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

Release 4.6C
Copyright

© Copyright 2001 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

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

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

ORACLE® is a registered trademark of ORACLE Corporation.

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Caution" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="image2" alt="Example" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="image3" alt="Note" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="image4" alt="Recommendation" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="image5" alt="Syntax" /></td>
<td>Syntax</td>
</tr>
<tr>
<td><img src="image6" alt="Tip" /></td>
<td>Tip</td>
</tr>
</tbody>
</table>
Contents

Generic Business Tools for Application Developers (BC-SRV-GBT)................. 8
Business Communication Interface - Sending from R/3 Applications ................. 9

Business Communication Interface ........................................................................ 11
  Information Availability via Object Links ............................................................... 12
  Status Confirmation .................................................................................................... 13
    Send Process and Status Confirmation ................................................................. 14
    Active Status Confirmation by Event ................................................................. 15
    Active Status Confirmation by Mail ..................................................................... 16
  Status Display by Object Links ............................................................................. 17
Connection to the Interface ....................................................................................... 18
  Message ................................................................................................................... 19
    Create ................................................................................................................... 21
    Attach ................................................................................................................... 23
    Find ..................................................................................................................... 24
    Submit .................................................................................................................. 25
    Save ..................................................................................................................... 26
    AddCopyOfRecipient ............................................................................................ 27
    RemoveRecipient ................................................................................................. 28
    EditRecipientList ................................................................................................. 29
    SetAsynchron ....................................................................................................... 30
    AddApplObjectID ................................................................................................. 31
    SetOutboxFlag ..................................................................................................... 32
    SetLinkFolder ...................................................................................................... 33
    ClearLinkFolder ................................................................................................... 34
    AddGeneralNote .................................................................................................. 35
    DeleteGeneralNote ............................................................................................. 36
    SetProcGeneral .................................................................................................... 37
    SetProcMemoryID ............................................................................................... 38
    AddProcSetGetParam ......................................................................................... 39
    AddProcImportParam ......................................................................................... 40
    ImportRecipientList ............................................................................................ 41
    SetSensitivity ....................................................................................................... 42
    Copy .................................................................................................................... 43
    SetNextSender .................................................................................................... 44
  Recipient ................................................................................................................ 45
    CreateAddress ..................................................................................................... 47
    FindAddress ........................................................................................................ 48
    Copy .................................................................................................................... 49
    Edit ..................................................................................................................... 50
    Save .................................................................................................................... 51
    ImportSendAttributes ......................................................................................... 52
    Delete .................................................................................................................. 53
Expand ................................................................................................................................. 54
AddNote ................................................................................................................................. 55
DeleteNote ............................................................................................................................. 56
SetExpress ............................................................................................................................. 57
SetCopy ................................................................................................................................. 58
SetBlindCopy ......................................................................................................................... 59
SetNotForwarding .................................................................................................................. 60
NoPrinting ............................................................................................................................... 61
SetSendPriority ....................................................................................................................... 62
SetToDoByAll ......................................................................................................................... 63
SetToDoByOne ......................................................................................................................... 64
SetReplyRequired .................................................................................................................. 65
SetDeliver ............................................................................................................................... 66
SetNotDeliver .......................................................................................................................... 67
SetRead .................................................................................................................................. 68
SetStatusInfoByMail ............................................................................................................. 69
SetResend ............................................................................................................................... 70
Indirect Connection using Message Control ......................................................................... 71
Indirect Connection using SAPscript ...................................................................................... 72
Sending Under Another Name ............................................................................................... 74
Creating Express Messages in Applications ........................................................................... 76
Connection to the Interface .................................................................................................... 78
SO_EXPRESS_FLAG_SET ........................................................................................................ 79
Business Appointment Services (4.6B) ..................................................................................... 82
Terms ..................................................................................................................................... 83
Authorizations ....................................................................................................................... 85
Appointment Types and Event Handling .................................................................................. 86
Event Handling for the Appointment Type ‘Service Order’ ...................................................... 87
Connection to the Interface .................................................................................................... 88
APPT_CREATE_INTERNAL ................................................................................................. 89
APPT_MODIFY_INTERNAL ................................................................................................. 92
APPT_DELETE_INTERNAL ................................................................................................. 96
DAY_COVERING_GET .......................................................................................................... 98
DAY_COVERING_AND_PATTERN_GET .............................................................................. 101
SC_GET_APPT_NEIGHBOR ............................................................................................... 103
APPOINTMENT_GRP_PARTICIPAN_GET ........................................................................ 105
SCHEDULE_OWNER_ATTRIBUTES_GET ........................................................................... 106
Examples of the Use of Function Modules ............................................................................. 107
Creating a Group Appointment with a Title and a Description .............................................. 108
Creating Dominants for Several Days ................................................................................... 110
Creating an Individual Appointment with a Link to an Application Object ......................... 111
Moving an Individual Appointment ...................................................................................... 113
Moving a Group Appointment for all Participants ............................................................... 116
Deleting Appointments ......................................................................................................... 119
Scenario Meeting Arrangement ............................................................................................ 120
Use of Event Handling .......................................................................................................... 125
Appointment Type Attributes ................................................................................................. 126
<table>
<thead>
<tr>
<th>Class Diagrams</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_APPL_CUST_PPF</td>
<td>209</td>
</tr>
<tr>
<td>CL_CONTEXT_DEF_PPF</td>
<td>212</td>
</tr>
<tr>
<td>CL_CONTEXT_MANAGER_PPF</td>
<td>213</td>
</tr>
<tr>
<td>CL_CONTEXT_PPF</td>
<td>214</td>
</tr>
<tr>
<td>CL_CUST_MANAGER_PPF</td>
<td>215</td>
</tr>
<tr>
<td>CL_MANAGER_PPF</td>
<td>216</td>
</tr>
<tr>
<td>CL_PARTNER_COLL</td>
<td>217</td>
</tr>
<tr>
<td>CL_PARTNER_PPF</td>
<td>218</td>
</tr>
<tr>
<td>CL_SBOOK_CONTEXT (Example)</td>
<td>219</td>
</tr>
<tr>
<td>CL_SBOOK_PARTNER_PPF (Example)</td>
<td>220</td>
</tr>
<tr>
<td>CL_SBOOK_PPF (Example)</td>
<td>221</td>
</tr>
<tr>
<td>CL_SF_MAIL_CUST_PPF</td>
<td>222</td>
</tr>
<tr>
<td>CL_SF_PRINT_CUST_PPF</td>
<td>223</td>
</tr>
<tr>
<td>CL_TRIGGER_PPF</td>
<td>224</td>
</tr>
<tr>
<td>CL_TTYPE_CUST_PPF</td>
<td>225</td>
</tr>
<tr>
<td>CL_TTYPE_PPF</td>
<td>226</td>
</tr>
<tr>
<td>IF_DETERMINATION_PPF</td>
<td>227</td>
</tr>
<tr>
<td>IF_LOCK_PPF</td>
<td>228</td>
</tr>
<tr>
<td>IF_MEDIUM_CUST_PPF</td>
<td>229</td>
</tr>
<tr>
<td>IF_MEDIUM_PPF</td>
<td>230</td>
</tr>
<tr>
<td>IF_MERGE_PPF</td>
<td>231</td>
</tr>
<tr>
<td>IF_PARTNER_PPF</td>
<td>232</td>
</tr>
</tbody>
</table>
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 [Page 9]
- Creating Express Messages in Applications [Page 76]
- Business Appointment Services [Page 82]
- Relationship Service [Page 137]
- Generic Object Services [Page 147]
- Post Processing Framework (PPF) [Page 177]
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.

