

PPF
Post Processing Framework
Guidelines for application developers

June 30, 2003

Contact Persons: Daniel-Alexander Heller (PPF)

Contents:

| | | |
|------------|---|-----------|
| 1 | <i>PPF Overview</i> | 4 |
| 2 | <i>Preparation</i> | 5 |
| 2.1 | Necessary Classes | 5 |
| | The implementation steps are explained using the example of a demo application. The demo application and all mentioned classes are available in all systems (transaction SPPFDEMO, development class SPPF_DEMO). | 5 |
| 2.1.1 | Application Class..... | 5 |
| 2.1.2 | Application-Specific Processing..... | 9 |
| 2.2 | Customizing | 15 |
| 2.2.1 | Define New Application..... | 15 |
| 2.2.2 | Defining the Action Profile..... | 16 |
| 2.2.3 | Define Action Definitions for the Action Profile..... | 17 |
| 2.2.4 | Details for the Action Definition..... | 18 |
| 2.2.5 | Assignment of Processing Types for an Action Definition..... | 20 |
| 2.3 | Determination and Merging of Actions | 23 |
| 2.3.1 | Determination (Condition Configuration)..... | 23 |
| 2.3.2 | Action Merging..... | 28 |
| 3 | <i>Interaction Between Application and PPF at Runtime</i> | 29 |
| 3.1 | Calling the PPF | 29 |
| 3.2 | Processing Actions | 32 |
| 3.2.1 | Immediate Processing..... | 32 |
| 3.2.2 | Processing with Document Posting..... | 32 |
| 3.2.3 | Later Processing..... | 32 |
| 3.2.4 | Manual Triggering of Processing in the Dialog..... | 32 |
| 3.3 | User Interface at Runtime | 33 |
| 3.3.1 | Standard User Interface..... | 33 |
| 3.3.2 | Connection of Generic Object Services (GOS)..... | 35 |
| 3.4 | Transaction Concept | 36 |
| 3.4.1 | Overview..... | 36 |
| 3.4.2 | Object Pool..... | 36 |
| 4 | <i>Administration User Interface</i> | 38 |
| 5 | <i>Extendibility</i> | 39 |
| 5.1 | Business Application Add Ins (BAdIs) | 39 |
| 5.1.1 | Exit for the printer determination (PRINTER_DETERM_PPF)..... | 39 |
| 5.1.2 | Exit After Generated Action (TRIGGER_EXECUTED)..... | 39 |
| | The BADI has the application names as the filter value. The action is also transferred and can deliver its processing status (successful, with error) or other information..... | 39 |
| 5.1.3 | Exit for Context Extension (CONTEXT_EXTEND_PPF)..... | 39 |
| 5.1.4 | Exit for Completion of Processing Options (COMPLETE_PROC_PPF)..... | 40 |
| 5.1.5 | Extend Container for Condition Evaluation (CONTAINER_PPF)..... | 40 |
| 5.1.6 | Exit for Execution of Actions (EXEC_METHODCALL_PPF)..... | 40 |
| 5.1.7 | Exit for Getting Possible Partner Functions of an Application (GET_PARTN_ROLES_PPF)..... | 41 |
| 5.1.8 | Exit for Double Clicking on Values in the Display (GRID_CLICK_PPF)..... | 41 |
| 5.1.9 | Exit for Checking if Deletion of Action Profile is allowed (CONTEXT_DELETE_PPF)..... | 41 |
| 5.1.10 | Exit for evaluation of schedule conditions (EVAL_SCHEDCOND_PPF)..... | 41 |
| 5.1.11 | Exit for evaluation of start conditions (EVAL_STARTCOND_PPF)..... | 41 |
| 5.1.12 | Exit for Adding further data to workflow container (WF_CONT_MODIFY_PPF)..... | 41 |
| 5.2 | PPF Interface | 42 |
| 5.2.1 | Connection of Your Own Processing Options..... | 42 |
| 5.2.2 | Connection of a Logic for the Determination..... | 42 |

| | | |
|------------|---|-----------|
| 5.2.3 | Connection to a Separate Logic for Action Merging | 42 |
| 6 | <i>Appendix</i> | 44 |
| 6.1 | Description of Interfaces | 44 |
| 6.1.1 | CL_MANAGER_PPF | 44 |
| 6.2 | Class Diagram | 47 |
| 6.2.1 | Customizing Classes | 47 |
| 6.2.2 | Runtime Classes | 48 |
| 6.2.3 | Service Classes | 49 |
| 6.3 | Sequence Diagrams | 50 |
| 6.3.1 | Calling the PPF | 50 |
| 6.3.2 | Method DETERMINE Part 1 | 51 |
| 6.3.3 | Method DETERMINE Part 2 | 52 |

1 PPF Overview

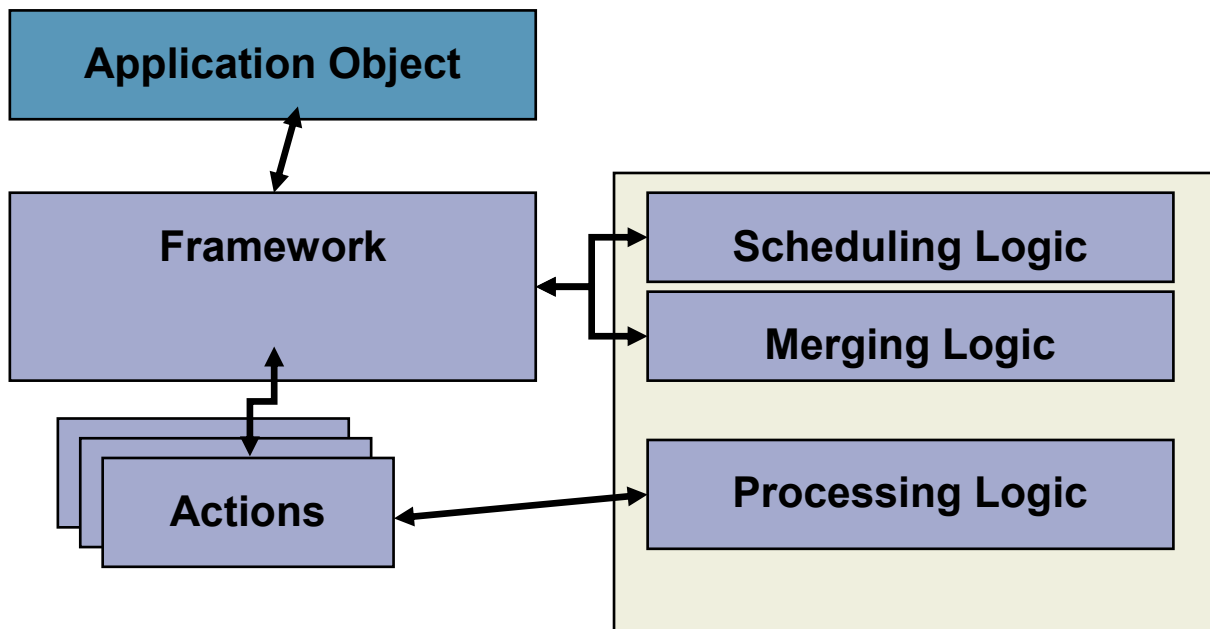
The Post Processing Framework (PPF) is a tool for the generic execution of functions and processes. It provides the applications with a uniform interface to any actions. Actions can be outputs in the traditional sense such as print, fax, mail, or XML but functions such as the triggering of workflows or any method call can also be triggered. Which actions this eventually are, is determined by an individually configurable or a self-programmable determination technology, depending on the application document data. Execution of the action can also take place depending on the data of the application document.

