for him.

The whole group appointment is affected. Only the organizer of the meeting can make the changes. The GUID (ID) is obtained from the return table for group appointments after the appointment has been created (field GENAPPT_ID). All the participants and the normal appointment attributes have to be specified again, even if they are not to be changed (for example, title and room). The parameter VISIBILITY is mandatory.

If the organizer has maintenance authorization for the calendars of the other participants, a new appointment is created for them or the original appointment deleted. These appointments are returned via the return tables, along with the changed appointment of the organizer.

```
* appointment properties
appt-date_from      = sy-datum + 1.
appt-date_to        = appt-date_from.
appt-time_from      = '123000'.
appt-time_to        = appt-time_from + 3600.
refresh participant_list.
wa_participant_list = organizator.
append wa_participant_list to participant_list.
wa_participant_list = 'J_ROBERTS'.
append wa_participant_list to participant_list.

* modify the appointment
call function 'APPT_MODIFY_INTERNAL'
     exporting
         owner              = organizator
         id                 = appointment_pattern_create-genappt_id
         date_from          = appt-date_from
         date_to            = appt-date_to
         time_from          = appt-time_from
         time_to            = appt-time_to
         description        = appt-txt_short
         ROOM               = appt-room
         visibility         = '1'
         body_access_id     = appt-bd_accs_id
    importing
         error_message      = error_message
    tables
         participant_list   = participant_list
         appointment_create = appointment_create
         appointment_modify = appointment_modify
         appointment_delete = appointment_delete.
```

J_ROBERTS cannot attend the lunch now. She deletes her participant appointment. R_GERE receives a mail informing him of this.

The GUID (ID) is obtained from the return tables after the group appointment has been created/changed (field APPT_ID).

```
* find the correct appointment
read table appointment_create where owner = 'J_ROBERTS'
                              into wa_appointment_create.

* delete the appointment
call function 'APPT_DELETE_INTERNAL'
     exporting
          owner         = 'J_ROBERTS'
          id            = wa_appointment_create-appt_id
     importing
          error_message = error_message
     tables
          appointment   = appintment_delete.
```

Use of Event Handling

Use

Event handling and also the exit functions of the appointment management are based on the linkage between appointment types and function module exits [Page 128]. This assignment can be set within transaction OSC1. If no exit function module is specified for an appointment type, appointments of this type do not take part in event handling.

Every appointment has an appointment type and the corresponding function module exit is called with a specific event, depending on the scenario. The appointment structures (for example, APPOINTMENT) also contain the field EXIT_INFO, which the applications can use to store local technical information, such as an employee number, for each appointment created. This field is not evaluated within the appointment management, but is forwarded via event handling to the exits.

Events can be used, on the one hand, to display additional information for an appointment within the appointment processing dialog, for example the customer contact person or the telephone number. On the other hand, they can also control the behavior of the appointment calendar.

Besides a function module exit, appointment types also have other attributes that can be maintained via the above transaction. When actions are executed on an appointment of the type concerned, these attributes influence the behavior of the interface of a user calendar.

Activities

To influence the behavior of the appointment calendar for each appointment type, make settings for the appointment type attributes [Page 126] in transaction OSC1.

In order to integrate application-related functions, create an exit function module [Page 128] and assign this to the required appointment type in transaction OSC1. Various appointment-related events are available. You can react to these using the function modules. The following are examples of exit function modules: Jump to an Application Transaction [Page 129], Calling a Search for Customer Master Records [Page 132].

Appointment Type Attributes

Definition

Attributes that control the communication between the appointment management and the application.

Structure

General Attributes

Key field

This can be any character string, for example 'CUSTOMER'.

Text

Language-dependent text for the appointment type. It appears in the selection field for the appointment type, among others, when appointments are created or changed, for example 'Customer appointment'.

Title is a required-entry field

If this flag is set, a title must be specified for this appointment type. This is useful, for example, with a default appointment type "No appointment type chosen".

Exit function module

A function module, which is called with specific events in different situations, can be specified here. The behavior of the appointment calendar after such a call can be influenced by returned control commands.

Attributes for Maintenance within the R/3 Appointment Calendar Interface

No creating

Appointments of this type cannot be created or copied (that is, another identical appointment cannot be created at a different time point) via the calendar interface.

No changing

Appointments of this type cannot be changed or moved via the calendar interface. When such appointments are selected, the system automatically switches to display mode. The appointments can, however, be deleted.

No deleting

Appointments of this type cannot be deleted or cut via the calendar interface. The appointments can, however, be changed.

Attributes for Maintenance via External Calendar Clients

These attributes have an effect on the external interface of the appointment management.

No creating

The external API does not create the appointment and returns an appropriate error.

No changing

The external API does not change the appointment and returns an appropriate error.

No deleting

The external API does not delete the appointment and returns an appropriate error.

Function Module Exit for the Appointment Management

Definition

Function module for implementing additional program logic in the appointment management.

Structure

Function module exits must have the interface described in the following.

Import Parameters

Parameter	Description
APPOINTMENT_IN	Processed appointment with its attributes.
EVENT	Situation-dependent parameter, for example, ‘CREATE‘ when an appointment is being created.

Export Parameters

Parameter	Description
APPOINTMENT_OUT	Returned appointment, possibly with changed attributes.
ERROR_MESSAGE	This parameter is used for transferring errors. An error output can be displayed using the command message. The behavior of the appointment calendar can also be influenced by certain control commands, for example, suppression of the usual creation dialog if the appointment is to be created by the application within the function module exit. These are transferred using the parameter ERROR_MESSAGE-MSG_VAR1.

Table Parameters

Parameter	Description
TEXT	Table for additional data exchange. The entries in this table are interpreted differently, depending on the event to be processed.

Exceptions

None.

Jump to an Application Transaction

In the following example, a sales activity and the corresponding appointments for the sales representative responsible and the participating sales executives are to be changed at the same time. This change can be made via the application SD-CAS itself. The prerequisite is that the appointments have already been created via this application by simply calling the appointment management interface.

SD-CAS reacts to the events with subroutines, which are to process not only the sales activity but also the appointments. These subroutines are not specified in more detail here. The field EXIT_INFO is used to store and change technical information during appointment management and to evaluate it before it is displayed. This information helps the application to identify the customer for whom the sales activity is taking place.

A jump to the application transaction takes place within the form routine TO_MODIFY. This application transaction changes not only its own sales activity but also the corresponding appointments via the interface. The control variable ERROR_MESSAGE-VAR1 is immediately set to 'EXIT' so that the change dialog of the calendar does not continue to be displayed after the change has been made.

The appointments cannot be moved. Instead, an application-specific error message is output and further appointment processing, which would move the appointment, is suppressed again. Alternatively, a different transaction could be used to move the appointment, as is the case when appointments are changed.

```
function sdcas_schedule_exit.
*"----------------------------------------------------------------------
-
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(APPOINTMENT_IN) LIKE  SCSAPPT STRUCTURE  SCSAPPT
*"                             OPTIONAL
*"             VALUE(EVENT) LIKE  SCSEXIT-EVENT
*"       EXPORTING
*"             VALUE(APPOINTMENT_OUT) LIKE  SCSAPPT STRUCTURE  SCSAPPT
*"             VALUE(ERROR_MESSAGE) LIKE  SCSERROR STRUCTURE  SCSERROR
*"       TABLES
*"              TEXT STRUCTURE  SCSEXITTXT OPTIONAL
*"----------------------------------------------------------------------
-

  move-corresponding appointment_in to appointment_out.

  case event.
    when c_event_to_modify.
      perform to_modify.
      error_message-msg_var1 = 'EXIT'.
    when c_event_to_move.
*     operation not allowed
      message s209(bc).
      error_message-msg_var1 = 'EXIT'.
    when c_event_modify.
      perform modify
        tables   text.
        using    appointment_in-exit_info
        changing appointment_out-exit_info
    when c_event_display.
      perform display
        tables text.
        using  appointment_in-exit_info
    when others
      perform others
        using    event
        changing error_message.
  endcase.

endfunction
```

Calling a Search for Customer Master Data

When creating and changing a customer appointment, you can branch to the search for customer data via the function module CUSTOMER_SELECT. Detailed information on the selected customer, for example, the contact person and telephone number, is displayed in the additional information area of the appointment maintenance. The lines of the table TEXT are filled accordingly. The customer number is stored in the field EXIT_INFO and is used to obtain the detailed information when an appointment is displayed or changed (CUSTOMER_READ). A consistency check using the module CUSTOMER_CHECK ensures that only customer appointments for which a contact person has been specified can be created.

```
function customer_schedule_exit.
*"----------------------------------------------------------------
-
*"Lokale Schnittstelle:
*"       IMPORTING
*"             APPOINTMENT_IN LIKE SCSAPPT STRUCTURE SCSAPPT
*"             EVENT
*"       EXPORTING
*"             APPOINTMENT_OUT LIKE SCSAPPT STRUCTURE SCSAPPT
*"             ERROR_MESSAGE LIKE SCSERROR STRUCTURE SCSERROR
*"       TABLES
*"             TEXT
*"----------------------------------------------------------------
-

  move-corresponding appointment_in to appointment_out.

  case event.
    when c_event_create.
      call function 'CUSTOMER_SELECT'
             exporting
                  customer_number = space
             importing
                  customer_number = appointment_out-exit_info
             tables
                  address_text    = text.
    when c_event_modify.
      call function 'CUSTOMER_SELECT'
             exporting
                  customer_number = appointment_in-exit_info
             importing
                  customer_number = appointment_out-exit_info
             tables
                  address_text    = text.
    when c_event_display.
      call function 'CUSTOMER_READ'
             exporting
                  customer_number = appointment_in-exit_info
             importing
                  error_message   = error_message.
      if not error_message = initial
```

Events

Use

There are different types of events. On the one hand, there are events that enable the application to intervene directly in the behavior of the calendar (**action events**). On the other hand, there are events that are used within the appointment maintenance screen to obtain additional information on an appointment and to display this information in the appointment processing dialogs in a subarea defined for it (**information events**). The information can then be processed, either by double-clicking on this additional info area itself or by using an additional info button that has also been created, or read (in the display mode of the calendar). There is normally a whole sequence of different events within one scenario.

In the case of action events, ERROR_MESSAGE-MSG_VAR1 is used as the interface parameter of the exit function module. A control command (OK-Code) for the calendar is transferred to this parameter. Data exchange for the information events occurs via the table parameter TEXT.

It is recommended that a constant definition of the includes SSC_TOP is used.

Use of Action Events

The following action events are processed when the specified actions are executed:

Control Event	Action
C_EVENT_COPY	Copy appointment
C_EVENT_CREATE(*)	Create appointment
C_EVENT_CUT	Cut appointment
C_EVENT_DELETE	Delete appointment
C_EVENT_TO_MODIFY	Change appointment
C_EVENT_TO_MOVE	Move appointment
C_EVENT_TO_DISPLAY	Display appointment
C_EVENT_MARK	Select appointment
'CHECK'	Check appointment when creating it

(*)occurs both as an action event and as an information event.

The control parameter ERROR_MESSAGE-VAR1 currently supports the value 'EXIT'. If it is set whilst events are being processed, the calendar does not continue to behave as normal after the exit module has been called. In this case, the dialog box *Change appointment* would no longer be displayed after the event C_EVENT_TO_MODIFY, for example, and a confirmation prompt would not appear after the event C_EVENT_DELETE. Further processing by the calendar should be suppressed if it has already be carried out by the application within the exit.

Use of Information Events

The following information events can be used:

Control Event	Action

C_EVENT_CREATE(*)	Select an appointment type for the appointment; Create additional information.
C_EVENT_MODIFY	Change additional information.
C_EVENT_DESCRIPT	Obtain additional information.
C_EVENT_DISPLAY	Create or update calendar. Display appointment. Display additional information.
C_EVENT_READ	Obtain additional information in display mode.

(*)occurs both as an action event and as an information event.

The table TEXT is interpreted for the different information events as follows:

Event	Text			
	Row 1	**Row 2**	**Row 3**	**Row 4**
C_EVENT_CREATE	3-line info text in additional info box			
C_EVENT_MODIFY	3-line info text in additional info box			
C_EVENT_DESCRIPT	Title Additional info box	Text Additional info button	Icon Additional info button	Quick info Additional info button
C_EVENT_DISPLAY	3-line info text in additional info box			
C_EVENT_READ	3-line info text in additional info box			

Relationship Service (BC-SRV-GBT)

Use

Documents in the business process are not independent of one another. For example, a request for a quotation is followed by a quotation and then a delivery. The need therefore arises for an overview of all the documents occurring in a business process or related in some other way. You also want to be able to jump back and forwards between such documents in the R/3 System.

In addition, you want to know what type the relationships are which are between the individual documents. In the above example, this is "consists of" both times, whereby the offer occurs once in the role as the successor document and in the other occasion in the role of the preceding document. You therefore want to reduce the relationship to a simple **model**, through which time and causal connectivities between documents become apparent.

In order to reach these targets for any document, the object relationship service has been created. With this it is possible to link objects from the *Business Object Repository* (BOR) with one another. The relationships are modeled and are written by the application via the calling of a specific interface by the object relationship service API. All that the application has to transfer to this API is the Business Object from the BOR and the data from the object relationship model.

Some generic object services [Page 147] use the API from object relationships internally. Through this, the relationships that you write via the API are also displayed.

Prerequisites

Only BOR objects can be linked.

Activities

- Before you use the relationships you must make a Relationship model [Page 138].
- You use the API from object relationships [Page 143] to read, write or delete relationships.

Object Relationships Model

Definition

Set of regulations by which objects can be linked and which describes the meaning of the relationships.

Use

Before you link your BOR objects to one another, you must integrate the types in a relationship model. As a result

- "Your" relationships are displayed later comprehensibly
- Meaningless relationships are not possible
- several object types can share the same model

The object or document types "request for quotation" and "quotation" are in a unique predecessor-successor relationship. That is the meaning of the relationship which is defined in the model. Thus, a request for a quotation can never be linked as a successor of a quotation. The predecessor-successor relationship also applies to other object types.

Structure

You maintain your model in the view cluster VRBINRELATION. It consists of

- Roles (more precise: role types): You must describe what meaning an object has in a specific relationship. An object type can occur in several roles and vice versa several object types can be assigned to a specific role type.

  The document type "quotation“ can appear in both the role of the predecessor and also in the role of the successor document.

  A person can appear in both the role of the superior and also in the role of the employee.

  By assigning its type to a role, a BOR object can be linked in this role. Relationships are always written between objects in specific roles.

- Relationship types: You must describe what meaning the relationship has between two objects (in their roles). There are two *descriptions* of a relationship type according to whether the relationship from object A with respect to object B or vice versa shall be displayed.

  A request for a quotation *causes* a quotation. Oppositely, a quotation *consists of* a request from a quotation.