Features

The BCI provides the following functions:

- Sending of documents with attachments and notes

When documents are sent externally, the BCI transfers the send objects to SAPconnect [Ext.].
Business Communication Interface - Sending from R/3 Applications

- 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.

1. Active status confirmation by event
2. Active status confirmation by mail
3. 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 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE</td>
<td>No status confirmation at all</td>
</tr>
<tr>
<td>E</td>
<td>Message only if errors exist</td>
</tr>
<tr>
<td></td>
<td>This is the standard setting.</td>
</tr>
<tr>
<td>W</td>
<td>Message only if errors exist and for warnings.</td>
</tr>
<tr>
<td>I</td>
<td>Message for every status</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Title of the document</td>
</tr>
<tr>
<td>Recipients</td>
<td>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).</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Flag indicating that sending is to be asynchronous.</td>
</tr>
<tr>
<td>OutboxFlag</td>
<td>Flag indicating that a link to the document sent is to be stored in the outbox.</td>
</tr>
<tr>
<td>LinkFolderType,</td>
<td>ID of the folder in which (possibly in addition to the outbox) a link to the document sent is to be linked.</td>
</tr>
<tr>
<td>LinkFolderYear,</td>
<td></td>
</tr>
<tr>
<td>LinkFolderNumber</td>
<td></td>
</tr>
<tr>
<td>NextSender</td>
<td>Sender for the next send process of this document</td>
</tr>
</tbody>
</table>

Message Object Methods
Create [Page 21]
Attach [Page 23]
Find [Page 24]
Submit [Page 25]
Save [Page 26]
AddCopyOfRecipient [Page 27]
RemoveRecipient [Page 28]
Message

EditRecipientList [Page 29]
SetAsynchron [Page 30]
AddAplObjectID [Page 31]
SetOutboxFlag [Page 32]
SetLinkFolder [Page 33]
AddGeneralNote [Page 35]
DeleteGeneralNote [Page 36]
SetProcGeneral [Page 37]
SetProcMemoryID [Page 38]
AddProcSetGetParam [Page 39]
AddProcImportParam [Page 40]
ImportRecipientList [Page 41]
SetSensitivity [Page 42]
Copy [Page 43]
SetNextSender [Page 44]
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

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

### Export Parameters

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENTNAME</td>
<td>Name of the document</td>
</tr>
</tbody>
</table>
Attach

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

Structure

Import Parameters

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

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

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENTNAME</td>
<td>Name of the document</td>
</tr>
<tr>
<td>DOCUMENTTYPE</td>
<td>Class of document</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllBindingDone</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewRecipient</td>
<td>New Recipient object to be added</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient</td>
<td>The Recipient object to be removed</td>
</tr>
</tbody>
</table>

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.
SetAsynchronous

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

Structure

Import Parameters

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>Flag that controls whether sending is asynchronous (X) or synchronous (Space).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object ID</td>
<td>ID of the application object</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutboxFlag</td>
<td>Flag that controls whether, in the outbox, a link to the document sent is stored (X) or not (Space).</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FolderID</td>
<td>ID of the private folder in which the document sent is to be stored.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteText</td>
<td>Table with the text of the note</td>
</tr>
</tbody>
</table>

Export Parameters

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteType</td>
<td>Type from the ID</td>
</tr>
<tr>
<td>NoteYear</td>
<td>Year from the ID</td>
</tr>
<tr>
<td>NoteNumber</td>
<td>Number from the ID</td>
</tr>
</tbody>
</table>
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

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

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` 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**

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemoryID</td>
<td>Memory ID</td>
</tr>
</tbody>
</table>