Therefore, the PPF automatically generates actions from document data (for example, delivery notes or order confirmations, generation of an item in the document, creation of a subsequent document).

PPF additionally provides uniform action administration. There is status management and a processing log for every action.

The PPF itself was programmed with ABAP objects. In addition, various object services were used. However, you do not have to program your PPF application in an object-oriented manner to use the PPF.

Overview graphic:



2 Preparation

2.1 Necessary Classes

Create the following classes in the Class Builder:

- A persistent class (in the sense of Object Services) that represents the application object (for example: CL_BOOK_PPF)
- A partner class that represents a partner of an application object (for example: CL_BOOK_PARTNER_PPF)
- A context class that encapsulates all information for the PPF (Example: CL_DEMO_CONTEXT_PPF)
- A processing class provided Smart Forms are used (Example: CL_PROCESSING_DEMOBOOK_PPF)
- A BADI implementation provided the processing method call is used
- Workflow template provided a workflow is to be triggered

The implementation steps are explained using the example of a demo application. The demo application and all mentioned classes are available in all systems (transaction SPPFDEMO, development class SPPF_DEMO).

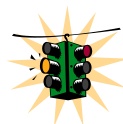
The classes to be implemented by the application are shaded in gray in the following graphics. All other classes are provided by the PPF.

2.1.1 Application Class

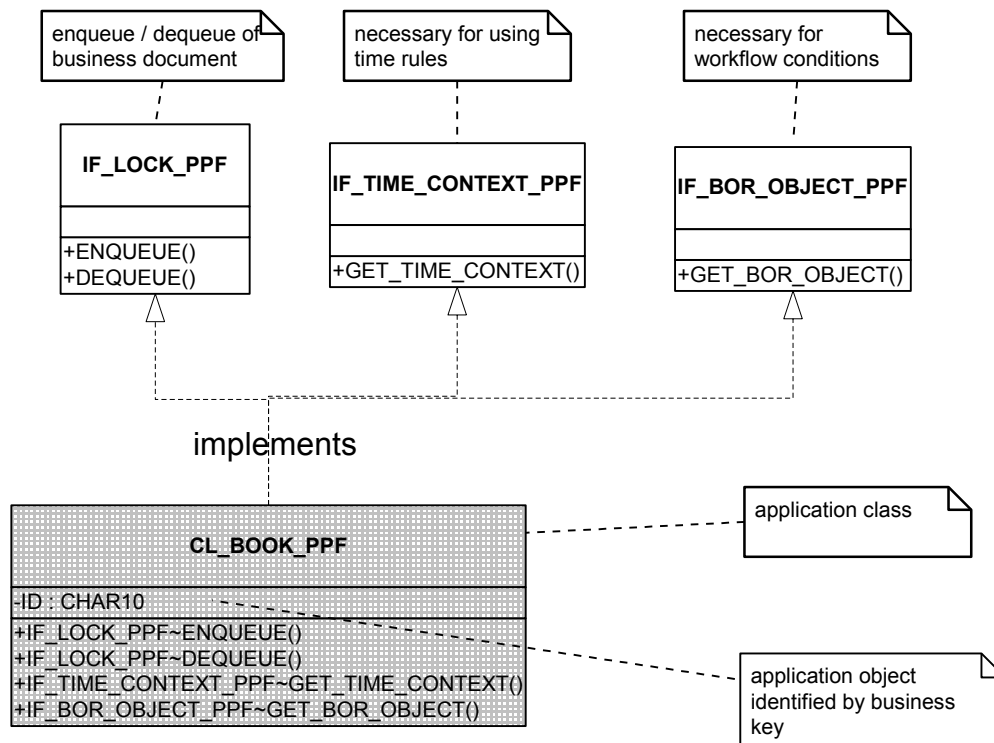
The PPF expects a persistent class so it can handle different types of application objects.

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. The key of the application object also appears as an attribute of the proxy class. The application object implements the interface IF_LOCK_PPF and thus both methods for locking and unlocking the application object. These two methods are important so that an action does not operate on documents that are already locked. If the interface is not implemented, this can result in actions being executed twice under certain circumstances. Generally, the interface must be implemented.

When using the workflow conditions, a BOR object is necessary in addition to the persistent application object since the schedule condition and start condition is defined on attributes of the BOR object (more on this in the relevant sections).



The implementation of IF_LOCK_PPF is important so that the action does not change document data for a document that is being processed. The implementation also prevents the same action being executed multiple times in parallel processing.



Generation of the object at runtime:

DATA:

* reference to application/proxy object

```
appl_object TYPE REF TO cl_book_ppf,
```

```
book_id     TYPE CHAR10.
```

* create a persistent application object via the class agent of it's co-class

* object services create 2 service classes for any persistent class

ca class contains all persistency services, e.g. creation of persistent object

create application object, key (book_id) must be set before

```
appl_object ?=
```

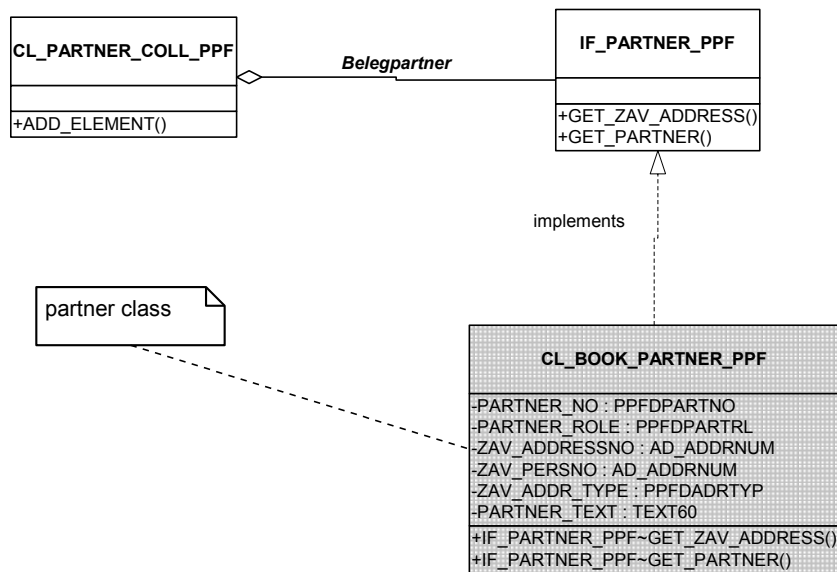
```
ca_book_ppf=>agent->if_os_factory~create_persistent_by_key(
    i_key    = book_id ).
```

Partner class

Create a partner class for the document partner that implements the interface IF_PARTNER_PPF. The document partners are collected in a collection and transferred to the PPF. The class of the partner collection already exists.

A partner consists of a partner function, partner number, and the data assigned from the central address management (CAM: ZAV "Zentrale Adressverwaltung" in German): Person number, address number, and address type. This data must be made available for every document partner by the application.

The PPF determines the relevant communication data for fax processing or mail processing using the address numbers transferred.



Generation of a partner object at runtime and appendage to the partner collection:

DATA:

```

* reference to partner object
partner TYPE REF TO cl_book_partner_ppf,

* reference to partner collection
partner_coll TYPE REF TO cl_partner_coll_ppf,

* create partner collection
CREATE OBJECT partner_coll.

* create first partner object
CREATE OBJECT partner
EXPORTING ip_partner_role = 'LF'
         ip_partner_no   = '1234567890'
         ip_partner_text = 'Vendor Meyer'
         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 ).
  