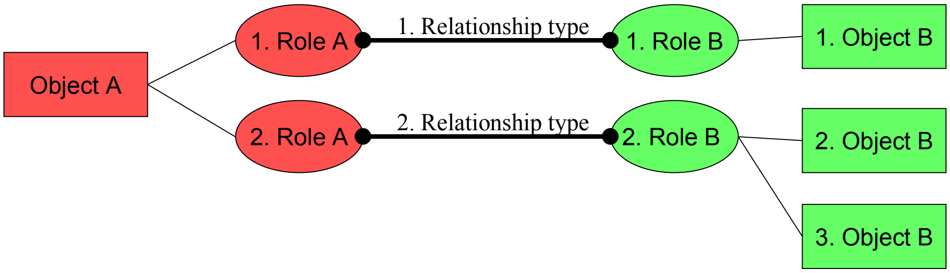
  Object A *belongs to* object B. Inversely, object B *contains* object A.

You must define the application table in which the relationships of this type should be written. It must have the same structure as in table SRGBINREL.

The model restricts the assignment of roles to relationship types: Two different role types can always only be in a specific relationship to one another, represented with a specific relationship type.

- Relationship type attributes [Page 142] (optional): You may want to describe your relationship through more parameters than just through its type.

Diagram: Example for a model



Creating a Model (Example)

Prerequisites

You have already created the object types that you want to link to one another in the *Business Object Repository*.

The default attribute should be entered (in the basic data) for these BOR objects: Its describing text is displayed in the relationship display. Otherwise, only the object key is displayed.

Procedure

1. Choose *System* → *Services* → *Table maintenance* → *View cluster maintenance*.
2. Enter the view cluster VRBINRELATION and choose ✎ *Maintain.*
3. Check whether role types are available in the system which are applicable for the object types to be linked. If not, double-click on *Role types* and choose *New entries*. Enter your role types and sufficiently descriptive texts.
4. Select a role type. Double-click on *Object types for role types* in navigation and assign the relevant object types to the role type. Proceed in exactly the same way for your other role types.

   You can also not enter any object type. Then all BOR object types in this role can occur.
5. Double-click on *Binary relationship types* and choose *New entries*. Also enter the *Application table* here in which the relationships should be written. If you want to create attributes (last step) you must set the corresponding indicator.

   Also enter sufficiently descriptive texts here (as "*semantics*“)! Only in this way is the display of the relationships subsequently comprehensible.

   By using the *Cardinality* of the role you enter whether several or only one object in this role can be linked with an object in the other role. For *Cardinality* 1: *Cardinality* N - relationship can therefore link any number of objects in the second role with one object in the first role.

   The name of the application table should be **`<component>SRGBINREL`**, with **`<component>`** = **`SD`**, **`MM`**,..., according to application.
6. Choose 💾.

   Your data is put in a transport request.
7. *Optional*: Position the mouse on *Attributes for binary relationship types* and enter the attribute, the application table (for *attribute mode* P = "persistent") and the structure, by which the data part of the application table should be read. Assign your *relationship type* (= "binary relationship type“) to the attribute. Choose 💾.

   The name of the application table should be **`<component>POSFORM`**, with **`<component>`** = **`SD`**, **`MM`**,..., according to application.

   The structure of the application table is described in the section Attributes of relationship types [Page 142].

Example 1: You model a predecessor-successor relationship of request for quotation and quotation. You make the following entries:

Model parameter	**Value**
Role type	**Predecessor, successor**
Object type for role type	`BUS2030` (request for quotation) to **`predecessor`**, `BUS2031` (quotation) to **`successor`**
Relationship type (binary relationship type)	**`VONA`**, *Cardinality* currently 1 (each request for quotation is assigned exactly one quotation and vice versa), Application*table* **`SDSRGBINREL`**, *Attributes available* not selected

Example 2: You must model a predecessor-successor relationship of two lists, whereby the second list consists of the first list filtered. As an attribute you enter the filter used.

Model parameter	**Value**
Role type	**Predecessor, successor**
Object type for role type	`LIST` for **`predecessor`** and **`successor`**
Relationship type (binary relationship type)	**`VONA`**, *Cardinality* `1` for role type **`predecessor`**, **`N`** for **`successor`** (via the use of different filters, several different lists can exist from one list), Application*table* **`SDSRGBINREL`**, *Attributes available* selected
Attribute	**`ATTRVONA`**, *Attribute table* **`SDFILTER`**, *Attribute mode* **`P`**, *Data type* of *attribute* **`T`** (table), *Format* of *attribute* **`SRABREL<component>`**

Attributes of Relationship Types

Definition

Refinement of a relationship type.

Use

Attributes are optional: Therefore if you want to describe a relationship in a more detailed way than just by its type, you must assign one or several attributes to the relationship type in the model.

Object A (a list) was created from object B (also a list) via filters. The connection type is *created from* and the *Filter* used is the attribute.

Structure

You must define persistent attributes, like relationship types, in a special application table. The attributes are transferred as character strings of length 250. They are split via a structure into single fields, which you define as the *Attribute format* in the relationship model. You can define a structure with a single field if you only want to transfer a single attribute. When writing a relationship you can transfer a whole table of attributes with the API (*Data type* **T**). For example, you can therefore transfer several fields with the attribute *Filter*, which were filtered accordingly.

The application tables must have the following structure:

Field	**Key**	**Field type**	**Meaning/Use**
CLIENT	x	MANDT	
RELATIONID	x	BINRELID	Unique relationship number (GUID)
POSNO	x	POSNO	For attribute tables (*Data type* **T**): Number of line in the table For one-line attributes (*data type* **S**) you must leave the field empty.
BRELTYP		BINRELTYP	Relationship type
ATTRIBUT		RELATTRIB	
.INCLUDE		<your attribute format>	Your structure is at most 250 bytes long

At present, attributes can only be written by the API. Therefore, if you want to display your attributes, for example, in a separate browser, then you must write a function module which reads the attributes from the table and displays them correspondingly.

Virtual (not saved in the database) attributes are not yet supported.

The Object Relationships API

Use

You use relationship service function modules to write, read or delete relationships. Special function modules are available for display purposes.

Prerequisites

You have modeled [Page 138] your relationships.

Features

The tables contain names and short descriptions of the function modules. Further information, in particular on the interface, is available in the function module documentation in the system.

Writing, reading, deleting

Function module	Description
BINARY_RELATION_CREATE	Creates a relationship between two objects, including the attributes
BINARY_RELATION_DELETE	Deletes a relationship, including the attributes
SREL_GET_NEXT_RELATIONS	Reads the relationships of an object. The maximum interval can be entered.
RELMOD_READ_BINRELTYPE	Reads the texts that describe the relationships from the model. Supplement to SREL_GET_NEXT_RELATIONS.
SREL_GET_NEXT_NEIGHBORS	Reads the objects linked to the object entered. The maximum interval can be entered.
RELMOD_READ_ROLETYPE	Reads the texts that describe the functions from the model. Supplement to SREL_GET_NEXT_NEIGHBORS.

Simple list display

SREL_DISPLAY_LIST_OF_NEIGHBORS	Displays the next neighbor of an object (distance is zero)
SREL_DISPLAY_LIST_OF_RELATIONS	Displays the relationships of an object. The maximum interval can be entered. You can go to the display of an object from the list.

Create and Display Relationship (Example)

Create

The examples refer to the Modeling examples [Page 140]

Example 1: When creating a quotation the relationship is created for the preceding document "request for quotation“.

```
Data : newdocument type borident.

New document-objtyp  =     'BUS2030'.

New document-objkey     =     new document number.

Reference document-objtyp     =     'BUS2031'.

Reference document-objkey     =     Reference document number.

call function 'BINARY_RELATION_CREATE'

      export      obj_roleA         =     Reference document

                  obj_roleB         =     New document

                  relation type     =     'VONA'

...

COMMIT WORK.
```

Example 2: You link two lists, whereby the second list consists of the first list filtered according to the customer number . The application table is called Attrib_filter. The fields filter and field are defined in the INCLUDE structure.

```
      Data : newdocument type borident,

    i_Attrib_filter type Attrib_filter occurs 10 with header line.

      New document-objtyp     =     'LIST'.

      New document-objkey     =     new document number.

      i_Attrib_filter-attribut      =     'ATTRVONA'.

      i_Attrib_filter-posno         =     '0001'.

      i_Attrib_filter-filter        =     'KUNDENNUMMER'.

      i_Attrib_filter-field         =     Customer number.

      call function 'BINARY_RELATION_CREATE'

            export      obj_roleA         =     Reference document

                        obj_roleB         =     new_document

                        relation type     =     'VONA'

            tables      binrel_attrib     =     i_Attrib_filter.

      ...

      COMMIT WORK.
```

Display

Example 1: The simple API list display outputs all documents linked to the request for quotation (up to distance 1), in particular also the quotation.

```
DATA: OBJECT TYPE BORIDENT

....

OBJECT-OBJTYPE = 'BUS2030'.

OBJECT-OBJKEY = new document number.

CALL FUNCTION 'SREL_DISPLAY_LIST_OF_RELATIONS'

    EXPORTING

        object    = OBJECT

        max_hops  = 2     <-----The distance is enough to cover the next neighbor but one.

    EXCEPTIONS

        NO_LOGSYS = 1

        OTHERS  = 2.

IF sy-subrc <> 0.

    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

        WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.

ENDIF.
```

Presenting Objects in other Systems

Use

For the presentation of links with objects in different systems, a local proxy object is required which recognizes the methods for presenting the "removed object“. This also applies to other logical systems, therefore also if only the client is different.

Features

The API display module [Page 143] checks whether the object method REMOTEDISPLAY is available. If not, it triggers an information message. If it is, the object can be displayed from the list by double-clicking.

Activities

You must define a local proxy object type with the same name as the remote object type. It must support the following methods:

- SETLOGICALDESTINATION - In this method you address the remote object by setting the logical system as an attribute of the proxy object:

```
 begin_method SetLogicalDestination changing container.

   SWC_GET_ELEMENT CONTAINER ' LogicalDestination'
LogicalDestination.
 end_method.
```

- REMOTEDISPLAY - In this method you program
  - The connection to the logical system
  - The presentation of the remote object

Generic Object Services (BC-SRV-GBT)

Use

There are general functions that you want to execute with various documents (in general: object). Some of these functions, like for example archiving, require detailed knowledge about the object and can therefore only be realized for a specific object type with a great deal of programming. On the other hand, the **generic object services** only need to know that the object exists to be effective.

> Entering a note for a document is a generic object service. The object is the application document.

Prerequisites

In order to be able to take part in the generic object services, a document must belong to an object type in the Business Object Repository (BOR). The application must also publish the document as an application object. For further information see Application Connection to Object Services [Page 167].

Features

For Users:

List of Object Services [Page 148]

For Developers:

Application Connection to Object Services [Page 167]

Create New Object Service [Page 176]

See also:

BC - Relationship Service [Page 137]

Activities

With the exception of the object history, all services are offered as pushbuttons in a separate "toolbox“ window. The symbol appears on the top left of the screen as a pushbutton in an application that uses the object services. The toolbox appears when the button is pressed.

List of Object Services

The following list shows all possible object services. Not all object services are available - it depends on objects that are already active. Individual object services can also only be accessed using a submenu, so that the symbol is not displayed in the toolbox.

Adding a PC Document (to the Storage System) [Page 149]

Adding a PC Document (in the R/3 System) [Page 151]

Entering a Note with an Object [Page 152]

Sending an Object with a Note [Page 153]

Starting a Workflow for an Object [Page 154]

Linking an Internet Address [Page 155]

Entering a Bar Code [Page 156]

Tracing Events for the Object (Subscription) [Page 158]

Telephony [Page 161]

Displaying Object History [Page 162]

Displaying Workflows for the Object [Page 163]

Displaying Transmissions for an Object [Page 164]

Displaying Relationships [Page 165]

Displaying Attachments [Page 166]

Adding a PC Document (to the Storage System)

Use

You use this to store a PC document with SAP ArchiveLink and link it to your application document (your object).

Prerequisites

As well as the general prerequisites (see BC – Generic Object Service [Page 147]), the following Archive Link specific prerequisites also apply:

- A storage system is connected
- You have defined a content repository [Ext.] (IMG activity Maintain Content Repositories [Ext.], transaction OAC0)
- You have maintained document types with the status "active“ (IMG activity Maintain document types [Ext.], transaction OAC2).
- You have defined Relationships [Ext.] for your object type with the status "active" (IMG activity Maintain relationships [Ext.] transaction OAC3)

Procedure

1. In your object display, open the submenu with the symbol 🗋 in the Toolbox [Page 147].
2. Choose *Store business document.*
3. Choose your document type from the selection window by double-clicking on it.

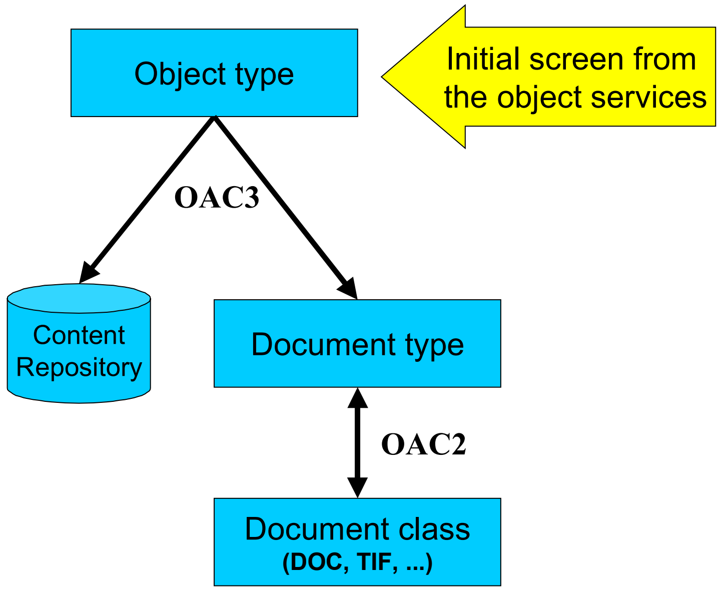
   The document **type** is assigned a specific document **class** (see the graphic below), for example TIF for PC screens or DOC for Word documents.
4. Select your PC document in the window *File name*. Also use the F4 Help. Make sure that you select a document type that you defined in the previous step.

   Your document is stored. To that end, a copy with a technical name is created which is placed in the storage system that you defined in Customizing.

Result

Your PC document is stored. This can be reached using the Attachment List [Page 166] for your object.

Connection of SAP Archive Link objects, maintenance transactions and object services


Object type
Initial screen from the object services
OAC3
Content Repository
Document type
OAC2
Document class
(DOC, TIF, ...)

Adding a PC Document

Use

This enables you to link a PC document with your object.