In this case, parameters for the SAP memory have to be set using the `AddProcImportParam` 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

<table>
<thead>
<tr>
<th>Export Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the parameter</td>
</tr>
<tr>
<td>Value</td>
<td>Value that the parameter shall have when it is called.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MasterMessage</td>
<td>Handle for the Message object that serves as a template.</td>
</tr>
</tbody>
</table>

Export Parameters
None.
SetSensitivity

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

**Structure**

**Import Parameters**

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Recipient address (possibly distributed over Value, Value2, Value3, Value4, Value5)</td>
</tr>
<tr>
<td>TypeDescription</td>
<td>Description of the address type</td>
</tr>
<tr>
<td>SendExpress</td>
<td>Send attribute Express</td>
</tr>
<tr>
<td>SendCopy</td>
<td>Send attribute Send as copy</td>
</tr>
<tr>
<td>SendBlindCopy</td>
<td>Send attribute Send as blind copy</td>
</tr>
<tr>
<td>NoForwarding</td>
<td>Send attribute No forwarding</td>
</tr>
<tr>
<td>NoPrinting</td>
<td>Send attribute No printing</td>
</tr>
<tr>
<td>SendPriority</td>
<td>Send attribute Send priority</td>
</tr>
<tr>
<td>ToDoByAllRecipients</td>
<td>Send attribute ToDo by all recipients</td>
</tr>
<tr>
<td>ToDoByOne</td>
<td>Send attribute ToDo by one recipient</td>
</tr>
<tr>
<td>ReplyRequired</td>
<td>Send attribute Reply required</td>
</tr>
<tr>
<td>StatusInfoByMail</td>
<td>Send attribute Status Confirmation by Mail</td>
</tr>
<tr>
<td>NoteType, NoteYear, NoteNumber</td>
<td>ID of the private note that is to be sent to a recipient</td>
</tr>
</tbody>
</table>
Recipients

**Message Object Methods**

CreateAddress [Page 47]
FindAddress [Page 48]
Copy [Page 49]
Edit [Page 50]
Save [Page 51]
ImportSendAttributes [Page 52]
Delete [Page 53]
Expand [Page 54]
AddNote [Page 55]
DeleteNote [Page 56]
SetExpress [Page 57]
SetCopy [Page 58]
SetBlindCopy [Page 59]
SetNotForwarding [Page 60]
NoPrinting [Page 61]
SetSendPriority [Page 62]
SetToDoByAll [Page 63]
SetToDoByOne [Page 64]
SetReplyRequired [Page 65]
SetDeliver [Page 66]
SetNotDeliver [Page 67]
SetRead [Page 68]
SetStatusInfoByMail [Page 69]
SetResend [Page 70]
CreateAddress

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

Structure
Import Parameters

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

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

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

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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>Handle for the Recipient object that serves as a template.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Export Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResultTable</td>
<td>Table with the handles for the ‘flat’ Recipient objects</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteText</td>
<td>Table with the text of the note</td>
</tr>
<tr>
<td></td>
<td>This parameter is mandatory.</td>
</tr>
</tbody>
</table>

**Export Parameters**

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoteType</td>
<td>Part of the note ID that specifies the type of the note created</td>
</tr>
<tr>
<td>NoteYear</td>
<td>Part of the note ID that specifies the year in which the note was created.</td>
</tr>
<tr>
<td>NoteNumber</td>
<td>Part of the note ID that specifies the number of the note created</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendExpress</td>
<td>Value for switching on or off the send attribute <em>Express</em></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendCopy</td>
<td>Value for switching on or off the send attribute <em>Send as copy</em></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlindCopy</td>
<td>Value for switching on or off the send attribute <em>Send as blind copy</em></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoForwarding</td>
<td>Value for switching on or off the send attribute <em>No forwarding</em></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoPrinting</td>
<td>Value for switching on or off the send attribute <em>No printing</em></td>
</tr>
</tbody>
</table>

Export Parameters
None.
**SetSendPriority**

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

**Structure**

**Import Parameters**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendPriority</td>
<td>Value of the send priority</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToDoByAll</td>
<td>Value for switching on or off the send attribute <em>ToDo by all recipients</em></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToDoByOne</td>
<td>Value of the send attribute <em>ToDo by one recipient of the group</em></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplyRequired</td>
<td>Value for switching on or off the send attribute Reply required</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliver</td>
<td>Value of the send status <em>Received</em></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotDeliver</td>
<td>Value of the send status <em>Not received</em></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Value of the send status <em>Read</em></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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**

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resend</td>
<td>Value for switching on (X) or off (Space) the send attribute <em>Resend</em></td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Import Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI_OBJTYPE</td>
<td>Type of application BOR object</td>
</tr>
<tr>
<td>MSG_OBJKY</td>
<td>NAST object key</td>
</tr>
</tbody>
</table>

  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 SAPscript, 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 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

<table>
<thead>
<tr>
<th>Import Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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.
SO_EXPRESS_FLAG_SET

| 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

<table>
<thead>
<tr>
<th>Export Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sent_to_all</td>
<td>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</td>
</tr>
</tbody>
</table>

Internal Table

<table>
<thead>
<tr>
<th>Internal Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec_tab</td>
<td>Internal table for transfer of a list of recipients.</td>
</tr>
</tbody>
</table>

Fields of the internal table rec_tab:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Data element</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>recnam</td>
<td>Recipient name</td>
<td>SO_REC_EXP</td>
<td>CHAR</td>
<td>130</td>
</tr>
</tbody>
</table>
### SO_EXPRESS_FLAG_SET

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| **recec** | 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. | |
| **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. | |