```

Context class

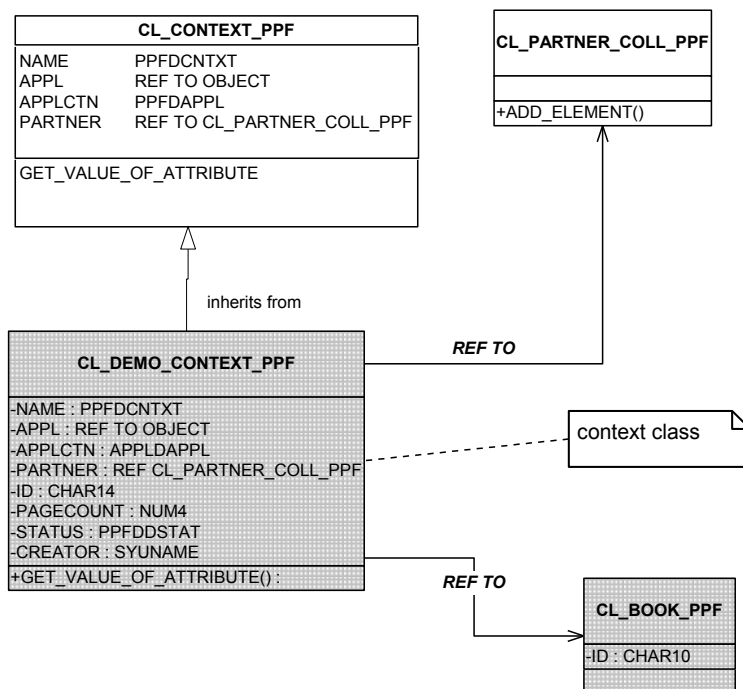
The context class encapsulates all the application data necessary for the PPF:

- Application name
- Reference to the application object (instance of the application class or proxy object)

- Reference to a partner collection
- Further application attributes (fields that can be included as sort fields in the management table of actions)

Note when creating the context class that this is not marked as final. The customer should potentially have the possibility to extend this using further attributes. This is only possible provided the class has not been marked as final.

The context class redefines the method `GET_VALUE_OF_ATTRIBUTE` of class `CL_CONTEXT_PPF`. This serves dynamic attribute accesses that are not yet supported in the ABAP standard. For the redefinition, simply copy the coding from the template and set it again in the method. This step is necessary so that the method is executed for your action profile class itself and can access its attributes.



Fill the context instance at runtime:

DATA:

```

* reference to context object
context TYPE REF TO cl_demo_context_ppf.

* create context object
create object context.

* set context attribute
context->applctn = 'BOOK'.      "Declared in Customizing (see chapter 2.2)
context->name     = 'BOOK'.      "Declared in Customizing (see chapter 2.2)
context->appl     = appl_object.
context->partner  = partner_coll.
  
```



```

* additional context fields
*.these fields can be used as sort fields for a later processing of the actions
* the fields can be overtaken into the database table for actions (PPFTTRIGG)
context->ID = '1234'.
context->creator = sy-uname.
....

```

2.1.2 Application-Specific Processing

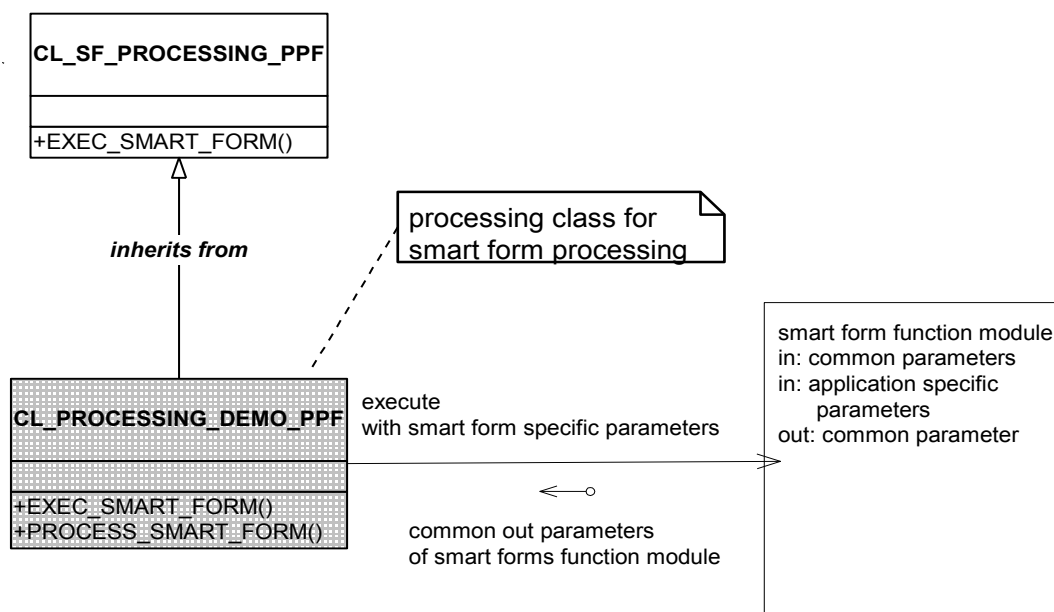
2.1.2.1 Using Smart Forms Processing Options

If Smart Forms are used for the action, you must program a processing class. Here, the PPF helps you as much as possible.

Smart Forms are function modules whose interface consists of a general part that is independent from the relevant Smart Form and an application-specific part. Rather, the PPF cannot directly call the function module. However, it fills the general part of the interface.

Your processing class must also inherit the processing method EXEC_SMART_FORMS from the PPF class CL_SF_PROCESSING_PPF that serves as the copy template and sets the general parameter of the Smart Forms function module. You only need to fill the specific parameter.

You must also set the language of your Smart Form here. If your application object involves a BOR object then you can link the action (Smart Form) with your BOR object, or optically archive the action using Archive Link.



The copied processing methods in the example application are as follows:

```
METHOD process_smart_form .
```

```
* function name
```

```
DATA: function_name TYPE rs381_fnam,
```

```
dummy(254)    TYPE c.

* get the function name for this smart form
CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
  EXPORTING
    formname      = ip_smart_form
*   VARIANT      = ' '
*   DIRECT_CALL   = ' '
  IMPORTING
    fm_name       = function_name
  EXCEPTIONS
    no_form       = 1
    no_function_module = 2
    OTHERS        = 3
    .

IF sy-subrc <> 0.
*   add an error message to processing protocol
  MESSAGE i015(sppf_media) WITH ip_smart_form.
  CALL METHOD cl_log_ppf=>add_message
    EXPORTING
      ip_problemclass = '1'
      ip_handle       = cp_application_log.
  EXIT.
ENDIF.

*----- get application specific data-----
DATA: lo_book TYPE REF TO cl_book_ppf,
      ls_book TYPE ppftbook.
...
* cast imported object so that we can refer to its attributes
lo_book ?= io_appl_object.

* get key fields of application object
ls_book-id = lo_book->get_id( ).
ls_book-author = lo_book->get_author( ).
ls_book-title = lo_book->get_title( ).
ls_book-pagecount = lo_book->get_pagecount( ).
ls_book-creator = lo_book->get_creator( ).
ls_book-datecreate = lo_book->get_datecreate( ).
ls_book-datechange = lo_book->get_datechange( ).
ls_book-status = lo_book->get_status( ).
ls_book-isbn = lo_book->get_isbn( ).
*-----

*----- is_mail_appl_obj -----
```