Prerequisites

The general prerequisites [Page 147] apply.

Procedure

5. In your document display (your object), open the submenu with the symbol 🗋 in the Toolbox [Page 147].
6. Choose *Create attachment*.
7. In the next screen select the directory and the file (therefore the PC document) that you want to copy into the SAP System and link with your application document.
8. Choose *open*.

    The document is stored in the Business Workplace and linked to your object.

Result

You have added a document that you can display using the Attachment List [Page 166]. The original PC document is not deleted from the PC.

Entering an Object with a Document

Use

You enter either a note that can be seen by all users or a personal note for your application document (your object).

Prerequisites

The general prerequisites [Page 147] apply.

Procedure

1. In the object display choose in the Toolbox [Page 147]
   - **either** personal note **or**
   - *Create Note* (in - submenu) for the notes that should be generally viewable.
2. You write your note in the next screen, which is stored as a Business Workplace document of type RAW.

   ⇨

   The relationship service is called for this service because notes are Business Object Repository (BOR) objects. Generally, an application program can link any BOR objects with one another via an API. For further information see Object relationships [Page 137].

Result

You have created a note that you can display using the List of relationships [Page 165].

Sending an Object with a Note

Use

You use this to send notes to other users within the SAP System which contain attachments with links to your document (your object). When the attachment is displayed, the object is opened for the recipient.

Procedure

9. In your object display, open the submenu with the symbol in the toolbox.
10. Choose *Sending object with a note*.
11. You enter the notes in the next screen. Enter the recipient(s) of the document with the note. To send choose .

Result

The recipients receive the note as a short message in their inbox. The message contains a link to the object as an attachment. By double-clicking you go to this default method. That can, for example, be the display method.

You can likewise display a sent note [Page 164] from the source object.

Starting a Workflow for an Object

Use

This is used for starting workflows that use the object. All suitable workflows automatically determine the system and are offered for execution.

Prerequisites

The SAP System contains active workflows that fulfill the following prerequisites:

- They must be allowed to start the workflow.
- The workflow is only allowed to have the object as an obligatory import parameter.

⚠

Tasks that can be executed in the dialog are not included.

Procedure

1. In your object display, open the submenu with the symbol ▶ in the toolbox.
2. Choose *Start workflow*.

   All active workflows which fulfill the prerequisites above are displayed.
3. Select the workflow that you want to start.

For example, the display of a standard order can concern a workflow, within which a further standard order is stored.

Linking an Internet Address

Use

This enables you to link your document (object) with an Internet address.

Procedure

1. In your object display, open the submenu with the symbol 🗋 in the Toolbox [Page 147].
2. Choose *Add external document (URL)*.
3. Enter the Internet address in the next screen.

   Normally it will involve an http protocol hence the address begins with **`http://`**.

Result

You have linked an Internet address to your object which you can select using the Attachment list [Page 166].

Entering a Bar Code

Use

You can assign a bar code to your application document (your object) for late storage with bar code [Ext.]. A scanned document that contains this bar code can therefore be automatically stored via bar code recognition.

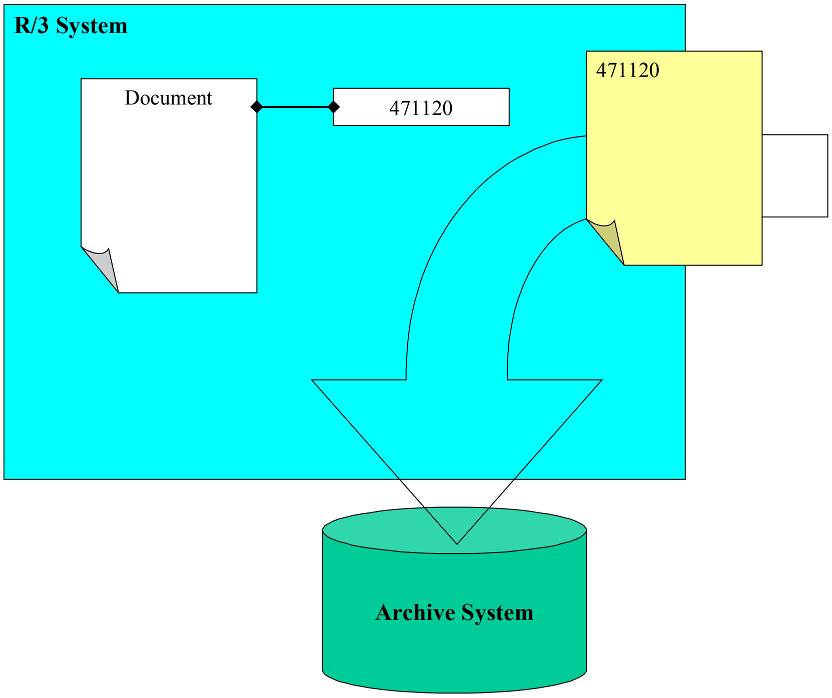
Prerequisites

- A storage system is connected
- Storing with bar code is activated for your object type (IMG activity activate barcode storage [Ext.], transaction OAC5)
- You have defined Relationships [Ext.] for your object type with the status "active" (IMG activity Maintain relationships [Ext.] transaction OAC3)

Procedure

12. In your object display, open the submenu with the symbol 🗋 in the toolbox.
13. Choose *Enter bar code.*
14. Enter a bar code for your document.

Late, automatic content repository with bar code recognition and generic object services



Tracing Events for the Object (Subscription)

Use

If you subscribe to a document (an object) you are informed of relevant events by mail. In the standard system this is changes to objects and deletion of objects. You can store these events or also add further events via the Settings [Page 159].

Prerequisites

The general prerequisites [Page 147] apply.

If necessary, the subscription service must be explicitly activated for your object type and event (see settings [Page 159]).

Procedure

1. In the object display choose 🔔 *Change subscription* in the Toolbox [Page 147].

   You are entered as the subscriber of the object. If you are already a subscriber, you can terminate your subscription in this way.

Result

For example, as soon as changes are made to the object, you receive a corresponding message in the *Business Workplace*. You can define the message text and title yourself using the Settings [Page 159].

Configuring Subscription Service

Use

You determine the events for the object types that the subscription service should be available for and enter the message text.

Without table entries, the service is available for both events CHANGED (document has been changed) and DELETED (document has been deleted) and all object types.

Procedure

1. Go to the view maintenance (transaction SM30) for the table SGOSSUB.
2. Check or change the requested *Object type* and the *Events*. Also see the F1/F4 Help. Descriptions of the object types can be found in the *Business Object Builder* (transaction SWO1).
3. Do not set the indicator *Subscription* if the service is not to be generally available for the event.

   You want to switch off the object service for purchase orders, that is, there should not be a pushbutton offered in the Toolbox [Page 147]. You must also switch it off for the events CHANGED and DELETED as they are automatically available. The simplest way to do this is to enter an asterisk in the field *Event* for the object type **BUS2012** (and naturally remove the indicator).
4. Determine a title for the mails to the subscribers.

   You can transfer SAP*Script* parameters (within two '&' characters):

   - Object attribute. The parameter is called <OBJECTTYPE>.<ATTRIBUTE>
   - The object key (document number). The parameter is called OBJKEY
   - The special default attribute. The parameter is called ATTRIBUT
   - System fields, for example SY-DATE

   You want to subscribe to events for purchase orders and requests and to be able to see from the mail title whether a purchase order or a request has been changed and which purchase order or request this is. Also define the title

   "The purchase order &OBJKEY& has been changed"

   for object type **BUS2012** and a corresponding title for the object type **BUS2032**.
5. You can determine a text for the mails to the subscribers. The text must exist as *general text* in the document maintenance (transaction SE61).

You can transfer the same SAP*Script* parameters as the title. An example is the general text SGBT_DEF_SUB.

You want to see the release group, an attribute for the corresponding object type, in the mail text for the purchase order. Also define the text in the document maintenance.

"The purchase order &OBJKEY& has been changed. It belongs to the release group &BUS2012.ReleaseGroup&."

and enters these for object type `BUS2012`.

Telephony

Use

From the SAP System outwards you use your telephone system. For example, you can initiate, accept or transfer calls. For the full functional scope see the SAPphone Documentation [Ext.].

Prerequisites

SAPphone [Ext.] is installed in your system.

Procedure

Choose ☎ *Telephony* in the toolbox.

In the next screen enter the number of your partner and start the call.

Displaying Object History

Use

The object history is a log of the documents (objects) that you have recently viewed or edited. It contains the access time and a link to the objects. For example, you can therefore go directly to the object that you edited yesterday

Procedure

1. Choose *System* → *Object history*

   The objects edited last are displayed in time order in a list.

2. If required, you can go to an object by double-clicking on it.

Displaying Workflows for an Object

Use

All workflows that are running or have been executed, in which your document (object) has already been edited, are displayed. You can search in the database or in archives.

➡

When searching in the archive you should use the SAP archive info system. You must also activate [Ext.] the archive info system SAP_BO_2_WI_001 (from the central archiving transaction SARA via *Infosystem* → *Customizing*) and create [Ext.] it (via *Infosystem* → *Status*).

Procedure

4. In your object display, open the submenu with the symbol ▶ in the toolbox.

5. If you want to search in the database in the SAP System, you must choose *Workflow Overview*.

   The corresponding workflows are displayed. You do not need to execute the following steps.

6. If you want to search archives, you must choose *Archived Workflows*. The search report selection screen is displayed in the next screen, in which your object type and its key is already entered. For further information see *Help* → *Application Help.* Start the report.

   If the SAP archive info system is connected (see above), the archives available are searched immediately. Otherwise you must still explicitly select the archive that is to be searched for workflows.

Displaying Transmissions for an Object

Use

You can display all transmissions that refer to your document (your object). This for example can be:

- Notes that have been sent with the generic object services [Ext.]
- Message objects (for example notes) that were generated directly from the application program
- Message objects that were generated with Message Control

Prerequisites

The general prerequisites [Page 147] apply. If the application program has generated a BOR object MESSAGE directly, it must enter the object ID in the import parameter *Reference Type*.

Procedure

15. In your document display, open the submenu with the symbol in the menu of the services available.
16. Choose *Sent Documents.*

Result

As well as the documents sent, the number of recipients is also displayed and the number who have already read the documents.

Displaying Relationships

Use

This service is used to display relationships that already exist (for example, with other application documents). This does not apply to Business Workplace attachments (for example, PC documents) which can be found in the Attachment List [Page 166].

Prerequisites

The general prerequisites [Page 147] apply.

Procedure

2. In your document display (your object) choose *Display Relationships* in the Toolbox [Page 147].

   The list of existing relationships is then displayed. You can arrange this list in columns (double-click on the column or the button). Other typical list functions are available, for example exporting as a spreadsheet (using) or printing (using).

3. If required, you can select an object from the list and display further relationships by using .

4. By selecting or double-clicking you can go to the display of any object from the list.

Displaying Attachments

Use

In contrast to the relationship list, BOR objects are not displayed here, but rather Business Workplace Attachments (stored/not stored in an archive). These are mostly PC documents.

Prerequisites

The general prerequisites [Page 147] apply.

Procedure

5. In the display of your document, choose *Attachment list* in the menu of available services.

   The list of existing relationships is then displayed. You can arrange this list in columns (double-click on the column or the button). Other typical list functions are available, for example exporting as a spreadsheet (using) or printing (using).

6. By selecting or double-clicking you can go to the display of any object from the list.

Application Connection to Object Services

Purpose

Every application that will use at least one of the object services can access this through a small number of program steps. Here, it can decide

- which object services it will offer
- how the object services should be offered

Process Flow

- Some generic object services use the object relationships. The BOR IDs are linked. Your document (document header, item...) must be available and released as an object type in the BOR. Also see the workflow documentation Create object type [Ext.].
- Calling the services depends on whether or not your object is unique:
  - In your application the object is unique. Then you **publish** it and generate an instance of *cl_gos_manager*. You must put the object ID in the export parameter of the type *borident*:

    ```
    data:                 go_myobject type ref to
    cl_gos_manager,

                          ls_object type borident
    ```

    …

    ```
    create object go_myobject
                          exporting is_object = ls_object
                                                no_commit = ‚x‘
                          exceptions     others = 1
    ```

    If the parameter *no_commit* is set, you enter that your application starts the *Commit Work* command. That is also the default value. If you explicitly transfer the parameter empty, the object services create the *Commit Work*. However, your data should be consistent when called because the database changes can no longer be reversed using *Rollback*.

    If you do not know which object type your application document has, choose *Tools* → *Business Workflow* → *Development, Business Object Builder, Business Object Repository, Repository Browser* and choose *All object types*. Search for your application document in the outputted list using *Edit* → *Search*. For further information see *Business Object Repository Browser* in the workflow documentation

  - Which object the services are to take up is not decided until the process flow of the transaction. To avoid having to program several publication calls, you should in this case publish a callback [Page 169].

➡

You can always only publish one object. If you publish many times, then it is always the case that only the last publication is valid.

For multiple publication, you can use the old function module SWU_OBJECT_PUBLISH. A window is then displayed when calling the services where the published objects can be selected. You should avoid this additional hurdle for the user if possible.

- In the standard system, the services are displayed in their own window (the "toolbox") which appears when the user presses the button. The following possibilities are available for more specific displays:
  - You do not want to offer a separate window but rather the services in the context menu ("right mouse button“). In this case you do not publish anything, but rather request a context menu for your object [Page 171].
  - You want to offer the toolbox directly [Page 174]. Also in this case you do not publish.
- All available services are offered in the standard system. You can restrict [Page 172] the services available or also only offer one service [Page 173].

Publishing Callback

Use

You use the callback to return the responsibility for determining the object to generic object services. For this, the services call a method that you have implemented. At runtime, your application only publishes the callback.

Procedure

1. Implement the interface method `if_gos_callback~get_object` in a local class:

```
class lcl_myclass definition.
      interfaces if_gos_callback.
      …
endclass.

class lcl_myclass implementation.
      method if_gos_callback~get_object.
      …
      endmethod.
endclass.
```

2. Program the callback (more precise: the publication of the callback)

```
data: go_myobject type ref to cl_gos_manager,
      lo_callback type ref to lcl_myclass
…
create object lo_callback.
…
create object go_myobject
      exporting   io_callback = lo_callback
      exceptions  others = 1
```

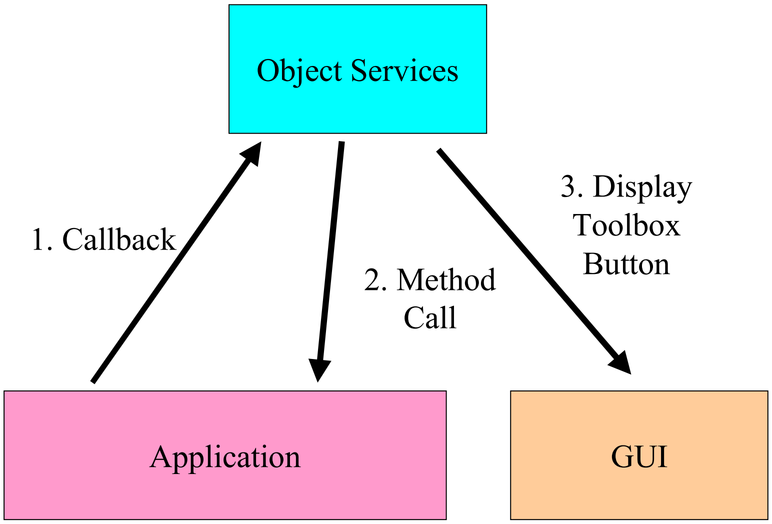
Result

At runtime, the following now happens:

1. Your application publishes the callback and provides an instance of your local class *lcl_myclass*.

2. As soon as the user demands the menu of available services, method *get_object* is called from your local class and thus determines the current object.