### Exceptions

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_RECEIVER_EXIST</td>
<td>None of the specified recipients exist.</td>
</tr>
<tr>
<td>OFFICE_NAME_ERROR</td>
<td>No Business Workplace is assigned to the caller of the function module.</td>
</tr>
</tbody>
</table>
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
Terms

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_FROM, mandatory</td>
<td>Date on which appointment starts.</td>
</tr>
<tr>
<td>DATE_TO, mandatory</td>
<td>Date on which appointment ends.</td>
</tr>
<tr>
<td>TIME_FROM, mandatory</td>
<td>Time at which appointment starts on the day DATE_FROM.</td>
</tr>
<tr>
<td>TIME_TO, mandatory</td>
<td>Time at which appointment ends on the day DATE_TO.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Appointment type</td>
</tr>
<tr>
<td>DESCRIPTION, mandatory if set in Customizing of appointment type attributes [Page 126].</td>
<td>Title of appointment</td>
</tr>
<tr>
<td>ROOM</td>
<td>Room in which the appointment takes place.</td>
</tr>
<tr>
<td>VISIBILITY</td>
<td>Classification: 1 = public, 2 = private, 3 = secret (not for group appointments)</td>
</tr>
<tr>
<td>BODY_ACCESS_ID</td>
<td>Access authorization for description of group appointment.</td>
</tr>
<tr>
<td></td>
<td>1 = changeable by all participants</td>
</tr>
<tr>
<td></td>
<td>2 = only changeable by organizer.</td>
</tr>
<tr>
<td>APPOINTMENT_IS_PLANNED</td>
<td>Not supported</td>
</tr>
<tr>
<td>ACTIVATE_SUBSTITUTE</td>
<td>Activation of substitution in HR organization model for appointment time period.</td>
</tr>
<tr>
<td>SUPPRESS_MAIL</td>
<td>No sending of notification mails for group appointments</td>
</tr>
</tbody>
</table>

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.
The calendar authorization checks are suppressed.

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.

Not supported

Flag indicating whether it should be a dominant. If so, the fields TIME_FROM and TIME_TO are not included.

Internal

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.

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

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.
**APPT_CREATE_INTERNAL**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPOINTMENT_BODY</td>
<td>Import parameter appointment description in the form of any long ASCII text.</td>
</tr>
<tr>
<td>APPOINTMENT_DATES</td>
<td>Internal</td>
</tr>
<tr>
<td>APPOINTMENT_CONFLICTDATES</td>
<td>Internal</td>
</tr>
</tbody>
</table>

**Exceptions**

None.

**Example call**

```plaintext
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

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

The calendar authorization checks are suppressed.

Application-specific (technical) information, which is not processed within the calendar but which is made available via the exit functions of the application.

Not supported

Flag indicating whether it should be a dominant. If so, the fields TIME_FROM and TIME_TO are not included.

Internal

Internal

Internal

Internal

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

Import parameter
New participant list

Internal

Internal

Internal

Internal
Import parameter
New appointment description in the form of any long ASCII text.

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.

Export parameter
Internal

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| OWNER, mandatory       | Owner of the appointment
                        | Owner of the individual appointment to be changed or organizer of the group |
| 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                                           |
|                        | NOCALENDAR_AUTHORITY_CHECK.                                                 |
| NO_CALENDAR_AUTHORITY_|
| CHECK                  | The calendar authorization checks are suppressed.                           |

Export Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_MESSAGE</td>
<td>Error structure. An error output can be displayed using the command</td>
</tr>
<tr>
<td></td>
<td>message.</td>
</tr>
</tbody>
</table>

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

```plaintext
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_FROM, mandatory</td>
<td>Date from which the appointments are to be returned.</td>
</tr>
<tr>
<td>DATE_TO, mandatory</td>
<td>Date up to which the appointments are to be returned.</td>
</tr>
<tr>
<td>NOCALENDAR AUTHORITY_CHECK</td>
<td>The calendar authorization checks are suppressed.</td>
</tr>
<tr>
<td></td>
<td>If this parameter is not set, information, such as the title of the</td>
</tr>
<tr>
<td></td>
<td>appointment, is hidden, if necessary, according to the authorizations that</td>
</tr>
<tr>
<td></td>
<td>the inquirer has for the calendar of the other user.</td>
</tr>
</tbody>
</table>

Export Parameters

None.

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPOINTMENTS</td>
<td>Export parameter&lt;br&gt;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.</td>
</tr>
<tr>
<td>DOMINANTS</td>
<td>Export parameter&lt;br&gt;Return of the dominants found.</td>
</tr>
<tr>
<td>BODY_LIST</td>
<td>Export parameter&lt;br&gt;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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_HAS_INVALID_FORMAT</td>
<td>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.</td>
</tr>
<tr>
<td>DATE_INCONSISTENCY</td>
<td>Error: DATE_TO &lt; DATE_FROM</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER, mandatory</td>
<td>Owner of the individual appointment to be changed or organizer of the group appointment to be changed.</td>
</tr>
<tr>
<td>ID, mandatory</td>
<td>GUID of an individual or group appointment created with APPT_CREATE_INTERNAL.</td>
</tr>
</tbody>
</table>

Export Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY_COVERING</td>
<td>Return of the individual appointment found.</td>
</tr>
<tr>
<td>DAY_COVERING_PATTERN</td>
<td>Return of the group appointment found.</td>
</tr>
</tbody>
</table>

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY_COVERING_BODY</td>
<td>Appointment description in the form of any long ASCII text.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY_COVERING_NOT_FOUND</td>
<td>No individual appointment found under the ID.</td>
</tr>
<tr>
<td>NO_AUTHORITY</td>
<td>Inquirer does not have authorization to read the attributes of the appointment of the other user.</td>
</tr>
<tr>
<td>OWNER_NOT_FOUND</td>
<td>User OWNER not found</td>
</tr>
<tr>
<td>DAY_COVERING_PATTERN_NOT_FOUND</td>
<td>No group appointment found under the ID.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPL_NEIGHBOR</td>
<td>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.</td>
</tr>
</tbody>
</table>

Export Parameters

None.