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

*-----fill archive parameters for archive link -----
IF is_output_options-tdarmod = '2' OR
   is_output_options-tdarmod = '3'.
*   archive_index_tab
   READ TABLE ct_archive_index_tab INTO ls_archive_index INDEX 1.
*   just fill the id of your actual BOR object
*   ----->
*   ls_archive_index-object_id      = 'ID_OF_YOUR_BOR_OBJECT'.
*   ----->
   IF ls_archive_index-object_id IS INITIAL.
       DELETE ct_archive_index_tab INDEX 1.
   ELSE.
       MODIFY ct_archive_index_tab FROM ls_archive_index INDEX 1.
   ENDIF.
ENDIF.
*-----

*-----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-----
       is_book            = ls_book
*-----
```

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

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

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

ENDMETHOD.
```

The return parameter is evaluated centrally using the PPF. You can add your own entries to the processing log and should also do this so that the application specific processing routines are also logged. A handle on the application log is transferred for this as the parameter: `ip_application_log`. Entries can simply be made using the interface of the service class `CL_LOG_PPF`:

Example:

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

2.1.2.2 Usage of Processing Method Call

The method calls are created using BADI implementations. The relevant BADI definition (transaction SE18) is called EXEC_METHODCALL_PPF. You can create an implementation for this definition (transaction SE19). The interface looks as follows:

| | | | | |
|--------------------|-----------|-------------|----------------------|-------------------------------------|
| FLT_VAL | Importing | Type | PPFDFTVAL | Parameter FLT_VAL of method EXECUTE |
| IO_APPL_OBJECT | Importing | Type Ref To | OBJECT | Reference to application object |
| IO_PARTNER | Importing | Type Ref To | CL_PARTNER_PPF | Message partner |
| IP_APPLICATION_LOG | Importing | Type | BALLOGHNDL | Handle on application log |
| IP_PREVIEW | Importing | Type | CHAR1 | Preview flag |
| II_CONTAINER | Importing | Type Ref To | IF_SWJ_PPF_CONTAINER | Container with parameter values |
| IP_ACTION | Importing | Type | PPFDTT | Name of action definition |
| RP_STATUS | Returning | Type | PPFDSTAT | PPF: Output status |

The filter value is created when the BADI implementation is created and is therefore freely definable. A description for the filter value should also be entered. For processing, you receive a reference to its application object and hence have access to the document data. In addition, the document partner, a reference to an application log (processing log), and a container with parameters are transferred. The parameter values can be determined in Customizing. After processing, the processing status must be set correspondingly (processed successfully = 1, processed with errors = 2). If the status is not explicitly set, the PPF sets the status to processed with errors = 2.

In the BADI-Builder (transaction SE19), this can appear as follows:

Implementation name: COPY_DOCUMENT Active

Implementation short text: Generate Subsequent Document (Copy)

Definition name: EXEC_METHODCALL_PPF

Attributes Interface

General Data

Package: CRM_ACTION_IMPL

Language: DE German

Last changed by: SAP Last activated by: SAP

Last change: 24.10.2000 14:35:11 Capitalized on: 24.10.2000 14:35:28

Type

☐ Within SAP

☐ Multiple use

☒ Filter-Depend. Filter type: PPFDFTVAL ☒ Enhanceable

Filter Value for BADI EXEC_METHODCALL_PPF

| Method | Short text Method |
|---------------|------------------------------|
| COPY_DOCUMENT | Generate subsequent document |

MARKR pwwf0313 INS

2.1.2.3 Use of the Processing Workflow

No application specific coding is needed here.

2.1.2.4 Use of the Processing of External Sending

The processing of external sending, like Smart Forms, also needs an application-specific logic for data retrieval and for calling the Smart Form. Creation no longer takes place using a method implementation, but rather using a BADI implementation. The BADI is called DOC_PERSONALIZE_BCS. The interface is very similar to the method for Smart Forms processing:

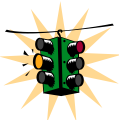
| | | | | |
|-------------------------|-----------|-------------|-------------------------|--|
| FLT_VAL | Importing | Type | BCSDFLTVAL | BCS: Filter value for BADI DOC_PERSONALIZE_BCS |
| IS_ARCHIVE_PARAMETERS | Importing | Type | ARC_PARAMS | ImageLink structure |
| IS_CONTROL_PARAMETERS | Importing | Type | SSFCTRLP | Smart Forms: Control structure |
| IS_OUTPUT_OPTIONS | Importing | Type | SSFCOMPOP | SAP Smart Forms: Options smart composer (transfer) |
| IO_APPL_OBJECT | Importing | Type Ref To | OBJECT | Application object |
| IP_SMART_FORM | Importing | Type | TDSFNAME | Smart Forms: Form name |
| IS_MAIL_APPL_OBJ | Importing | Type | SWOTOBJID | Structure for object ID |
| IS_MAIL_RECIPIENT | Importing | Type | SWOTOBJID | Structure for object ID |
| IS_MAIL_SENDER | Importing | Type | SWOTOBJID | Structure for object ID |
| IP_USER_SETTINGS | Importing | Type | TDBOOL | Selection field (yes or no) |
| IP_APPLICATION_LOG | Importing | Type | BALLOGHNDL | Application log: Log handle |
| IO_PERSONALIZE_DATA | Importing | Type Ref To | CL_PERSONALIZE_DATA_BCS | Service class for the personalization of a mail |
| ES_DOCUMENT_OUTPUT_INFO | Exporting | Type | SSFCRESPD | Smart Forms: Return of document information |
| ES_JOB_OUTPUT_INFO | Exporting | Type | SSFCRESCL | Smart Forms: Return when form printing is completed |
| ES_JOB_OUTPUT_OPTIONS | Exporting | Type | SSFCRESOP | Smart Forms: Return when starting form printing |
| ET_ERROR_TAB | Exporting | Type | TSFERROR | SAP Smart Forms: Runtime error |
| CT_ARCHIVE_INDEX_TAB | Changing | Type | TSFDARA | SAP Smart Forms: Table with archive indices |
| C_DOCUMENT_TITLE | Changing | Type | TDTITLE | Document title |

The logic for calling the Smart Forms is the same as the processing above, for an example, see the implementation BCS_PROC (transaction SE19).

2.2 Customizing

In Customizing (transaction SPPFC) you make your classes known to the PPF and define the determination of the outputs: You assign the possible action definitions (for example, delivery note, order acknowledgment, and so on) to your application. You must define a determination technology for each output type (see section “Determination and Merging of Actions”).

The definition in Customizing only determines the framework, that is, all potential actions and their possible processing options are declared. This definition can be used in an additional Customizing step to define conditions (see 2.3 onwards). The actions only appear when the conditions have been maintained. Definition in Customizing alone does not lead to the generation of actions.



Transaction SPPFC should never be added to the IMG directly. It can be called in a separate application specific transaction when the parameter is set for the application.

2.2.1 Define New Application

An application that wants to use the PPF must be entered in Customizing of the PPF. In the initial screen, you only enter a name and a description. In addition, you can enter the date profile of the application and the BOR object or the business class here. The date profile and the object type are required for the definition of the schedule condition. The conditions are based on attributes of the object type and date rules of the date profile can also be integrated. Here you must also enter the name of your context class.

Since Basis Release 6.10, the application is maintained in a separate transaction (SPPFCADM).