3. The 🧰 button for the toolbox is inserted at the top of the screen (GUI).



Offering the Context Menu

Use

Instead of a toolbox in a separate window, you can insert the service in a context menu that can be displayed by positioning the cursor on the object and right mouse-clicking.

Procedure

1. When generating an instance of *cl_gos_manager* you do not transfer an object identifier (or callback class). No object is published:

   ```
   data:     go_myobject type ref to cl_gos_manager.
   …
   create object go_myobject.
   ```

2. You do not transfer the object until you request the context menu:

   ```
   data:   ...,
               lo_menu type ref to cl_ctmenu,
               ls_object type borident.
   …
   call method go_myobject->get_context_menu
               exporting     is_object = ls_object
               importing     eo_menu = lo_menu.
   ```

   The import parameter *lo_menu* contains a reference to the context menu.

3. You can append the context menu to various *controls*. Normally you must write an *event handler* which forwards the menu entries selected by the user. In the *event handler* call a method from *cl_gos_manager* for generic object services:

   ```
   call method go_myobject->dispatch_menu_command
                importing ip_fcode = lp_fcode
                                    io_menu = lo_menu.
   ```

   You only have to request *lo_menu* here if you have requested more than one context menu within the same instance from *cl_gos_manager*.

Restricting Available Services

Use

As far as possible, you should avoid users being offered pointless services for your object. You can restrict the services in the standard menu window (the "toolbox") or also in a context menu [Page 171].

Procedure

When generating an instance from *cl_gos_manager* enter a selection table for the constructor, with which you can enter intervals or exclude specific services. For information on possible entries see the Documentation for Selection Tables [Ext.].

You want to forbid Notes [Page 152] for an object.

```
data:   go_myobject type ref to cl_gos_manager,
   lt_services type tgos_sels,
   ls_service type sgos_sels,
   ls_object type borident.
…
ls_service-sign = ‚E‘.
ls_service-option = ‚EQ‘.
ls_service-low = ‚NOTE_CREA‘.
append ls_service to lt_services.
ls_service-low = ‚PERS_NOTE‘.
append ls_service to lt_services.

create object go_myobject
             exporting      is_object = ls_object

   it_service_selection = lt_services
             exceptions others = 1.
```

Starting Only One Service Directly

Use

You only want to offer one service for your object. The service is started directly, that is, not only when the button is pressed.

Procedure

1. When generating an instance from *cl_gos_manager* you do not enter an object ID. Thus the object is not published:

```
data:     go_myobject type ref to cl_gos_manager.

…

create object go_myobject.
```

2. The name of the service requested (from table SGOSATTR) provides you with the method *start_service_direct*, for example for the service Displaying Attachments [Page 166]:

```
data:     ...,

     lo_container type ref to cl_gui_custom_container,

     ls_object type borident.

…

call method go_myobject->start_service_direct

     exporting     ip_service = ‚VIEW_ATTA‘

                             is_object = ls_object

                             io_container =
lo_container.
```

You can enter a control container for specific services in the parameter *io_container*. The services that allow this option can be found in table SGOSATTR.

Offering the Toolbox Directly

Use

You don't want the services to be available only when the button is pressed, but rather immediately.

Procedure

3. When generating an instance from *cl_gos_manager* you do not enter an object ID. Thus the object is not published:

```
data: go_myobject type ref to cl_gos_manager.

…

create object go_myobject.
```

4. When calling directly, you transfer the object ID as a parameter:

```
data: ...,
      lo_container type ref to cl_gui_custom_container,
      ls_object type borident.
…
call method go_myobject->display_toolbox
            exporting         is_object = ls_object
                                    io_container = lo_container.
```

The parameter *io_container* is optional: If you do not transfer a reference to a container, then the toolbox is started in a separate system like in the standard system.

Calling Services During Creating or Changing

Use

Create

The key is not yet known when creating an object. However, many services need the key, so you must subsequently hand it in.

Change

When changing an object, you must ensure that an implicit database commit is set after every screen change.

Procedure when Creating

1. When generating an instance of *cl_gos_manager* set the constructor parameter *ip_no_instance* and leave the object key empty:

```
data: go_myobject type ref to cl_gos_manager,
      ls_object type borident
…
ls_object-objtype = ‚BUS0815‘
create object go_myobject

exporting
            is_object = ls_object
                        ip_no_instance = ‚X‘
      exceptions  others = 1
```

2. Since your object is persistent, you must subsequently hand in the key:

```
ls_object-objkey = my_key.
call method go_myobject->set_id_for_published_object
      exporting         is_object = ls_object.
```

Creating New Object Service

Purpose

A new object service should be, by definition, generally usable.

Prerequisites

The object must be entered in the Business Object Repository (BOR).

Process Flow

You must execute the following steps:

- Develop your service (as a function module, for example).
- Your new service must be enclosed as the method *execute* in a separate class. The class inherits this instance method from the abstract class *cl_gos_service*. Your service is the composite definition of the method.
- Enter your class in the table SGOSATTR. Also use the table view maintenance (SM30) and see the F1 documentation.
- Finally enhance the documentation in the Knowledge Warehouse for the documentation for your new service. For this you use the Knowledge Workbench.

Result

Your service is displayed if the application is connected to generic object services [Page 167] and it is not explicitly excluded [Page 172].

Post Processing Framework (BC-SRV-GBT)

Purpose

The Post Processing Framework (PPF) provides applications with a uniform interface to any output media (print, fax, XML...). It automatically generates outputs from document data (for example delivery notes or order acknowledgments). The outputs are determined by an individually configurable or self-programmable determination technology.

Additionally, PPF provides uniform output administration. There is a status management and a processing protocol for each output.

The PPF is the object-oriented successor of output control.

Integration

The PPF is found in Basis and can be connected by the applications.

Features

If there is an output from a document posting, the following steps are performed in PPF:

- Determination
- Possible editing of output proposals
- Output processing

Connection of PPF to the Application

Purpose

If you want your application to use the PPF, you must perform a number of programming steps. The connection is object-oriented using the classes. However, your application itself does not need to be programmed object-oriented.

Process Flow

- Create the necessary classes
- If the standard technology is not sufficient, you must define your determination technology.
- Make your application and the determination technology known to the PPF in Customizing.

Result

Your application can now perform outputs using the PPF.

Creating Application Class

Use

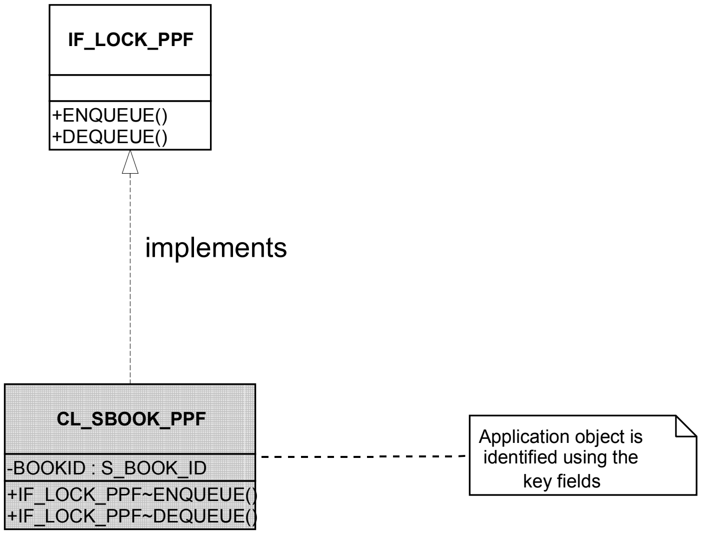
The PPF expects a persistent class in order to be able to handle different types of application objects.

In most cases the application object is not implemented as a persistent class, or it is a BOR object. You must then create a persistent proxy class that does nothing other than refer to the actual application object. Then the application object key appears as a proxy class attribute.

Procedure

1. Create the class in the Class Builder.
2. Implement the interface IF_LOCK_PPF in the class and thus both methods for locking and unlocking the application object.
3. Implement a method that determines the application object key.

   In the example, the attribute BOOKID is set.



Generation of the object at runtime:

DATA:

* reference to application/proxy object

```
    appl_object TYPE REF TO cl_sbook_ppf,

* create a persistent application object via the class agent of its co-class
* object services create 3 service classes for any persistent class
* ca class contains all persistency services, for example, creation of persistent object
  CLASS ca_sbook_ppf DEFINITION LOAD.
  appl_object ?=
    ca_sbook_ppf=>agent->if_os_factory~create_persistent( ).

* set key fields of application object
  CALL METHOD appl_object->set_carrid( sbook-carrid ).
```

Creating Partner Class

Use

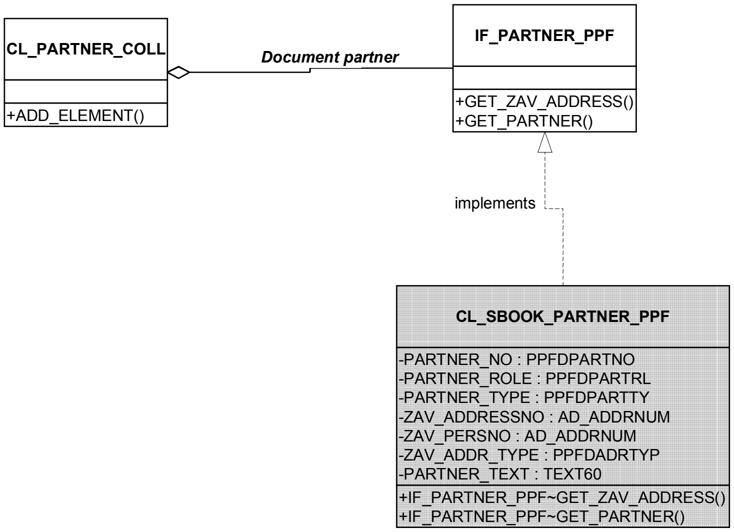
You create a partner class for the document partner. The document partners are collected together and transferred to the PPF. The partner collection class already exists.

Procedure

1. Create the class in the Class Builder.
2. Implement the interface IF_PARTNER_PPF in the class. There you must determine the partner.

   A partner consists of a partner number, -function and -type (optional field) as well as the data allocated from the central address management: Person number, address number and -type. You must provide this data for every document partner.

   In the example, exactly one partner is transferred to the collection at runtime.



```
DATA:
*  reference to partner object
   partner TYPE REF TO cl_sbook_partner_ppf,
*  reference to partner collection
   partner_coll TYPE REF TO cl_partner_coll_ppf,
* create partner collection
  CREATE OBJECT partner_coll.

* create first partner object
  CREATE OBJECT partner
  EXPORTING ip_partner_role  = 'LF'
            ip_partner_no    = '1234567890'
            ip_partner_text  = 'vendor Hibbert'
            ip_zav_addressno = '0000015762'
            ip_zav_persno    = '0000015763'
            ip_zav_addr_type = '3'.

* append partner to partner collection
  CALL METHOD partner_coll->add_element( partner ).
```

Creating Context Class

Use

The context class encapsulates all necessary application data for the PPF:

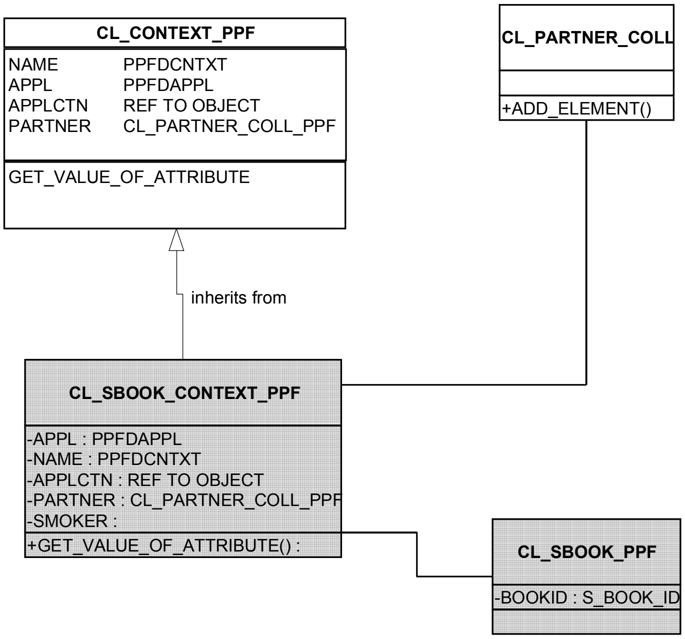
- Application name
- References to application object (proxy object) and partner collection
- All attributes that are required for output determination (for determination technology see the section "determination and grouping of outputs“)

Procedure

1. Create the context class in the Class Builder.
2. Redefine the method GET_VALUE_OF_ATTRIBUTE of the class CL_CONTEXT_PPF: At the same time simply copy the source code from the template and reinsert it in the redefined method. This step is necessary so that the method is executed for your context class itself and has access to its attributes.

   The method serves as dynamic attribute access, which is not yet supported in the ABAP language. Dynamic attribute access is important for generic determination technology.

   In the example, the attribute "smoking or non-smoking" is important.