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPT_ID</td>
<td>Export parameter Return of the appointment ID found.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNAL_ERROR</td>
<td>Internal error</td>
</tr>
<tr>
<td>NO_LOGSYS</td>
<td>Own logical system is not defined</td>
</tr>
<tr>
<td>NOT_FOUND</td>
<td>Object relationship not found.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown error</td>
</tr>
</tbody>
</table>
Example call

```plaintext
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.
```
APPONMENT_GRP_PARTICIPAN_GET

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

Structure
Import Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENAPPT_ID, mandatory</td>
<td>GUID of a group appointment created with APPT_CREATE_INTERNAL.</td>
</tr>
</tbody>
</table>

Export Parameters
None.

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTICIPANT</td>
<td>Export parameter Appointment participant list</td>
</tr>
</tbody>
</table>

Exceptions
None.

Example call

```java
    call function 'APPONMENT_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER, mandatory</td>
<td>Owner of an appointment calendar.</td>
</tr>
</tbody>
</table>

Export Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER_ATTRIBUTES</td>
<td>Attributes of the calendar ([user defaults [Ext.]], for example display time period [Ext.] or factory calendar set [Ext.].)</td>
</tr>
</tbody>
</table>

Table Parameters
None.

Exceptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER_NOT_FOUND</td>
<td>User OWNER not found</td>
</tr>
<tr>
<td>ATTRIBUTES_NOT_FOUND</td>
<td>Obsolete</td>
</tr>
</tbody>
</table>

Example call

```call function 'SCHEDULE_OWNER_ATTRIBUTES_GET'
exporting
    owner            = 'S_MCQUEEN'
importing
    owner_attributes = 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.
Creating an Individual Appointment with a Link to an Application Object
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_PARTICIPANT_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
Moving a Group Appointment for all Participants
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 corresponding participant appointments are deleted as well.
* 'M_DOUGLAS' is the owner resp. organizer 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 type 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
Scenario Meeting Arrangement

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 = organizer.
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 = organizer
    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 = appointment_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPOINTMENT_IN</td>
<td>Processed appointment with its attributes.</td>
</tr>
<tr>
<td>EVENT</td>
<td>Situation-dependent parameter, for example, 'CREATE' when an appointment is being created.</td>
</tr>
</tbody>
</table>

Export Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPOINTMENT_OUT</td>
<td>Returned appointment, possibly with changed attributes.</td>
</tr>
<tr>
<td>ERROR_MESSAGE</td>
<td>This parameter is used for transferring errors.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Table Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>Table for additional data exchange. The entries in this table are interpreted differently, depending on the event to be processed.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Control Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_EVENT_COPY</td>
<td>Copy appointment</td>
</tr>
<tr>
<td>C_EVENT_CREATE(*)</td>
<td>Create appointment</td>
</tr>
<tr>
<td>C_EVENT_CUT</td>
<td>Cut appointment</td>
</tr>
<tr>
<td>C_EVENT_DELETE</td>
<td>Delete appointment</td>
</tr>
<tr>
<td>C EVENT_TO MODIFY</td>
<td>Change appointment</td>
</tr>
<tr>
<td>C EVENT_TO MOVE</td>
<td>Move appointment</td>
</tr>
<tr>
<td>C EVENT_TO DISPLAY</td>
<td>Display appointment</td>
</tr>
<tr>
<td>C EVENT_MARK</td>
<td>Select appointment</td>
</tr>
<tr>
<td>'CHECK'</td>
<td>Check appointment when creating it</td>
</tr>
</tbody>
</table>

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

The control parameter ERROR_MESSAGE-MSG_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:

<table>
<thead>
<tr>
<th>Control Event</th>
<th>Action</th>
</tr>
</thead>
</table>

April 2001 135
Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_EVENT_CREATE(*)</td>
<td>Select an appointment type for the appointment; Create additional information.</td>
</tr>
<tr>
<td>C_EVENT_MODIFY</td>
<td>Change additional information.</td>
</tr>
<tr>
<td>C_EVENT_DESCRIPT</td>
<td>Obtain additional information.</td>
</tr>
<tr>
<td>C_EVENT_DISPLAY</td>
<td>Create or update calendar. Display appointment. Display additional information.</td>
</tr>
<tr>
<td>C_EVENT_READ</td>
<td>Obtain additional information in display mode.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Event</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>3-line info text in additional info box</td>
</tr>
<tr>
<td>Row 2</td>
<td></td>
</tr>
<tr>
<td>Row 3</td>
<td></td>
</tr>
<tr>
<td>Row 4</td>
<td></td>
</tr>
</tbody>
</table>
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 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.
- You use the API from object relationships 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:

<table>
<thead>
<tr>
<th>Model parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role type</td>
<td>Predecessor, successor</td>
</tr>
<tr>
<td>Object type for role type</td>
<td>BUS2030 (request for quotation) to predecessor, BUS2031 (quotation) to successor</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Model parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role type</td>
<td>Predecessor, successor</td>
</tr>
<tr>
<td>Object type for role type</td>
<td>LIST for predecessor and successor</td>
</tr>
</tbody>
</table>
| 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:

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

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 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

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

Simple list display

<table>
<thead>
<tr>
<th>Function module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SREL_DISPLAY_LIST_OF_NEIGHBORS</td>
<td>Displays the next neighbor of an object (distance is zero)</td>
</tr>
<tr>
<td>SREL_DISPLAY_LIST_OF_RELATIONS</td>
<td>Displays the relationships of an object. The maximum interval can be entered. You can go to the display of an object from the list.</td>
</tr>
</tbody>
</table>
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.
Create and Display Relationship (Example)

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 \( \text{\frown}\) in the Toolbox [Page 147].
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.
Adding a PC Document (to the Storage System)

Object type

Initial screen from the object services

OAC3

Content Repository

Document type

OAC2

Document class (DOC, TIF, ...)