Display View "Application": Overview

| Appl. | Description | Date Profile | Obj. Type |
|-----------|----------------------|--------------|------------|
| BBP_PD | Procurement Document | | BUS2201 |
| BILLING | Billing | | |
| BOOK | Book management | | |
| CRM_ORDER | CRM Order | | BUS2000115 |
| DNO_NOTIF | Message | | BUS7060 |

Application name Description Date profile of application Object type of the application

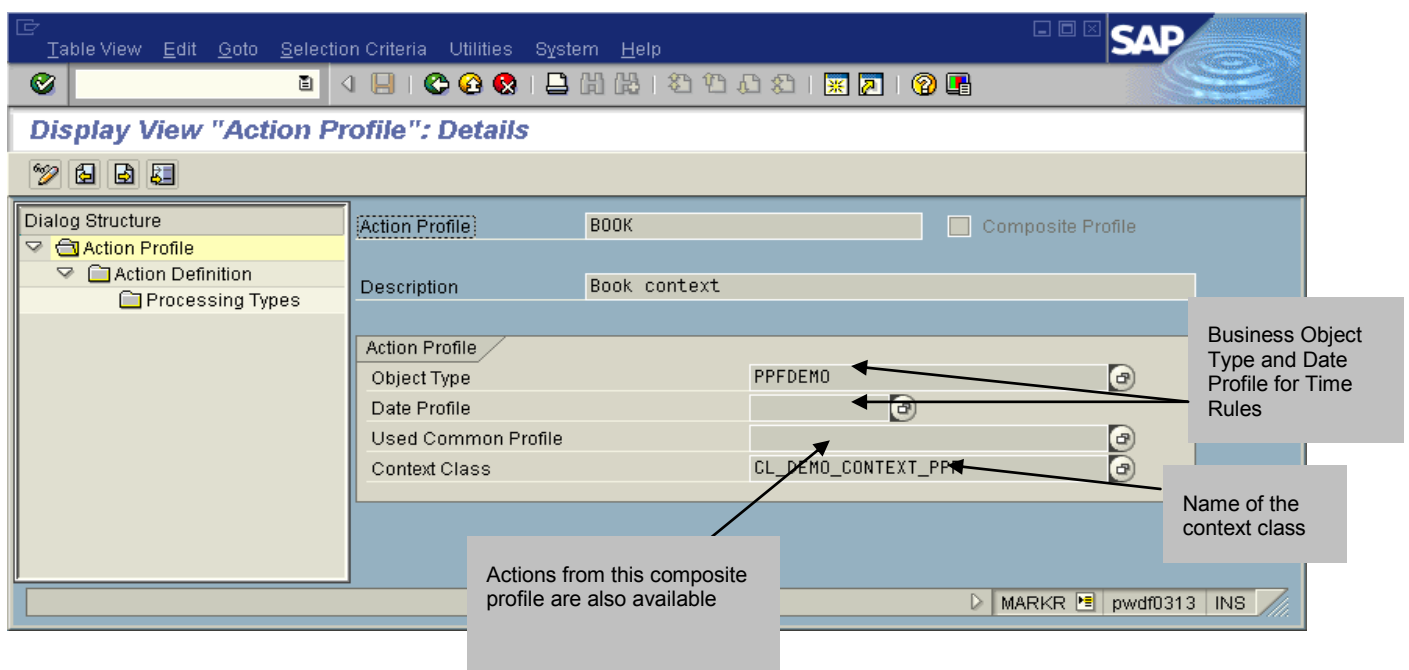
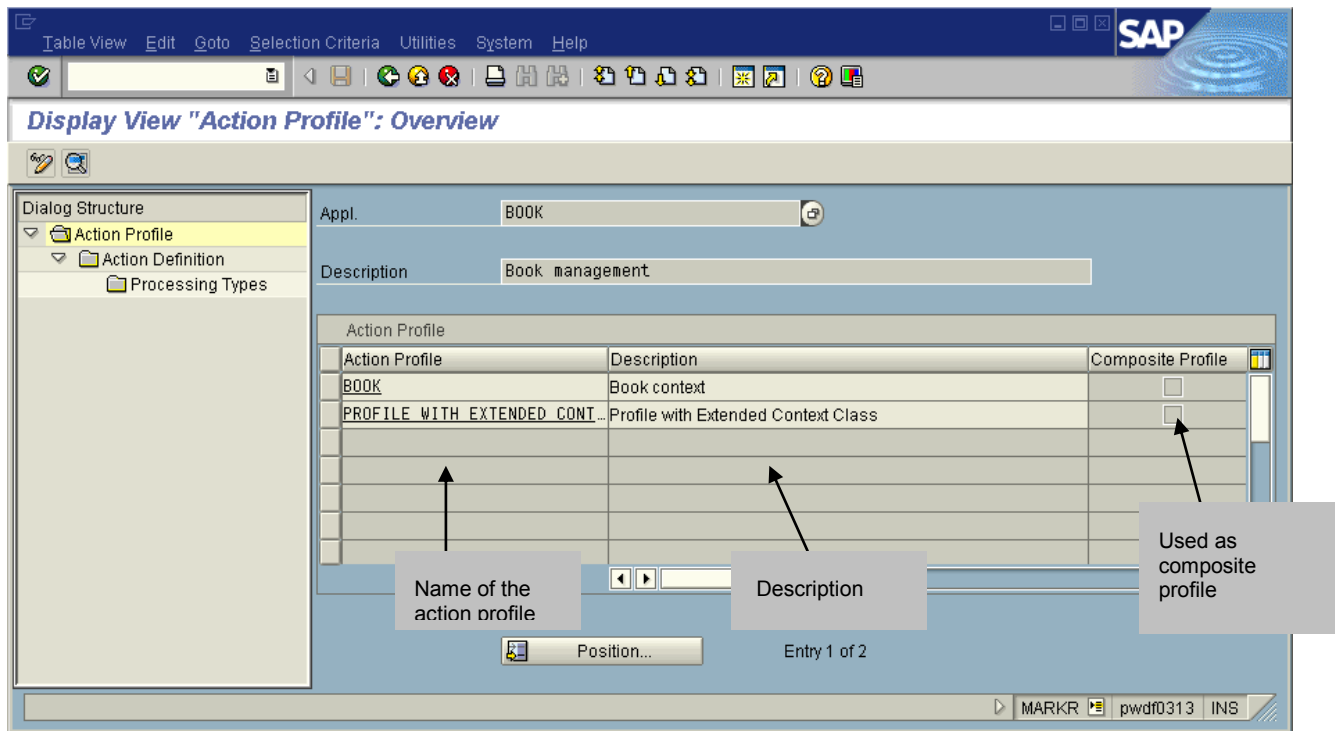
Position... Entry 1 of 5

2.2.2 Defining the Action Profile

An action profile is a collection area for actions in a specific application context (for example, leasing actions, actions for contracts, and so on).

You can transfer all relevant application data to the PPF using an instance of the context class. If you maintained the context class at the application level, it does not have to be entered again here when it is created from new. It is copied as the default value. The same applies for the date profile and object type.

The composite profile flag specifies that a profile is used as composite profile. A composite profile can be inserted in other action profiles. Thus the actions from the composite profile can be used in the current profile.