Fill the context instance at runtime:

```
DATA:
*   reference to context object
    context TYPE REF TO cl_sbook_context_ppf,

* set context attribute
  context->applctn = 'PPF_DEMO'.           ”declared in Customizing [Page 197]
  context->name    = 'DEMOCONTEXT01'.      ”declared in Customizing
  context->appl    = appl_object.
  context->partner = partner_coll.

* additional context fields
*.these fields are evaluated by a determination logic
  context->SMOKER = sbook-smoker.
```

Creating Processing Class

Use

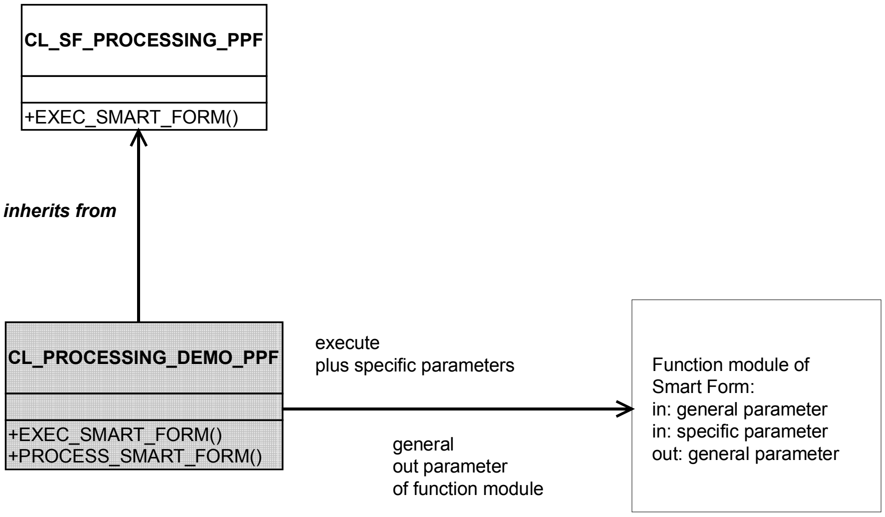
If Smart Forms are used for the output, you must program a processing class. The PPF reduces the user's work load as much as possible.

Smart Forms are function modules whose interface consists of a generally application specific part that is dependent on a particular Smart Form. The PPF cannot directly call the function module. However, it fills the general part of the interface.

Procedure

3. Create the processing class in the Class Builder. It must inherit the processing method EXEC_SMART_FORM from the PPF class CL_SF_PROCESSING_PPF.

   This method serves as a copy template and sets the general parameters of the Smart Forms function module. You only have to fill the specific parameters.



4. You must set the language of your Smart Form. If no entries are made, the logon language is used.
5. If your application object involves a BOR object, you can link the output (Smart Form) with your BOR object in this way. To do this, you must set the necessary data BOR-ID, logical system and object type for the relationship service.
6. The return parameters are evaluated centrally by the PPF. You can add your own entries to the processing log and should also do this so that application specific processing routines are likewise logged. For this purpose, you must transfer a handle to an application log as a

parameter: ip_application_log. Entries can be made using the interface of the service class CL_LOG_PPF:

```
MESSAGE i015(sppf_media) WITH ip_smart_form.
    CALL METHOD cl_log_ppf=>add_message
      EXPORTING
        ip_problemclass = '1'
        ip_handle       = ip_application_log.
```

The copied processing method is extended in the example using the parts marked with colors.

```
METHOD process_smart_form .

* function name
  DATA: function_name TYPE rs38l_fnam,
        dummy(254)    TYPE c.

* get the function name for this smart form
  CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
       EXPORTING
            formname           = ip_smart_form
*           VARIANT            = ' '
*           DIRECT_CALL        = ' '
       IMPORTING
            fm_name            = function_name
       EXCEPTIONS
            no_form            = 1
            no_function_module = 2
            OTHERS             = 3
            .
  IF sy-subrc <> 0.
*   add an error message to processing protocol
    MESSAGE i015(sppf_media) WITH ip_smart_form.
    CALL METHOD cl_log_ppf=>add_message
     EXPORTING
        ip_problemclass = '1'
        ip_handle       = cp_application_log.
    EXIT.
  ENDIF.



*------------------- get application specific data----------------------
  DATA:
    appl_object TYPE REF TO cl_sbook_ppf,
    ...
    bookid TYPE sbook-bookid,
    ...
    smoker  TYPE sbook-smoker.
    ...

* cast interface reference to real object reference
  appl_object ?= io_appl_object.
* get key fields of application object
  bookid = appl_object->get_bookid( ).
* we have all the keys, we can get any data we want
  SELECT SINGLE smoker
    INTO  smoker
    FROM  sbook
    WHERE bookid = bookid.
*--------------------------------------------------------------------

*---------- is_mail_appl_obj -------------------------------------------
* fill this parameter if your application object is a BOR object
* the output (smart form) will be connected with the BOR object
* is_mail_appl_obj-LOGSYS    =
* is_mail_appl_obj-OBJTYPE   =
* is_mail_appl_obj-OBJKEY    =
* is_mail_appl_obj-DESCRIBE  =
*----------------------------------------------------------------------


*-----------language of smart form-------------------------------------
* determine here the language of the smart form and set it
* default language is the system language
* is_control_parameters-langu = language_of_my_smart_form.
*----------------------------------------------------------------------


* call function to process smart form
 CALL FUNCTION function_name
    EXPORTING
      archive_index        = is_archive_index
      archive_parameters   = is_archive_parameters
      control_parameters   = is_control_parameters
      mail_appl_obj        = is_mail_appl_obj
      mail_recipient       = is_mail_recipient
      mail_sender          = is_mail_sender
      output_options       = is_output_options
      user_settings        = ip_user_settings

*-------- additional fields have to be filled by the application--------
      ip_bookid            = bookid
      ip_smoker            = smoker
*----------------------------------------------------------------------


    IMPORTING
      document_output_info = es_document_output_info
      job_output_info      = es_job_output_info
      job_output_options   = es_job_output_options
    EXCEPTIONS
      output_canceled      = 1
      parameter_error      = 2
      OTHERS               = 3
      .
```

```
  IF sy-subrc <> 0.
*   add an error message to processing protocol
    CASE sy-subrc.
      WHEN 1.
        MESSAGE e016(sppf_media).
      ...
    ENDCASE.
    CALL METHOD cl_log_ppf=>add_message
      EXPORTING
        ip_problemclass = '1'
        ip_handle       = cp_application_log.
  ENDIF.

* get error table
  CALL FUNCTION 'SSF_READ_ERRORS'
       IMPORTING
            errortab = et_error_tab.

ENDMETHOD.
```

Determination and Merging of Outputs

Use

The determination delivers templates for outputs if determined conditions are met by the application data

Merging is therefore always necessary if existing, unprocessed outputs, have to be mixed with newly found outputs. This is the case if an existing document is changed, for example. As the data has changed, outputs that were found when creating the document can be omitted or can be found again.

Integration

The application data is transferred to attributes of the Context Class [Page 183]. The determination technology defines which conditions must be met by what application data.

Prerequisites

The necessary classes are created:

Application Class [Page 179]

Partner Class [Page 181]

Context Class [Page 183]

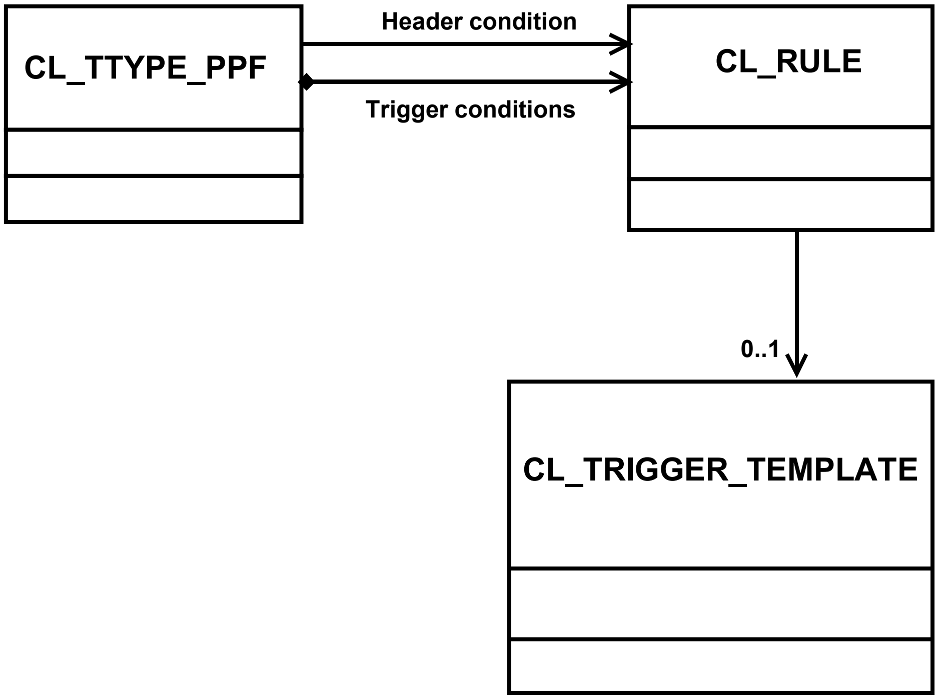
Processing Class [Page 185]

Features

Determination

A condition is encapsulated by a rule. You can think of the rule as a filter of the application data that controls which data is finally released to the condition. A template for an output is appended to the rule. It is returned as soon as the condition (and thus the rule) is met. Header conditions are also conceivable, that is, lower conditions are only checked as long as the header condition is met. There is no output proposal behind the header condition itself.

The determination technology depends on the output type. Thus various output types can use various determination technologies. The determination technology itself manages one or several rules.



Tools generally usable for condition evaluations should be used for determination. In Message Control condition technology was used for this purpose. Currently, the PPF does not offer a generally usable determination. In the future, it is planned to integrate the workflow condition editor in the PPF.

As a transition solution the PPF is offered with its own method based condition evaluation. The conditions are implemented as methods. The application therefore defines a pool of method conditions.

Merging of Outputs

There must be a logical procedure for merging the new and old outputs. Presently the PPF offers a standard logic that only allows one unprocessed output for each output type. You can program your own logic (see the unit "Extensibility").

Programming Your Own Determination

Use

If the standard determination technologies are not sufficient, you must write your own logic for the determination. This logic is the implementation of a PPF interface.

Procedure

Create a class in the Class Builder that implements the interface IF_DETERMINATION_PPF and its methods. The new determination is then integrated and usable in the framework. You can make selections in the output type with the value help in Customizing.

Interface methods:

GET_PERSISTENCY_TABLE

Delivers the table in which persistent data for the determination object is stored. The information is necessary for the compilation of the transport request in Customizing.

DETERMINE

The actual determination method receives as an import parameter a reference to a context object, the output type and a handle to a determination protocol that the determination should fill. Using this information the determination returns one or several output templates.

Parameter:

IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context class
IP_TTYPE	Importing	Type	PPFDTT	Output type
IP_DETLOG	Importing	Type	BALLOGHNDL	Log handle
RO_TRIGGER_TEMPL_COLL	Returning	Type Ref To	CL_TRIGGER_TEMPL_COLL_PPF	Collection of output templates

Programming Your Own Merging

Use

Various logics are offered in the standard system for merging outputs. If these are not sufficient, you can add your own logic, in which you implement an interface.

Procedure

Create a class in the Class Builder which implements the interface IF_MERGE_PPF.

Interface methods:

GET_PERSISTENCY_TABLE

Delivers the table in which persistent data for the determination object is stored. The information is necessary for the compilation of the transport request in Customizing.

MERGE

Called after determination in order to merge newly found outputs with possible existing outputs using determination.

Parameter:

IO_NEW_TRIGGER_COLL	Importing	Type Ref To	CL_TRIGGER_COLL_PPF	Newly found outputs
IO_OLD_TRIGGER_COLL	Importing	Type Ref To	CL_TRIGGER_COLL_PPF	Existing outputs
IP_DETLOG	Importing	Type	BALLOGHNDL	Log handle

RO_TRIGGER_COLL	Returning	Type Ref To	CL_TRIGGER_COLL_PPF	Merged outputs
MERGE_SINGLE				
Called after manual generation of an output in order to merge this with possible existing outputs.				
Parameter:				
IO_NEW_TRIGGER	Importing	Type Ref To	CL_TRIGGER_PPF	New output
IO_OLD_TRIGGER_COLL	Importing	Type Ref To	CL_TRIGGER_COLL_PPF	Existing outputs
RO_TRIGGER_COLL	Returning	Type Ref To	CL_TRIGGER_COLL_PPF	Merged proposals

Processing Using the Output Medium

Use

The compilation of output proposals in outputs is described as processing. Outputs always use an output medium, for example printing.

Integration

After determination, processing is triggered at a selected time.

Prerequisites

There are current output proposals for processing.

Features

Currently the following are supported using Smart Forms:

- Print
- Fax
- Mail

XML, EDI and triggering of workflow events is planned.

Activities

If the standard media are not sufficient, you can add your own output media at any time using the interfaces available. By implementing the interfaces, you automatically integrate the media without coding changes being necessary within the PPF.

Performing Customizing

Use

In Customizing (transaction SPPFC) you make your classes known to the PPF and define the determination of the outputs: You allocate your application to the possible output types (for example delivery note, order acknowledgment and so on). You must define a determination technology for each output type (see section "Determination and Grouping of Outputs“).

Prerequisites

You have made the necessary classes definitions:

Application Class [Page 179]

Partner Class [Page 181]

Context Class [Page 183]

Processing Class [Page 185]

Procedure

1. Go to Customizing by calling transaction SPPFC.
2. Define the application

   An application that is to use the PPF must be entered in the Customizing of the PPF. Its name and a description are simply entered in the initial screen.
3. Define the context

   Using the context you transfer all relevant application data to the PPF. As the data items relevant for the determination are always context class attributes, they should be the same or similar within a context. If this cannot be reached with a context, you can also allocate several context classes to your application (for example, you can group all outputs of orders in a context).
4. Define the output type for the context

   Various output types may be assigned to a context. An output type is the smallest business unit that should be outputted – for example, all delivery notes should belong to the output type "delivery note", whether they are now printed, faxed or otherwise outputted.
5. Enter the details for the output type.

   You can maintain various configurations for an output type:

- Processing time (immediately when posting the document, later using a selection report). This processing time can be overridden by the Determination [Page 190].

  The following possibilities exist for processing:

  Processing with document posting

In this processing type, outputs are processed immediately after document posting, therefore after a COMMIT WORK. Processing is triggered implicitly by the PPF.

For example: After posting the order, an order confirmation is sent immediately by mail.

Later processing

When later processing is involved, the application itself must trigger processing of the outputs. This usually takes place using a selection report which selects the outputs to be processed and triggers their processing. The PPF provides such a report (RSPPFPROCESS or transaction SPPFP). The report can also be triggered in the background (without a user interface).

For example: All faxes should be sent at night in a batch run.

Processing in the dialog

Outputs can be generated here without the document already being posted. This processing type must be declared as permitted in Customizing.

For example: During maintenance of a contact, an invitation letter should be printed out.

- Partner function: Default function of the partner to whom the output goes – is created when an output is created manually
- Can be sent many times (states whether the output can be repeated)
- Changeable (should you still be able to make manual changes after automatic determination?)
- Executable in the dialog (the output can take place during document editing, although no posting has taken place)
- Selection of determination technology that should be used for this output type (see the unit below on Determination)
- Selection of a technology for merging outputs: See the unit below on Determination
- Sort field 1 – 3: Application specific data that is transferred into the context class. The display or processing can be sorted according to this data

6. Allocate the media to the output type

   Outputs always take place using an output medium. Currently Smart Forms are supported as print, fax or mail. Support of XML, EDI and triggering of workflow events is planned.