April 2001
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
Entering a Bar Code
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 SAPScript 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 SAPScript 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_WL_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.

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
c1_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 \texttt{cl\_gos\_manager} you do not enter an object ID. Thus the object is not published:

   \begin{verbatim}
   data: go_myobject type ref to cl_gos_manager.
   ...
   create object go_myobject.
   \end{verbatim}

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

   \begin{verbatim}
   data: ..., lo_container type ref to cl_gui_custom_container,
   ls_object type borident.
   ...
   call method go_myobject\textasciitilde display_toolbox
     exporting is_object = ls_object
     io_container = lo_container.
   \end{verbatim}

The parameter \texttt{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.

```
CL_SBOOK_PPF
+BOOKID : S_BOOK_ID
+IF_LOCK_PPF~ENQUEUE()
+IF_LOCK_PPF~DEQUEUE()
```

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,
```

```
+ADD_ELEMENT()

+GET_ZAV_ADDRESS()
+GET_PARTNER()

IF_PARTNER_PPF

CL_PARTNER_COLL

Document partner

IF_PARTNER_PPF

CL_SBOOK_PARTNER_PPF

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

-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()
```
* 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 = '000015762'
    ip_zav_persno    = '000015763'
    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.

   \[
   \text{CL_SF_PROCESSING_PPF} \\
   \text{+EXEC_SMART_FORM()}
   \]

   \text{inherits from}

   \[
   \text{CL_PROCESSING_DEMO_PPF} \\
   \text{+EXEC_SMART_FORM()} \\
   \text{+PROCESS_SMART_FORM()}
   \]

   execute
   plus specific parameters

   \[
   \text{Function module of Smart Form:} \\
   \text{in: general parameter} \\
   \text{in: specific parameter} \\
   \text{out: general parameter}
   \]

   general out parameter of function module

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.
Creating Processing Class

    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  =
Creating Processing Class

*-----------------------------------------------------------------------

*--------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
    outputCanceled = 1
    parameter_error = 2
    OTHERS = 3

.
Creating Processing Class

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
Programming Your Own Determination

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Importing/Returning</th>
<th>Type</th>
<th>Reference/Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_CONTEXT</td>
<td>Importing</td>
<td>Type Ref</td>
<td>CL_CONTEXT_PPF</td>
<td>Context class</td>
</tr>
<tr>
<td>IP_TTYPE</td>
<td>Importing</td>
<td>Type</td>
<td>PPFDTT</td>
<td>Output type</td>
</tr>
<tr>
<td>IP_DETLOG</td>
<td>Importing</td>
<td>Type</td>
<td>BALLOGHNDL</td>
<td>Log handle</td>
</tr>
<tr>
<td>RO_TRIGGER_TEMPL_COLL</td>
<td>Returning</td>
<td>Type Ref</td>
<td>CL_TRIGGER_TEMPL_COLL_PPF</td>
<td>Collection of output templates</td>
</tr>
</tbody>
</table>
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_PERSISTENT
- CY_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
Programming Your Own Merging

RO_TRIGGER_COL

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
Performing Customizing

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 (RSPFPFPROCESS 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 balloghdl.

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
Example: Programming Calling of PPF

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
  - **IO_TRIGGER** Importing 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
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
## Application Interface

### RO_TRIGGER

<table>
<thead>
<tr>
<th>Returning Type</th>
<th>RO_TRIGGER</th>
<th>CL_TRIGGER_COLL_PP</th>
<th>Outputs for context</th>