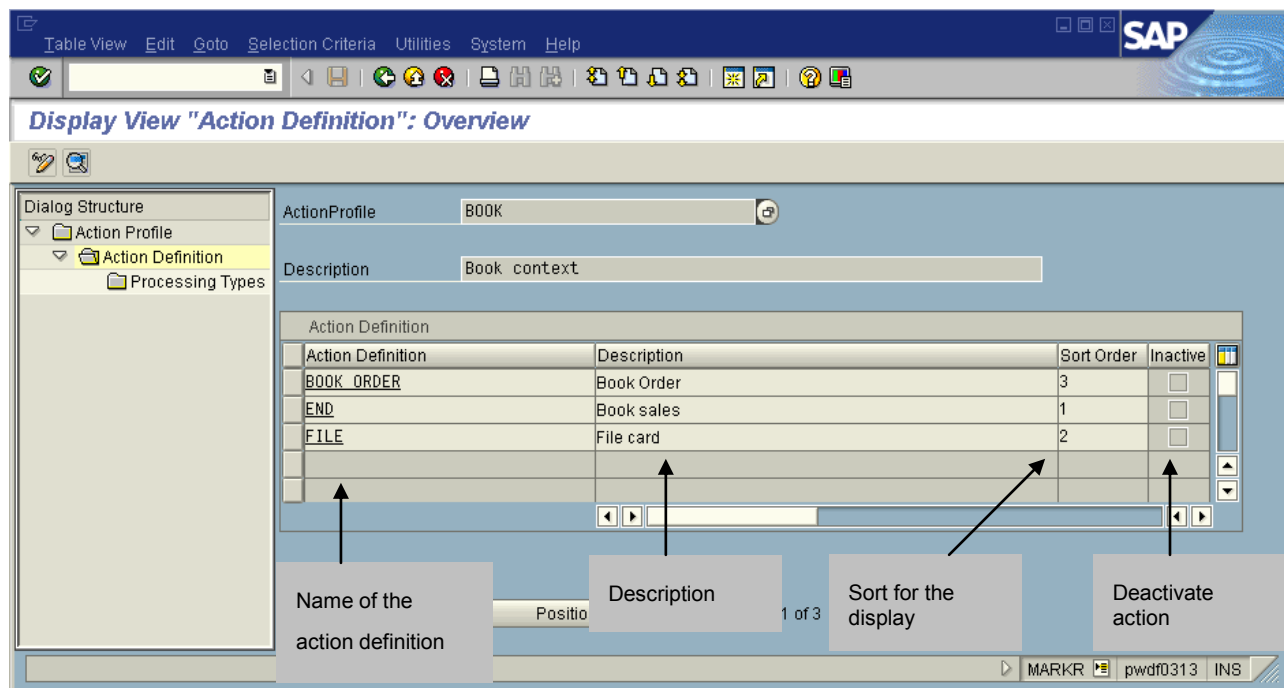
2.2.3 Define Action Definitions for the Action Profile

An action profile can be assigned various action definitions. An action definition is the smallest business unit that is to be output or processed – for example, if all delivery notes belong to the 'delivery note' action definition, whether they should now be printed, faxed, or output as normal.

You can deactivate an action using the action setting. If the action is set to inactive, it is ignored for action determinations. It can therefore be 'deactivated'.

The actions in the document can be sorted using the sort function.

By double-clicking on the action definition, you can display the detailed settings.



2.2.4 Details for the Action Definition

You can configure an action definition in various ways:

- Processing time (immediate processing, when the document is posted, later using a selection report)
- Processing times that are not permitted: Here you can enter processing times that make no sense for this action. For example, Send order confirmation -> Process time immediate. The action order confirmation may be processed after document editing is complete or using the selection report. The time immediate is entered here as a time that is not allowed.
- Schedule automatically: If the flag is set, the action is scheduled automatically provided the schedule condition is fulfilled. If it is not set, the action appears in the worklist and can be scheduled manually by the agent for document editing.
- Sorting in the display (in which order should the actions in the document for runtime be displayed, provided a user interface is included)
- Automatic scheduling (X = action is to be scheduled, SPACE = action should be put in the worklist, that is, the agent can schedule it manually)
- Delete after processing (the action is executed and then deleted)
- Can be changed in the dialog (after automatic determination, should you be allowed to make changes and repeat the action definition manually?)
- Can be executed in the dialog (the action can take place during document editing, even though no posting has yet taken place)
- Display on the toolbar: Specifies whether the action is to be displayed on the GOS toolbar (provided the service is used)
- Partner function: Default function of the partner to which the action goes – moved in the manual creation of an action
- Partner-dependent: specifies whether a partner determination is to be performed
- Selection of a determination technology, that is to be used for this action definition (see section below on determination)
- Selection of a technology for merging of actions: (see section below on action merging)
- Sort field 1 – 3: Application specific data that is supplied in the action profile class. Display or processing can be sorted according to this data
- The action description delivers details on the action and can be displayed by the agent for document editing

Table View Edit Goto Selection Criteria Utilities System Help

SAP

Display View "Action Definition": Details

Dialog Structure

- ▼ Action Profile
 - ▼ Action Definition
 - Processing Types

Settings in detail

ActionProfile: BOOK
Description: Book context

Action Definition: END
Description: Book sales

Action Definition | **Action Description** | Action Summarizatr

Action Settings

Time of Processing: Processing using selection report

Processing Time Not Permitted: No Restrictions

Sort Order For Display: 1

☒ ScheduleAutomatically ☒ Chngbl in Dialog

☐ Delete After Processing ☒ Executable in Dialog

☒ Display in Toolbox

Partner Determination for the Action

☒ Partner dependent PartnerFunction: MP
Description: MailPartner

Action Determination and Action Summarization

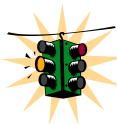
Determin. Tech.: Determination Using Conditions that Can Be Transport...

Action Summarizatr: Set Highest Number of Processed Actions

Sort Fields for the Execution of Actions

Sort Field 1
Sort Field 2
Sort Field 3

MARKR pdf0313 INS



The search help for the partner function field can only deliver values if the application creates and delivers a BAdI implementation for GET_PARTN_ROLES_PPF. Here you can use the example implementation for the demo application (GET_ROLES_BOOK_PPF) to get more information.

2.2.5 Assignment of Processing Types for an Action Definition

Actions always occur using the execution of a processing. Currently, Smart Forms are supported as print, fax or mail, plus the triggering of workflows and any method call. The new processing external sending can replace the existing Smart Forms processing and additionally offers further extended functions.

Add a separate configuration for every processing. The settings made here serve as default values and can be overridden in the conditions.

2.2.5.1 Smart Forms Processing Options

In the Smart Forms processing options, enter the name of the Smart Form used and a processing class with a processing method, that you have programmed (see section “processing class”). Provided the application supports optical archiving using Archive Link, the archive parameter archive object type and document type must be entered. These must first be created in Archive Link Customizing. You can reach Archive Link Customizing using the transactions (oac2, oac3, oaco). The option archive copies optically archives all multiple outputs, that is, it also archives identical multiple archives.

Display View "Processing Types": Overview

Dialog Structure

- ▼ Action Profile
 - ▼ Action Definition
 - Processing Types

Action Definition: END

Description: Book sales

Permitted Processing Types of Action

| Processing Type | Assignment |
|--|-------------------------------------|
| Assignment/Change Using Value Help in List | <input type="checkbox"/> |
| External Communication | <input type="checkbox"/> |
| Smart Forms Fax | <input type="checkbox"/> |
| Smart Forms Mail | <input checked="" type="checkbox"/> |
| Smart Forms Print | <input type="checkbox"/> |

Assignment of possible processing options to an action definition

Set Processing

Mail Settings

| | |
|-------------------|----------------------------|
| Form Name | SPPFDEMO BOOK END |
| Processing Class | CL_PROCESSING_DEMOBOOK_PPF |
| Processing method | PROCESS_SMART_FORM |
| Archive Mode | Mail Only |

Detailed settings for Smart Forms processing

MARKR pdf0313 INS

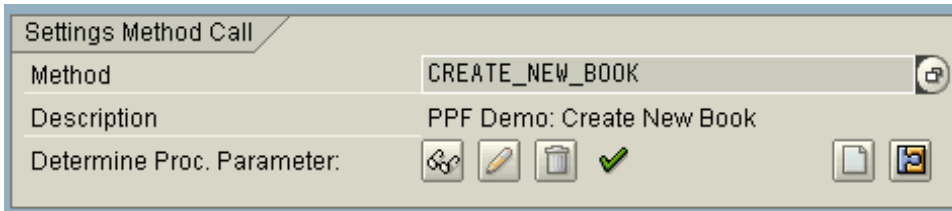
2.2.5.2 Method Call Processing





This processing enables the execution of any action. A BAdI implementation is called. This enables the creation of a subsequent document or the creation of an item in the document, for example.