   You add your own configuration that can be overridden by the determination at runtime for every medium.

   Enter the name of the Smart Form used in the Smart Forms media and a processing class with a processing method that you have programmed (see the section "Processing Class").

Example: Programming Calling of PPF

Use

The following documentation shows all the steps necessary for starting the PPF. The example from the development class SPPF_DEMO is displayed.

Prerequisites

You have carried out the following steps:

Creating Application Class [Page 179]

Creating Partner Class [Page 181]

Creating Context Class [Page 183]

Creating Processing Class [Page 185]

Performing Customizing [Page 197]

Procedure

1. Data declarations

```
DATA:
*  reference to application/proxy object
   appl_object TYPE REF TO cl_sbook_ppf,                ”application defined class
*  reference to context object
   context TYPE REF TO cl_sbook_context_ppf,            ”application defined class
*  reference to partner object
   partner TYPE REF TO cl_sbook_partner_ppf,            ”application defined class
*  reference to partner collection
   partner_coll TYPE REF TO cl_partner_coll_ppf,        ”PPF defined class
*  reference to PPF manager (interface to PPF services)
   manager TYPE REF TO cl_manager_ppf,                  ”PPF defined class
*  determination protocol
   determination_protocol TYPE balloghndl.
```

2. Get an instance of class CL_MANAGER_PPF. This class displays the interface to the PPF. All service methods are called here.

```
* get an instance of PPF manager
  manager = cl_manager_ppf=>get_instance( ).
```

3. Generate the application object and set the key fields so that the actual application object can be found/generated again. If the application object involves a BOR object, then its ID is set here. In our case we set the key fields of table SBOOK in order to be able to access the entry later.

```
* create application object
  CLASS ca_sbook_ppf DEFINITION LOAD.
  appl_object ?=
    ca_sbook_ppf=>agent->if_os_factory~create_persistent( ).
```

```
* set key fields of application object
  CALL METHOD appl_object->set_bookid( sbook-bookid ).
          . . .
```

4. Generate an object of the context class

```
* create context object
  CREATE OBJECT context.
```

5. Generate a partner collection that can include several partner objects. The partner collection represents the document partners

```
* create partner collection
  CREATE OBJECT partner_coll.
```

6. Generate a partner object and append it to the collection. You must carry out this step for every document partner.

```
* create first partner object
  CREATE OBJECT partner
  EXPORTING ip_partner_role  = 'LF'
          ip_partner_no    = '1234567890'
          ip_partner_text  = 'vendor Hibbert'
          ip_zav_addressno = '0000015762'
          ip_zav_persno    = '0000015763'
          ip_zav_addr_type = '3'.
```

```
* append partner to partner collection
```

```
  CALL METHOD partner_coll->add_element( partner ).

* create another partner object
  CREATE OBJECT partner
  EXPORTING ip_partner_role  = 'WE'
           ip_partner_no    = '0987654321'
           ip_partner_text  = 'goods recipient Smith'
              ip_zav_addressno = '0000015762'
           ip_zav_persno    = '0000015763'
           ip_zav_addr_type = '3'.

* append partner to partner collection
  CALL METHOD partner_coll->add_element( partner ).
```

7. The context object that encapsulates all information for the PPF is filled. The name of the application and the context that were defined in Customizing are transferred. The reference to the application object is transferred and the partner collection as well. Additionally, the application specific fields are set to the contexts that are relevant for the determination. Conditions for an output are defined in these fields.

```
* set context attribute
  context->applctn = 'PPF_DEMO'.
  context->name    = 'DEMOCONTEXT01'.
  context->appl    = appl_object.
  context->partner = partner_coll.

* additional context fields
        context->SMOKER = sbook-smoker.
     . . .
```

8. The determination is started. The application receives a determination protocol as a return value that displays how many outputs were found and which outputs were not found and why. The protocol is not persistently stored by the PPF itself. It is always created dynamically at runtime.

```
* start PPF
  CALL METHOD manager->determine
    EXPORTING io_context  = context
```

IMPORTING ep_protocol = determination_protocol.

9. COMMIT WORK

   The PPF uses the persistence services of Object Services. These services only start running after a COMMIT WORK, that is, a COMMIT WORK must take place at the end so that the generated outputs are written to the database.

Result

The steps above mean that at runtime the PPF is started and is provided with the current application data.

Application Interface

Use

The class CL_MANAGER_PPF serves as the central interface (API) for the PPF. You can use it, for example, to develop your own determination technologies.

Features

The class CL_MANAGER_PPF serves as the central interface (API) for the PPF. The following methods (services) are offered:

GET_INSTANCE (static method)

Parameter: none

Effect: Delivers the instance of class (Singleton)

ADD_TRIGGER (instance method)

Parameter:

IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
CO_TRIGGER	Changing	Type Ref To	CL_TRIGGER_PPF	Output

Effect:

Adds a new output for the context supplied

REPEAT_TRIGGER (instance method)

Parameter:

IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
IO_TRIGGER	Importing	Type Ref To	CL_TRIGGER_PPF	Output to be repeated

RO_TRIGGER	Returning	Type Ref To	CL_TRIGGER_PPF	Repeated output
Effect:				
Repeats processing of the transferred output				
DELETE_TRIGGER (instance method)				
Parameter:				
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
IO_TRIGGER	Importing	Type Ref To	CL_TRIGGER_PPF	Output
Effect:				
Deletes the transferred unprocessed output				
GET_TTYPES (instance method)				
Parameter:				
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
RO_TTYPES	Returning	Type Ref To	CL_TTYPE_COLL_PPF	Output types of context
Effect:				
Delivers the output types for the context transferred				
GET_TRIGGERS (instance method)				
Parameter:				
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context

RO_TRIGGERS	Returning	Type Ref To	CL_TRIGGER_COLL_PPF	Outputs for context
Effect:				
Delivers all possible outputs for the context				
CREATE_TRIGGER (instance method)				
Parameter:				
IP_TTYPE_NAME	Importing	Type	PPFDTT	Output type
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
RO_TRIGGER	Returning	Type Ref To	CL_TRIGGER_PPF	Output
Effect:				
Generates an output of the type transferred for this context				
DETERMINE (instance method)				
Parameter:				
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context
EP_PROTOCOL	Exporting	Type	BALLOGHNDL	Determination protocol
Effect:				
Starts the determination for the context transferred and delivers the determination protocol				

SET_APPLKEY (instance method)

Parameter:

IP_APPLKEY	Importing	Type	PPFDAPPKEY	Application key
IO_CONTEXT	Importing	Type Ref To	CL_CONTEXT_PPF	Context

Effect:

Sets the field APPLKEY in all outputs of the context, according to which searches and sorting can be performed

REFRESH (instance method)

Parameter: none

Effect:

All PPF manager data is lost. Resets the PPF manager, allocated memory is released

Extensibility Using BADIs

Use

Business Add-Ins (BADIs) for the PPF allow additional manipulations for the defined times.

Features

Currently the following BADIs exist:

BADI for printer determination (PRINTER_DETERM_PPF)

The printer is generally returned by the determination with the template for the output. The determination returns an output template with output media. A printer can be entered for the output media.

It is often the case that a specific printer should be entered because of the application data. In this way, for example, the printer can be determined depending on the sales organization. The sales organization is naturally not recognized for defining the media template. For this reason, the following BADI is called provided that the template or the determination does not return a printer.

The BADI has the name of the output type as a filter value. As an additional import parameter the implementing class has a reference to the context object and hence access to all document data. A printer can now be found and returned using the document data.

BADI after generated output (TRIGGER_EXECUTED)

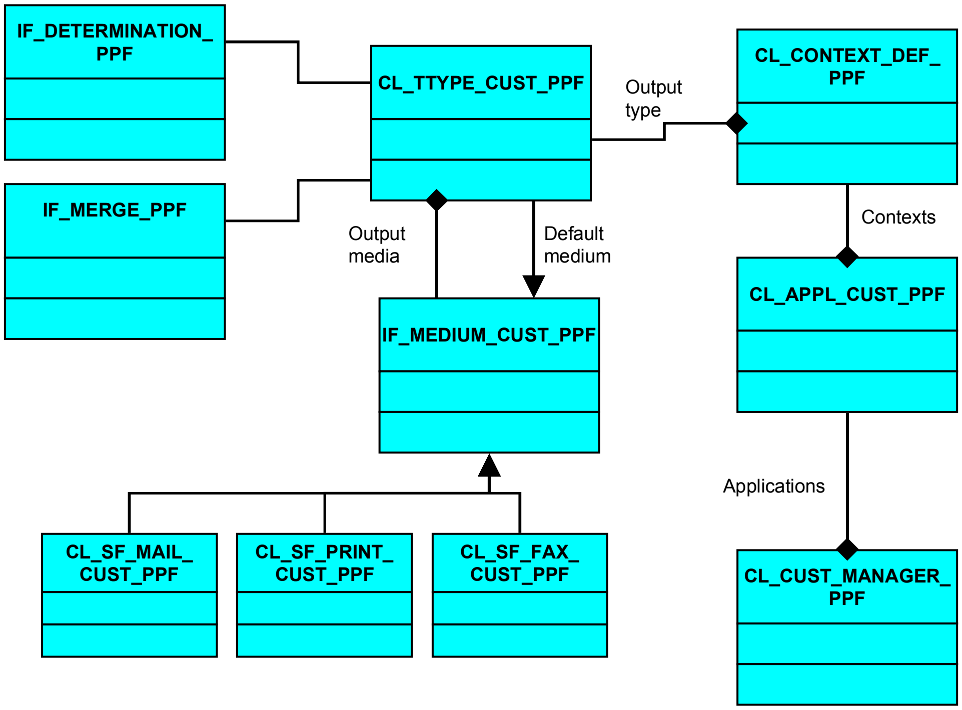
This BADI is called after a completed output, that is after processing of an output. As a result the application can execute follow-up actions. If an error takes place during processing of the output, a routine/transaction, for example, can be executed for error handling.

The BADI has the name of the application as a filter value. The output is likewise transferred with it and can deliver its processing status (successful, containing errors) or other additional information.

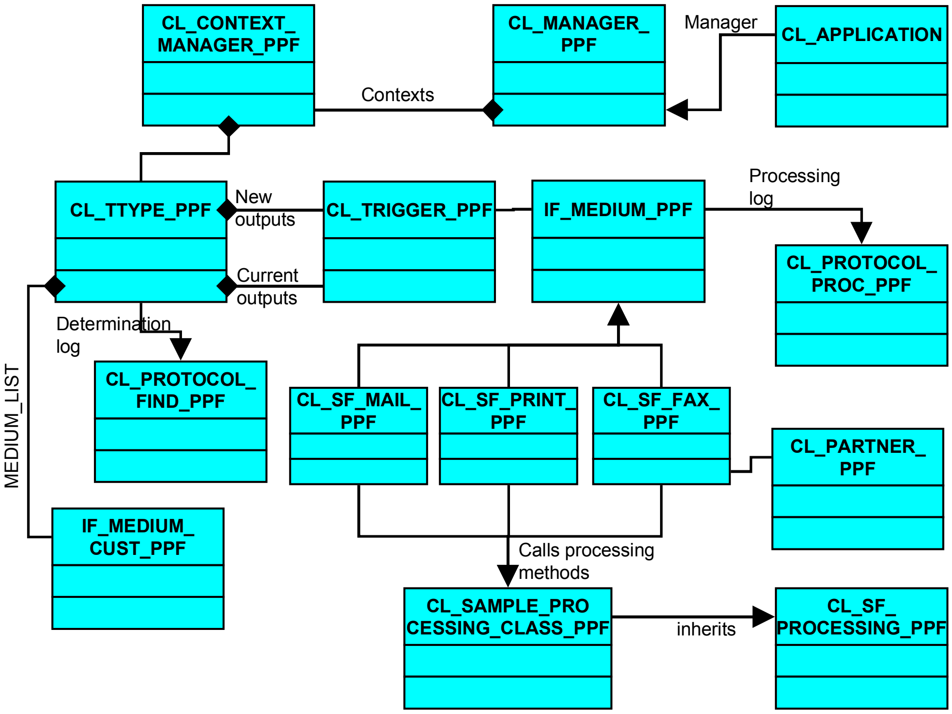
Class Diagrams

You can display details for the individual classes in the navigation frame.

Customizing Classes



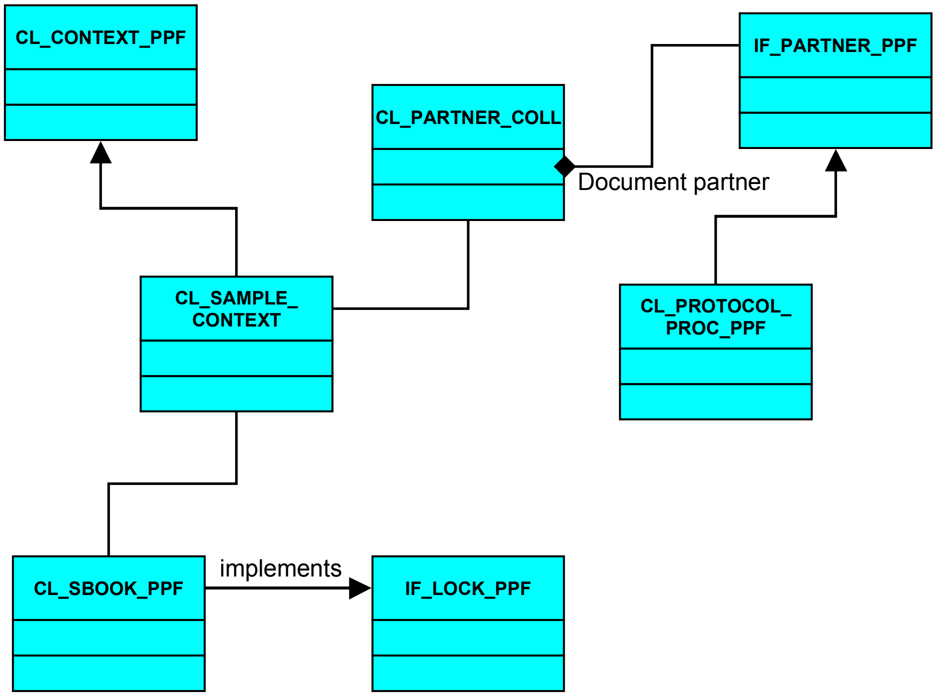
Runtime Classes



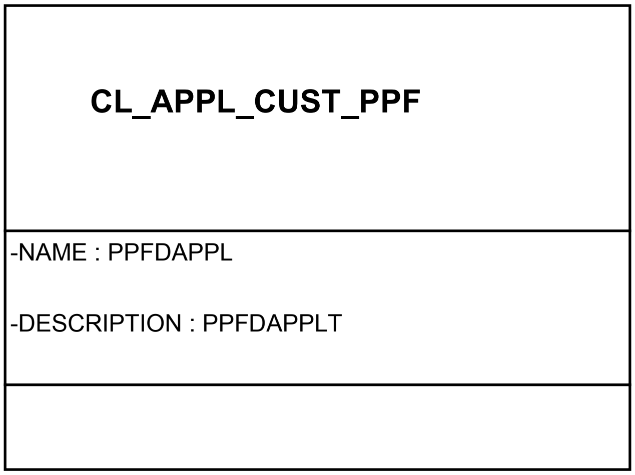
Service Classes

The superclass CL_CONTEXT_PPF encapsulates all necessary data for the PPF. An application must define a class that inherits from it and extend it by the determination relevant attributes. A reference to this class is transferred in order to start the PPF.