</tr>
</thead>
</table>

**Effect:**
Delivers all possible outputs for the context

### CREATE_TRIGGER

(Instance method)

**Parameter:**

<table>
<thead>
<tr>
<th>Importing Type</th>
<th>IP_TTYPE_NAME</th>
<th>PPFDTT</th>
<th>Output type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Importing Type Ref To</th>
<th>IO_CONTEXT</th>
<th>CL_CONTEXT_PPF</th>
<th>Context</th>
</tr>
</thead>
</table>

**Effect:**
Generates an output of the type transferred for this context

### DETERMINE

(Instance method)

**Parameter:**

<table>
<thead>
<tr>
<th>Importing Type Ref To</th>
<th>IO_CONTEXT</th>
<th>CL_CONTEXT_PPF</th>
<th>Context</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Exporting Type</th>
<th>EP протокол</th>
<th>BALLOGHNDL</th>
<th>Determination protocol</th>
</tr>
</thead>
</table>

**Effect:**
Starts the determination for the context transferred and delivers the determination protocol
GET APPLKEY
Y (instance
method)

Parameter:

IP_APPLKEY

Importing
Type
PPFDAPPKEY
Application key

IOCONTEXT

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
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_APPL_CUST_PPF
|------------------
| -NAME : PPFDAPPL
| -DESCRIPTION : PPFDAPPLT

### CL_APPL_CUST_PPF

- **NAME:** PPFDAPPL
- **DESCRIPTION:** PPFDAPPLT
## CL_CONTEXT_DEF_PPF

<table>
<thead>
<tr>
<th>CL_CONTEXT_DEF_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-NAME : PPFDCTXT</td>
</tr>
<tr>
<td>-DESCRIPTION : PPFDCTXTT</td>
</tr>
<tr>
<td>-CLASS : PPFDCLASS</td>
</tr>
</tbody>
</table>
## CL_CONTEXT_MANAGER_PPF

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>-CONTEXT : Ref To IF_CONTEXT_PPF</td>
</tr>
<tr>
<td>-TTYPES : Ref To CL_TTYPE_COLL_PPF</td>
</tr>
<tr>
<td>+GET_TRIGGERS(RO_TRIGGERS : CL_TRIGGER_COLL_PPF)</td>
</tr>
<tr>
<td>+DELETE_TRIGGER(IO_TRIGGER : IO_TRIGGER)</td>
</tr>
<tr>
<td>+CREATE_TRIGGER(IP_TTYPE : PPFDTT, RO_TRIGGER : CL_TRIGGER_PPF)</td>
</tr>
<tr>
<td>+DETERMINE()</td>
</tr>
<tr>
<td>+CONSTRUCTOR(II_CONTEXT : IF_CONTEXT_PPF)</td>
</tr>
<tr>
<td>+GET_TTYPES(RO_TTYPES : CL_TTYPE_COLL_PPF)</td>
</tr>
<tr>
<td>+ADD_TRIGGER(CO_TRIGGER : CL_TRIGGER_PPF)</td>
</tr>
<tr>
<td>+REPEAT_TRIGGER(IO_TRIGGER : CL_TRIGGER_PPF, RO_TRIGGER : CL_TRIGGER_PPF)</td>
</tr>
</tbody>
</table>
### CL_CONTEXT_PPF

<table>
<thead>
<tr>
<th>CL_CONTEXT_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>APPL</td>
</tr>
<tr>
<td>APPLCTN</td>
</tr>
<tr>
<td>PARTNER</td>
</tr>
</tbody>
</table>

GET_VALUE_OF_ATTRIBUTE
## CL_CUST_MANAGER_PPF

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-UNIQUE_INSTANCE</td>
</tr>
<tr>
<td>+GET_INSTANCE()</td>
</tr>
<tr>
<td>+GET_MEDIUM()</td>
</tr>
<tr>
<td>+GET_TTYPE_CUST()</td>
</tr>
<tr>
<td>+GET_TTYPE_COLL()</td>
</tr>
<tr>
<td>+GET_MEDIUM_COLL()</td>
</tr>
</tbody>
</table>

-UNIQUE_INSTANCE: ref to cl_ppf_customizing_manag
**CL_MANAGER_PPF**

<table>
<thead>
<tr>
<th>CL_MANAGER_PPF - singleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>+REPEAT_TRIGGER()</td>
</tr>
<tr>
<td>+DELETE_TRIGGER()</td>
</tr>
<tr>
<td>+CONSTRUCTOR()</td>
</tr>
<tr>
<td>+GET_TTYPES()</td>
</tr>
<tr>
<td>+GET_TRIGGERS()</td>
</tr>
<tr>
<td>+CREATE_TRIGGER()</td>
</tr>
<tr>
<td>+DETERMINE()</td>
</tr>
<tr>
<td>+ADD_TRIGGER()</td>
</tr>
<tr>
<td>+GET_INSTANCE()</td>
</tr>
</tbody>
</table>
### CL_PARTNER_COLL

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ADD_ELEMENT()</td>
</tr>
<tr>
<td>+DELETE_ELEMENT()</td>
</tr>
<tr>
<td>+GET_NEXT_ELEMENT()</td>
</tr>
<tr>
<td>+GET_NUMBER_OF_ELEMENTS()</td>
</tr>
<tr>
<td>+RESET_ITERATOR()</td>
</tr>
<tr>
<td>+CLEAR()</td>
</tr>
<tr>
<td>+CONTAINS()</td>
</tr>
</tbody>
</table>
CL_PARTNER_PPF

<table>
<thead>
<tr>
<th>CL_PARTNER_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-PARTNNO : PPFDPARTNO</td>
</tr>
<tr>
<td>-PARTNROLE : PPFDPARTRL</td>
</tr>
<tr>
<td>-PARTNTEXT : PPFDPARTTX</td>
</tr>
<tr>
<td>-PARTNTY : PPFDPARTTY</td>
</tr>
<tr>
<td>-ZAVADDR : AD_ADDRNUM</td>
</tr>
<tr>
<td>-ZAVPERS : AD_ADDRNUM</td>
</tr>
<tr>
<td>-ZAVTYPE : PPFDADRTYP</td>