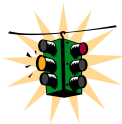
The method can be entered using the F4 help. All active BAdI implementations for BAdI definition EXEC_METHODCALL_PPF are displayed there.

The processing parameters are freely definable and can be provided with values. Static values are used here. The values can be changed again in the configuration of conditions.

Example: Create method subsequent offer, processing parameter: Type of subsequent offer



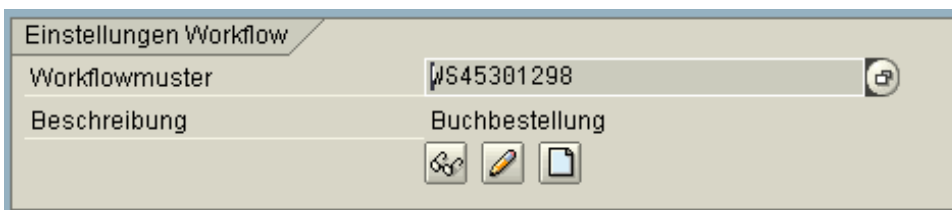
| Settings Method Call | |
|----------------------------|---|
| Method | CREATE_NEW_BOOK |
| Description | PPF Demo: Create New Book |
| Determine Proc. Parameter: |     |






Make sure that the implementation was activated, otherwise it is not displayed in the F4 Help.

2.2.5.3 Workflow Processing

Workflow templates can also be started using actions. The workflow template is added to it. The business object must be defined as an input parameter in the container definition of the workflow template. The name of the parameter must be BUSINESSOBJECT. Using Display/Change, the assigned workflow template can be edited and a new workflow template can be created using the Create button. The F4 help for the assignment only displays the workflow templates that support the assigned business object in the action profile.



| Einstellungen Workflow | |
|------------------------|---|
| Workflowmuster | WS45301298 |
| Beschreibung | Buchbestellung |
| |    |

2.2.5.4 External Sending Processing

The processing of external sending can take the place of the existing Smart Forms processing. It completely covers its functions and additionally makes it possible to send an attachment, send the outputs to copy recipients and in further releases it is also possible to send SMSs.

The Customizing settings are similar to the Smart Forms processing options. The name of a Smart Form is added, a format logic (as implementation of BAdI DOC_PERSONALIZE_BCS) and as the optimization parameter the type of document personalization can be specified. In recipient-dependent personalization, the formatting of the document takes place for every recipient, since recipient-dependent data is to be replaced. When personalization is not recipient dependent (an identical text or mail to all recipients), document formatting only takes place once for all recipients.

| Document | | FaxCoverSht | Archiving |
|----------------------|---|-------------|----------------------|
| Form name | SPPFDEMO_BOOK | | [Save] [Undo] [Redo] |
| Format | BOOKSALE | | [Save] [Print] [New] |
| Personalization Type | Recipient-Specific Variable Replacement | | [List] |

When sending a fax, you have the option of sending a fax cover page. The name of the cover page (Smart Form) is entered here. The format is fixed. FAX_COVER_PAGE_BCS must always be entered.

| Document | | FaxCoverSht | Archiving |
|-----------|--------------------|-------------|----------------------|
| Form Name | BCS_FAX_COVER | | [Save] [Undo] [Redo] |
| Format | FAX_COVER_PAGE_BCS | | [Print] |

Optical archiving is possible. The object type and the document type must be added in the same way as with Smart Forms.

| Document | | FaxCoverSht | Archiving |
|---------------|------------------|-------------|--|
| Archive Mode | Send and Archive | | [List] |
| Object Type | PPFDEMO | | |
| Document Type | PPFOUTPUT | | [Save] <input type="checkbox"/> Archive copies |

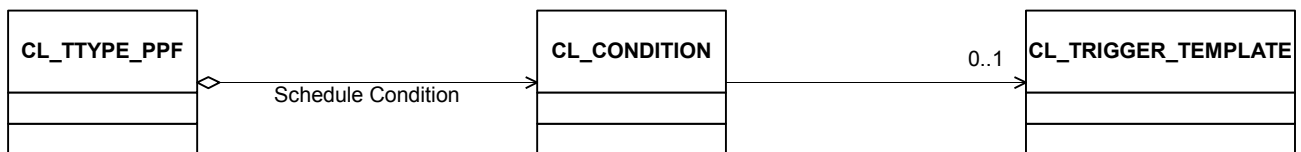
2.3 Determination and Merging of Actions

2.3.1 Determination (Condition Configuration)

The determination returns templates for actions if the previously defined conditions are fulfilled by the application data. The application data is transferred as a BOR object (for determination using conditions that can be transported) or attributes of the action profile class (for determination using conditions). The determination technology defines which conditions must be fulfilled by which application data.

A condition is encapsulated by a rule. You can think of the rule as an application data filter that controls which data is finally released to the condition. A template for an action is appended to the rule. It is returned as soon as the condition is fulfilled (and thus the rule).

The determination technology is appended to each action definition. Thus different action definitions can use different determination techniques. The determination technology itself manages one or more rules.



As standard, a general tool for condition evaluation is used. In Message Control condition technology was used for this purpose. Currently, the PPF offers the workflow condition editor and a self-developed determination technology with generated coding conditions as tools. The determination technology with generated coding conditions should no longer be used. It will not be developed any further. Instead, the workflow conditions should be used. The workflow conditions can also be transported, but the coding conditions cannot.

In release 6.30 there is a new condition logic available. These conditions are similar to the generated coding conditions, however they are realized by BAdI implementations and thus they are also transportable.

2.3.1.1 Workflow Condition Editor (Determination Using Conditions that Can Be Transported)

For Basis Release 6.10, the workflow condition editor replaces the old PPF determination technology. The condition editor provides a graphical user interface and a transport connection. Thus, the applications can preconfigure conditions (for example, document complete, example configurations) and deliver them to the customers.

To ensure good performance, there should be generated workflow conditions. The user interface is similar to PPF condition evaluation. The condition tool is simply a different one.