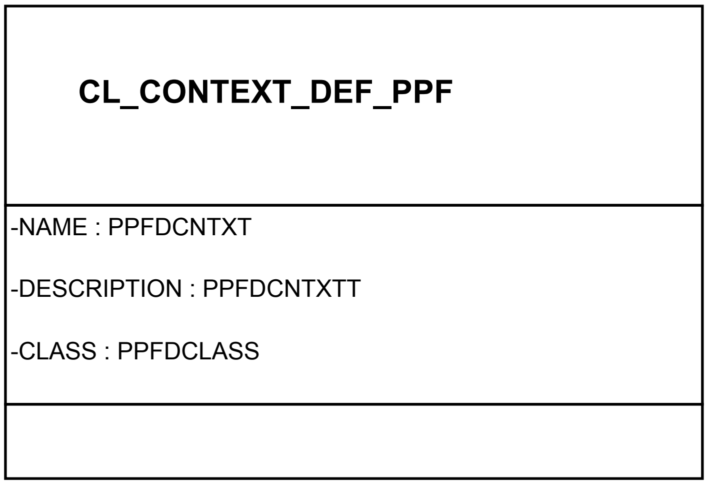
In the screen, for example, the application class is called CL_SAMPLE_CONTEXT. Further examples of application classes are CL_SBOOK_PPF and CL_SBOOK_PARTNER_PPF.


CL_CONTEXT_PPF
CL_PARTNER_COLL
IF_PARTNER_PPF
Document partner
CL_SAMPLE_ CONTEXT
CL_PROTOCOL_ PROC_PPF
CL_SBOOK_PPF
implements
IF_LOCK_PPF

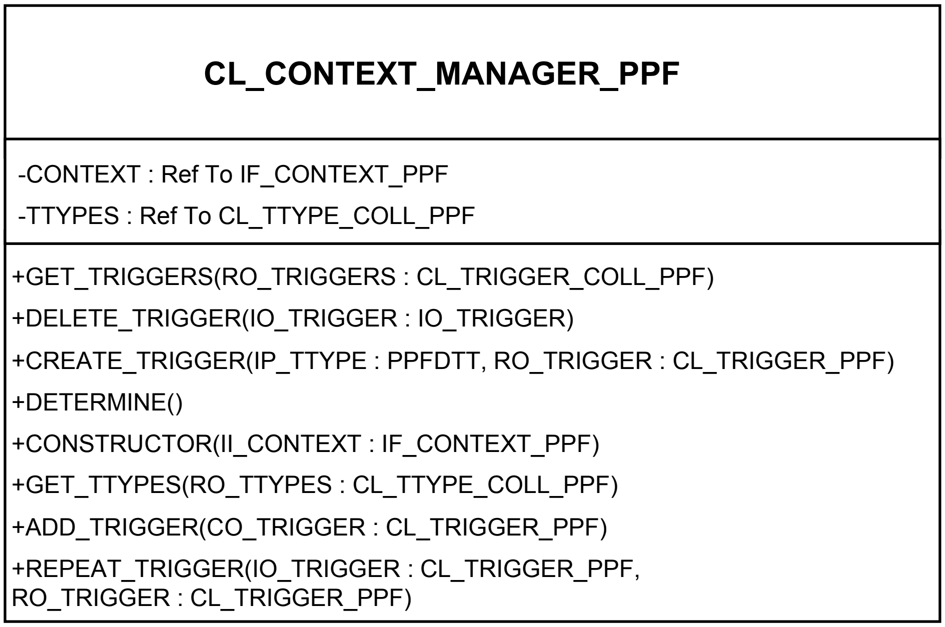
CL_APPL_CUST_PPF



CL_CONTEXT_DEF_PPF



CL_CONTEXT_MANAGER_PPF



CL_CONTEXT_MANAGER_PPF

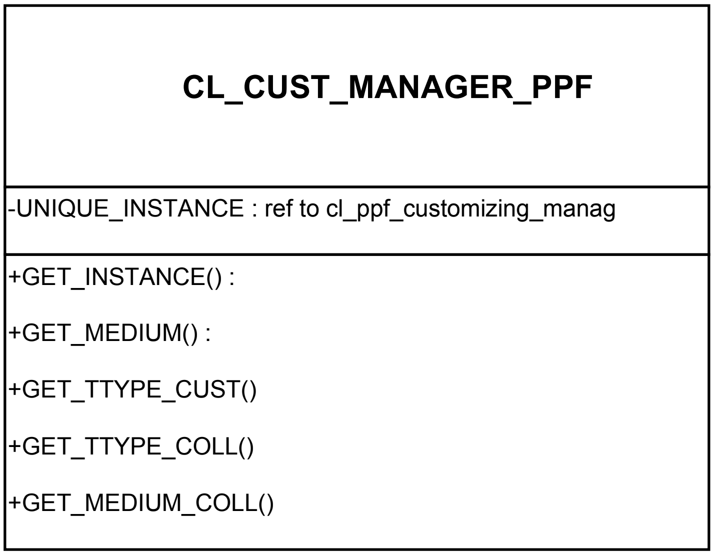
-CONTEXT : Ref To IF_CONTEXT_PPF
-TTYPES : Ref To CL_TTYPE_COLL_PPF

+GET_TRIGGERS(RO_TRIGGERS : CL_TRIGGER_COLL_PPF)
+DELETE_TRIGGER(IO_TRIGGER : IO_TRIGGER)
+CREATE_TRIGGER(IP_TTYPE : PPFDTT, RO_TRIGGER : CL_TRIGGER_PPF)
+DETERMINE()
+CONSTRUCTOR(II_CONTEXT : IF_CONTEXT_PPF)
+GET_TTYPES(RO_TTYPES : CL_TTYPE_COLL_PPF)
+ADD_TRIGGER(CO_TRIGGER : CL_TRIGGER_PPF)
+REPEAT_TRIGGER(IO_TRIGGER : CL_TRIGGER_PPF,
RO_TRIGGER : CL_TRIGGER_PPF)

CL_CONTEXT_PPF

CL_CONTEXT_PPF	
NAME	PPFDCNTXT
APPL	PPFDAPPL
APPLCTN	REF TO OBJECT
PARTNER	CL_PARTNER_COLL_PPF
GET_VALUE_OF_ATTRIBUTE	

CL_CUST_MANAGER_PPF



CL_CUST_MANAGER_PPF

-UNIQUE_INSTANCE : ref to cl_ppf_customizing_manag

+GET_INSTANCE() :

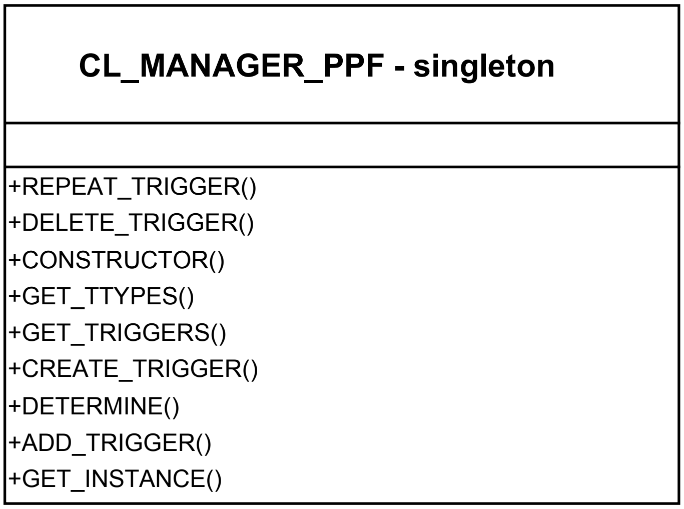
+GET_MEDIUM() :

+GET_TTYPE_CUST()

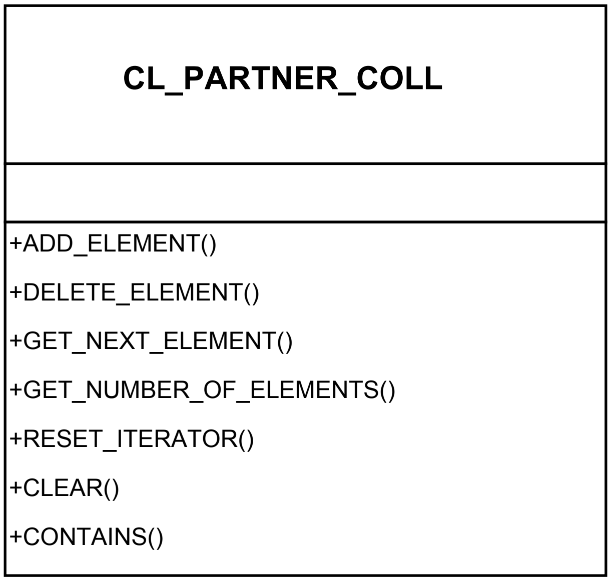
+GET_TTYPE_COLL()

+GET_MEDIUM_COLL()

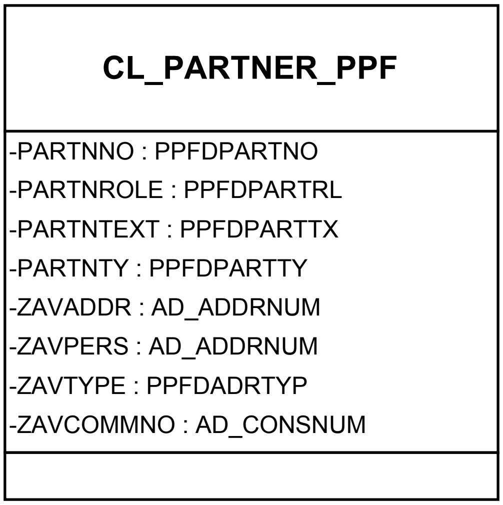
CL_MANAGER_PPF



CL_PARTNER_COLL



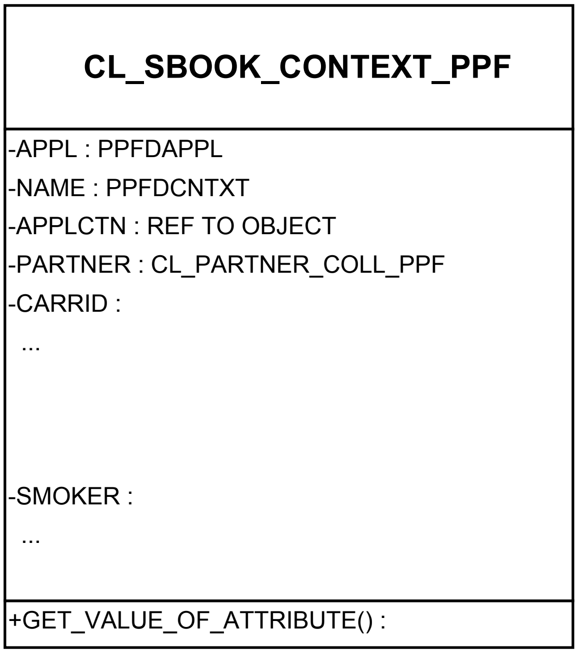
CL_PARTNER_PPF



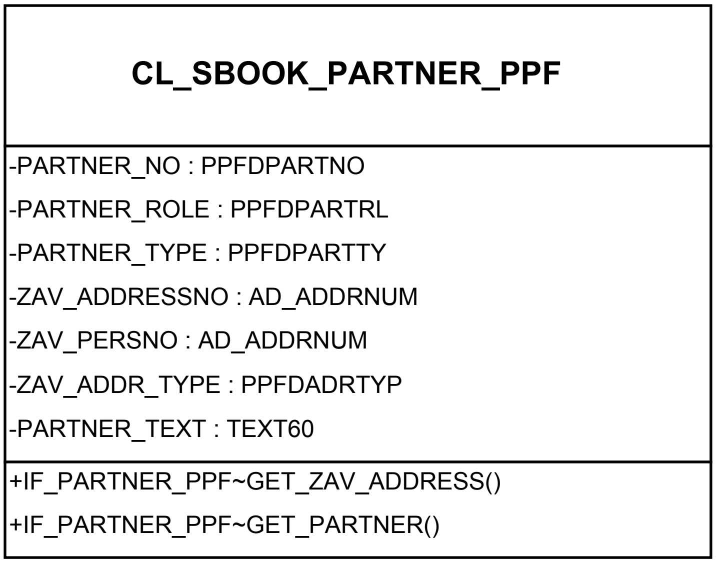
CL_PARTNER_PPF

-PARTNNO : PPFDPARTNO
-PARTNROLE : PPFDPARTRL
-PARTNTEXT : PPFDPARTTX
-PARTNTY : PPFDPARTTY
-ZAVADDR : AD_ADDRNUM
-ZAVPERS : AD_ADDRNUM
-ZAVTYPE : PPFDADRTYP
-ZAVCOMMNO : AD_CONSNUM

CL_SBOOK_CONTEXT (Example)



CL_SBOOK_PARTNER_PPF (Example)