</tr>
<tr>
<td>-ZAVCOMMNO : AD_CONSNUM</td>
</tr>
</tbody>
</table>
### CL_SBOOK_CONTEXT (Example)

<table>
<thead>
<tr>
<th><strong>CL_SBOOK_CONTEXT_PPF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>APPL</strong> : PPFDAPPL</td>
</tr>
<tr>
<td>- <strong>NAME</strong> : PPFDCTXT</td>
</tr>
<tr>
<td>- <strong>APPLCTN</strong> : REF TO OBJECT</td>
</tr>
<tr>
<td>- <strong>PARTNER</strong> : CL_PARTNER_COLL_PPF</td>
</tr>
<tr>
<td>- <strong>CARRID</strong> : ...</td>
</tr>
</tbody>
</table>

- **SMOKER** : ...

\*GET_VALUE_OF_ATTRIBUTE()*
# CL_SBOOK_PARTNER_PPF (Example)

<table>
<thead>
<tr>
<th>CL_SBOOK_PARTNER_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-PARTNER_NO : PPFDPARTNO</td>
</tr>
<tr>
<td>-PARTNER_ROLE : PPFDPARTRL</td>
</tr>
<tr>
<td>-PARTNER_TYPE : PPFDPARTTY</td>
</tr>
<tr>
<td>-ZAV_ADDRESSNO : AD_ADDRNUM</td>
</tr>
<tr>
<td>-ZAV_PERSNO : AD_ADDRNUM</td>
</tr>
<tr>
<td>-ZAV_ADDR_TYPE : PPFDADRTYP</td>
</tr>
<tr>
<td>-PARTNER_TEXT : TEXT60</td>
</tr>
</tbody>
</table>

+IF_PARTNER_PPF~GET_ZAV_ADDRESS()
+IF_PARTNER_PPF~GET_PARTNER()
CL_SBOOK_PPF (Example)

<table>
<thead>
<tr>
<th>CL_SBOOK_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-BOOKID : S_BOOK_ID</td>
</tr>
<tr>
<td>-CARRID : S_CARR_ID</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

+IF_LOCK_PPF~ENQUEUE() 
+IF_LOCK_PPF~DEQUEUE()
CL_SF_MAIL_CUST_PPF

- PROCLS : SEOCLSNAME
- PROCMETH : SEOCMPNAME
- SMARTFORM : TDSFNAME
### CL_SF_PRINT_CUST_PPF

- ARCHVMODE : SYARMOD
- PRINTPARAM : PPFDPRTPRM
- PROCCLS : SEOCLSNAME
- PROCMP : SEOCMPNAME
- SMARTFORM : TDSFNAME
## CL_TRIGGER_PPF

<table>
<thead>
<tr>
<th>CL_TRIGGER_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-APPL : Object</td>
</tr>
<tr>
<td>-APPLCTN : PPFDAPPL</td>
</tr>
<tr>
<td>-APPLKEY : SYSUID_C</td>
</tr>
<tr>
<td>-CONTEXT : PPFDCNTXT</td>
</tr>
<tr>
<td>-DISPATCH : PPFDDSPTCH</td>
</tr>
<tr>
<td>-IS_CHANGED : PPFDTTCHNG</td>
</tr>
<tr>
<td>-IS_INACTIV : PPFDIACT</td>
</tr>
<tr>
<td>-IS_LOCKED : PPFDTLCK</td>
</tr>
<tr>
<td>-IS_REPEAT : PPFDTRPT</td>
</tr>
<tr>
<td>-MEDIUM : IF MEDIUM_PPF</td>
</tr>
<tr>
<td>-STATUS : PPFDTSTAT</td>
</tr>
<tr>
<td>-TIMECHANGE : PPFDTCHNGD</td>
</tr>
<tr>
<td>-TIMECREATE : PPFDTCREAT</td>
</tr>
<tr>
<td>-TTYPE : PPFDTT</td>
</tr>
<tr>
<td>-USERCHANGE : PPFDUCHNGD</td>
</tr>
<tr>
<td>-USERCREATE : PPFDUCREAT</td>
</tr>
</tbody>
</table>

+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 : PPFDSPCTCH
-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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_TRIGGERS()</td>
<td></td>
</tr>
<tr>
<td>GET_TTYPE_CUSTOMIZING()</td>
<td></td>
</tr>
<tr>
<td>CHECK_TRIGGER()</td>
<td></td>
</tr>
<tr>
<td>CREATE_TRIGGER()</td>
<td></td>
</tr>
<tr>
<td>CONSTRUCTOR()</td>
<td></td>
</tr>
<tr>
<td>ADD_TRIGGER()</td>
<td></td>
</tr>
<tr>
<td>DETERMINE()</td>
<td></td>
</tr>
<tr>
<td>DELETE_TRIGGER()</td>
<td></td>
</tr>
<tr>
<td>REPEAT_TRIGGER()</td>
<td></td>
</tr>
</tbody>
</table>

**Attributes**

- **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
IF_DETERMINATION_PPF

IF_DETERMINATION_PPF

• GET_PERSISTENCY_TABLE()
• DETERMINE()
IF_LOCK_PPF

<table>
<thead>
<tr>
<th>IF_LOCK_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQUEUE()</td>
</tr>
<tr>
<td>DEQUEUE()</td>
</tr>
</tbody>
</table>
### IF_MEDIUM_CUST_PPF

<table>
<thead>
<tr>
<th>IF_MEDIUM_CUST_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>+GET_PERSISTENCY_TABLE()</td>
</tr>
<tr>
<td>+GET_MEDIUM()</td>
</tr>
<tr>
<td>+GET_MEDIUM_TEMPLATE()</td>
</tr>
</tbody>
</table>
IF_MEDIUM_PPF

IF_MEDIUM_PPF

+CHECK()
+MEDIUM_CHANGED()
+COPY()
+EXECUTE()
+PREVIEW()
+IS_EQUAL()
+LANGUAGE_IS_EQUAL()
+COMPLETE()
+PARTNER_IS_EQUAL()
+GET_PARTNER()
+GET_PROCESSING_LOG()
+SET_PARTNER()
+GET_TYPE()
IF_MERGE_PPF

IF_MERGE_PPF

+GET_PERSISTENCY_TABLE()
+MERGE()
+MERGE_SINGLE()
IF_PARTNER_PPF

<table>
<thead>
<tr>
<th>IF_PARTNER_PPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>+GET_ZAV_ADDRESS()</td>
</tr>
<tr>
<td>+GET_PARTNER()</td>
</tr>
</tbody>
</table>