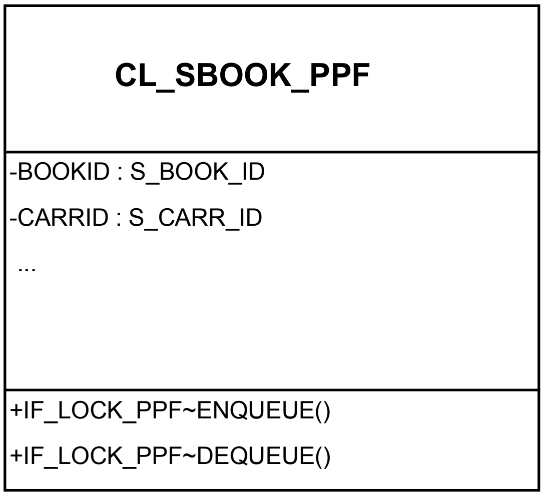


CL_SBOOK_PARTNER_PPF

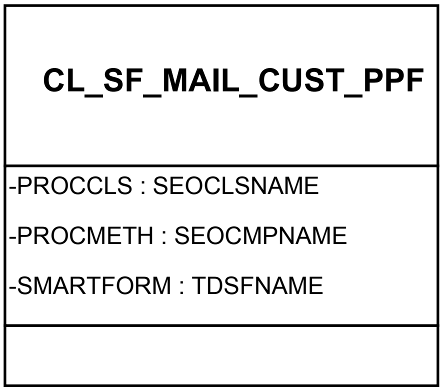
-PARTNER_NO : PPFDPARTNO
-PARTNER_ROLE : PPFDPARTRL
-PARTNER_TYPE : PPFDPARTTY
-ZAV_ADDRESSNO : AD_ADDRNUM
-ZAV_PERSNO : AD_ADDRNUM
-ZAV_ADDR_TYPE : PPFDADRTYP
-PARTNER_TEXT : TEXT60

+IF_PARTNER_PPF~GET_ZAV_ADDRESS()
+IF_PARTNER_PPF~GET_PARTNER()

CL_SBOOK_PPF (Example)



CL_SF_MAIL_CUST_PPF



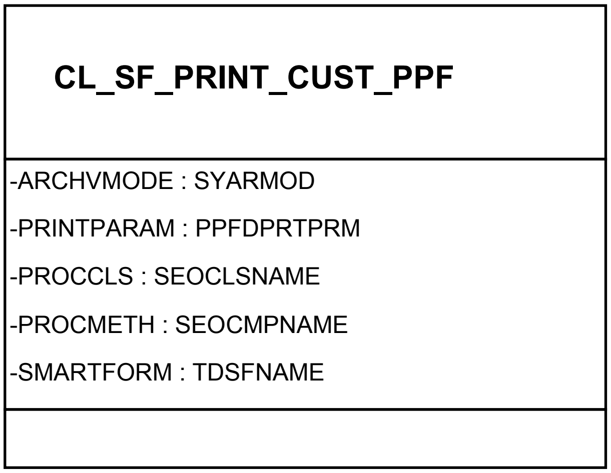
CL_SF_MAIL_CUST_PPF

-PROCCLS : SEOCLSNAME

-PROCMETH : SEOCMPNAME

-SMARTFORM : TDSFNAME

CL_SF_PRINT_CUST_PPF



CL_SF_PRINT_CUST_PPF

-ARCHVMODE : SYARMOD

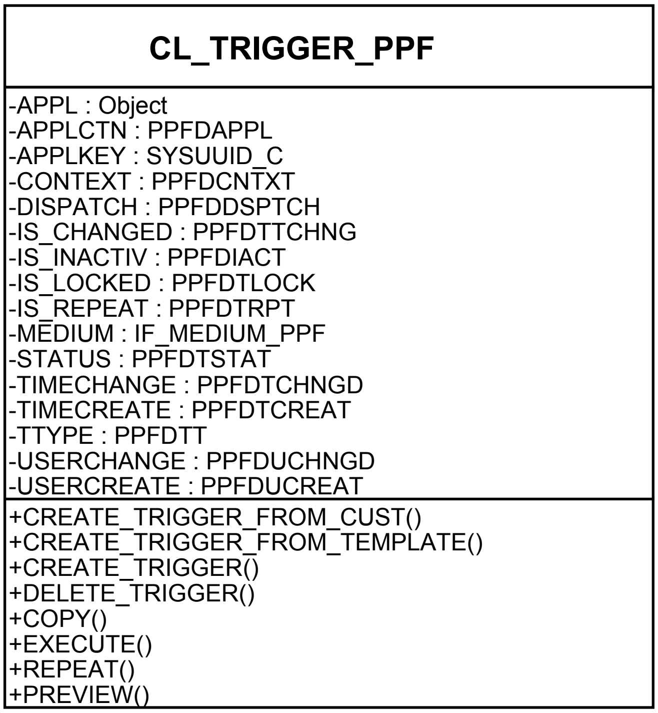
-PRINTPARAM : PPFDPRTPRM

-PROCCLS : SEOCLSNAME

-PROCMETH : SEOCMPNAME

-SMARTFORM : TDSFNAME

CL_TRIGGER_PPF

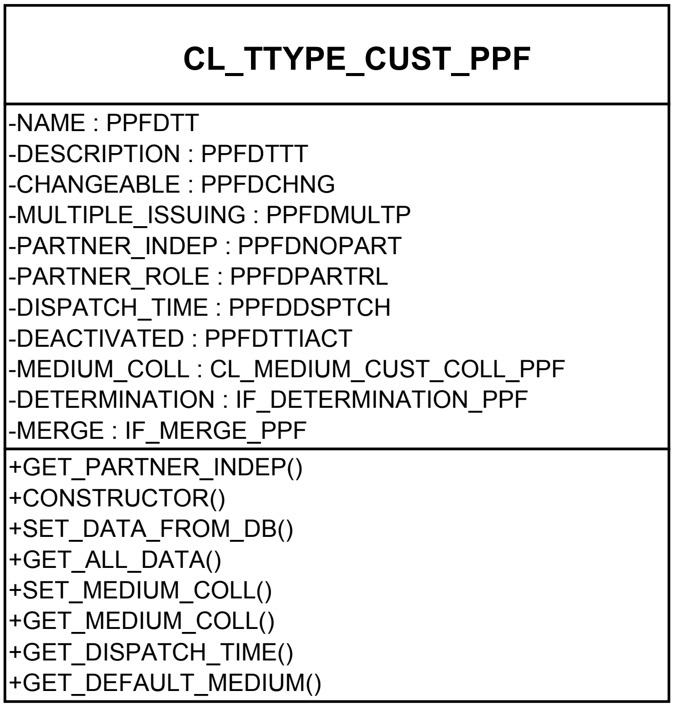


CL_TRIGGER_PPF

-APPL : Object
-APPLCTN : PPFDAPPL
-APPLKEY : SYSUUID_C
-CONTEXT : PPFDCNTXT
-DISPATCH : PPFDDSPTCH
-IS_CHANGED : PPFDTTCHNG
-IS_INACTIV : PPFDIACT
-IS_LOCKED : PPFDTLOCK
-IS_REPEAT : PPFDTRPT
-MEDIUM : IF_MEDIUM_PPF
-STATUS : PPFDTSTAT
-TIMECHANGE : PPFDTCHNGD
-TIMECREATE : PPFDTCREAT
-TTYPE : PPFDTT
-USERCHANGE : PPFDUCHNGD
-USERCREATE : PPFDUCREAT

+CREATE_TRIGGER_FROM_CUST()
+CREATE_TRIGGER_FROM_TEMPLATE()
+CREATE_TRIGGER()
+DELETE_TRIGGER()
+COPY()
+EXECUTE()
+REPEAT()
+PREVIEW()

CL_TTYPE_CUST_PPF

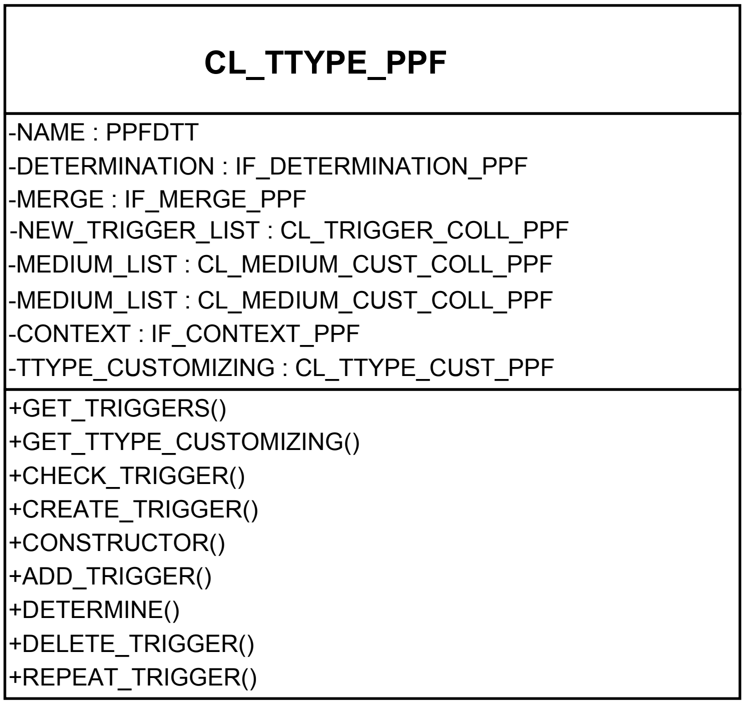


CL_TTYPE_CUST_PPF

-NAME : PPFDTT
-DESCRIPTION : PPFDTTT
-CHANGEABLE : PPFDCHNG
-MULTIPLE_ISSUING : PPFDMULTP
-PARTNER_INDEP : PPFDNOPART
-PARTNER_ROLE : PPFDPARTRL
-DISPATCH_TIME : PPFDDSPTCH
-DEACTIVATED : PPFDTTIACT
-MEDIUM_COLL : CL_MEDIUM_CUST_COLL_PPF
-DETERMINATION : IF_DETERMINATION_PPF
-MERGE : IF_MERGE_PPF

+GET_PARTNER_INDEP()
+CONSTRUCTOR()
+SET_DATA_FROM_DB()
+GET_ALL_DATA()
+SET_MEDIUM_COLL()
+GET_MEDIUM_COLL()
+GET_DISPATCH_TIME()
+GET_DEFAULT_MEDIUM()

CL_TTYPE_PPF

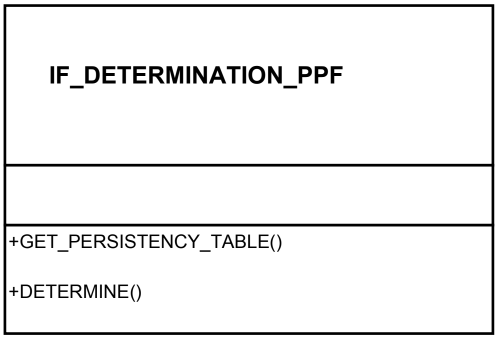


CL_TTYPE_PPF

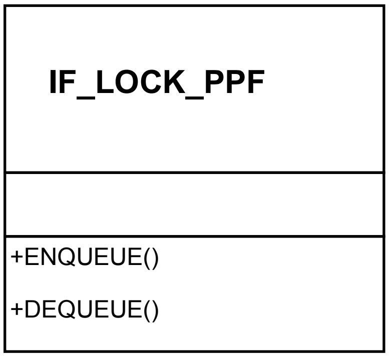
-NAME : PPFDTT
-DETERMINATION : IF_DETERMINATION_PPF
-MERGE : IF_MERGE_PPF
-NEW_TRIGGER_LIST : CL_TRIGGER_COLL_PPF
-MEDIUM_LIST : CL_MEDIUM_CUST_COLL_PPF
-MEDIUM_LIST : CL_MEDIUM_CUST_COLL_PPF
-CONTEXT : IF_CONTEXT_PPF
-TTYPE_CUSTOMIZING : CL_TTYPE_CUST_PPF

+GET_TRIGGERS()
+GET_TTYPE_CUSTOMIZING()
+CHECK_TRIGGER()
+CREATE_TRIGGER()
+CONSTRUCTOR()
+ADD_TRIGGER()
+DETERMINE()
+DELETE_TRIGGER()
+REPEAT_TRIGGER()

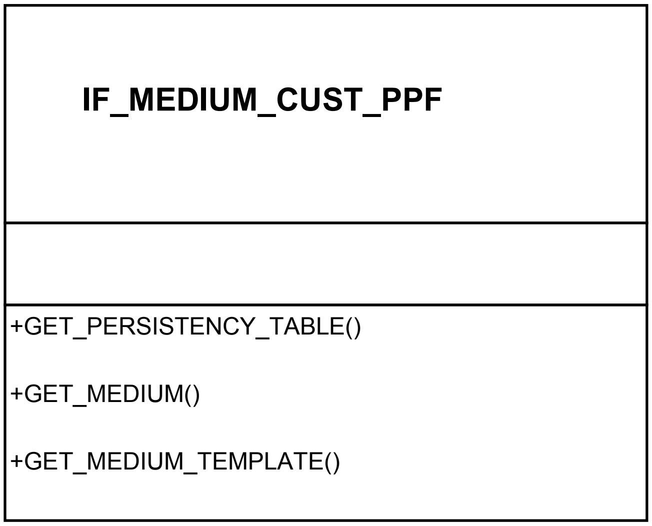
IF_DETERMINATION_PPF



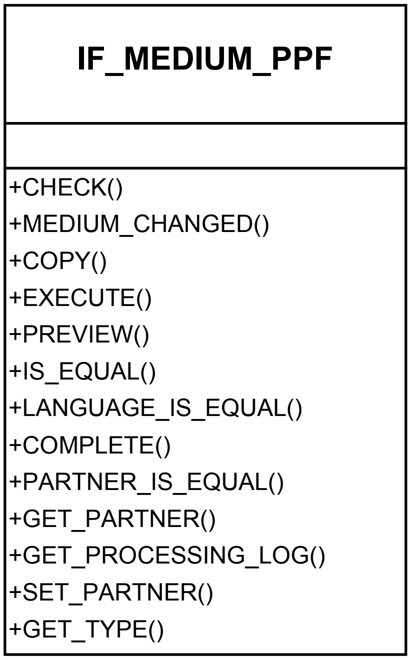
IF_LOCK_PPF



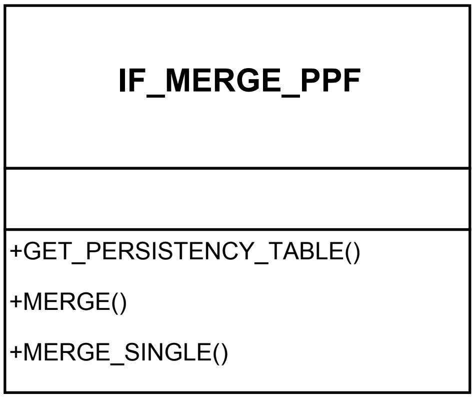
IF_MEDIUM_CUST_PPF



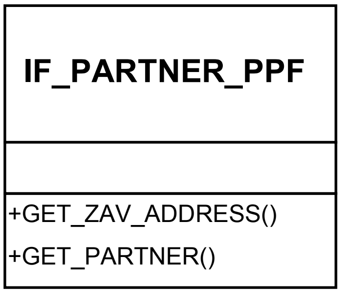
IF_MEDIUM_PPF



IF_MERGE_PPF



IF_PARTNER_PPF


IF_PARTNER_PPF
+GET_ZAV_ADDRESS()
+GET_PARTNER()