Conditions for Actions: Change

Scheduling of Actions

- Action Profile
 - Book context
 - Profile with Extended Co 1

Book context

| OK | ActionDef. | N... | Processing Type | Processing | Planning Condition | Start Condition | Stop |
|----|------------|------|------------------------|-------------------------------|-----------------------|-----------------|------|
| | Book sales | 1 | Mail | Mail Output | | | |
| | Book sales | 2 | External Communication | Standard Communication Method | | | |
| | File card | 1 | Mail | Mail Output | | | |
| | Book Order | 1 | Method call | Create New Book | Book Is in Poor Co... | | |

Overview | **ActionDetails** | **ScheduleCondtn** | **StartCondition**

Schedule

Planning Condition: No Condition (Seen as Fulfilled)

☒ ScheduleAutomatically
☐ In the Worklist

Action Summarization: Maximum 1 Successful Processed Action

Time of Processing

Start Condition: No Condition (Seen as Fulfilled)

Time of Processing: Processing using selection report

☒ Default Settings frm Action Definition

Assigned Processings

Smart Forms Mail

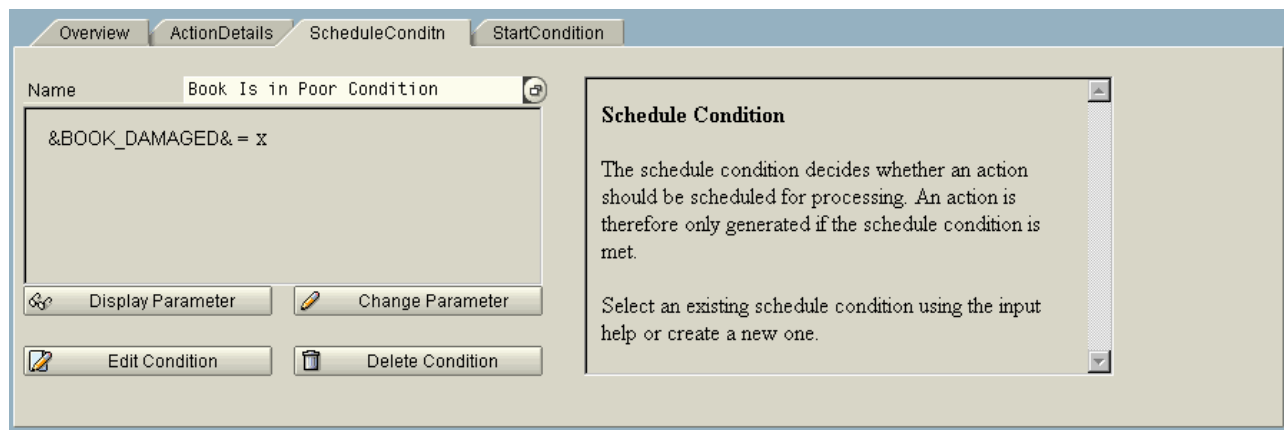
Partner Determination

Partner Function: MP
 Description: MailPartner
 Partner No.:

AEC (1) (000) | pdf0313 | INS

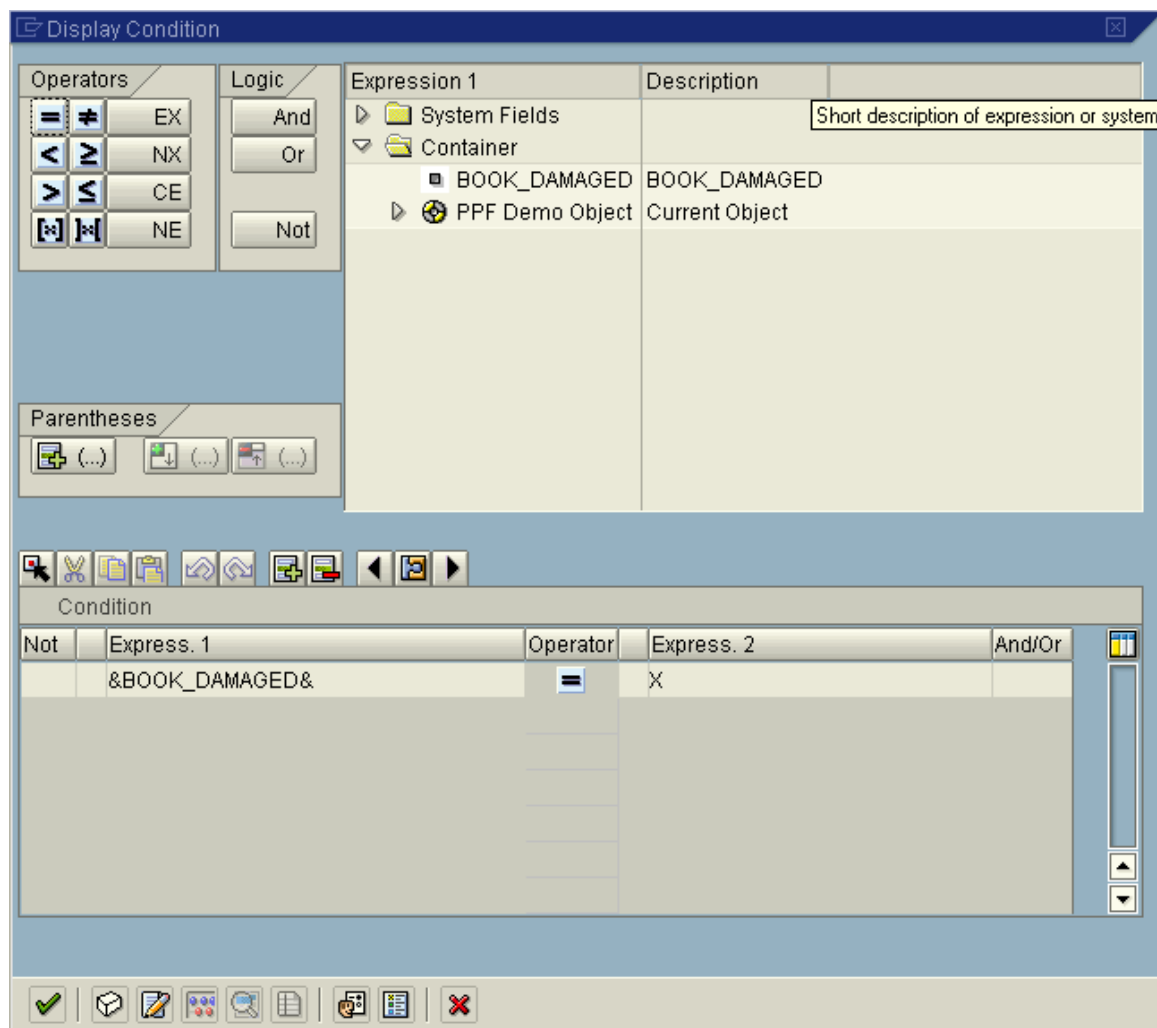
The conditions and settings can be transported and can thus be delivered.

A condition can look as follows:

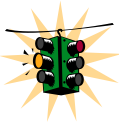


The parameter BOOK_DAMAGED is checked. If the book is damaged, the condition is fulfilled and the relevant action is scheduled.

The user interface for merging the conditions is the workflow condition editor. Here the conditions are simply brought together using the mouse. All attributes of the business object (here PPF demo object) and further parameters (here: BOOK_DAMAGED) are available.



2.3.1.2 PPF Determination Technology (Determination Using Conditions)



This condition logic should not be used any more. Use either the workflow conditions or for special scenarios the Badl conditions.

The determination technology developed by the PPF looks as follows:

Definition von Ausgabebedingungen

Bedingungen für Ausgabebetypen

- Kontext
 - Bücherkontext
 - Buchverkauf (3)
 - Bucheinkauf (1)

Condition profile and action definitions

Bedingungsdefinition

Status = D

```

* T = condition true
* F = condition false

IF 1_context->STATUS = 'D'.
  RULE_EVALUATION_OUTCOME = 'T'.
ENDIF.
  
```

Definition of the condition as ABAP Coding

Ausgabedefinition

Verarbeitungszeitpunkt: sofortige Verarbeitung

Ausgabemedien zuordnen:

- Medium
- Smart Forms Druck
- Smart Forms Druck

Partnerrolle: MP

Partnernummer:

Empfänger | Spooleinstellungen | Formular | Weitere Einstellungen

Ausgabegerät: P775

Ablagemodus: Nur Drucken

Result, if the condition is fulfilled

Conditions for the action definition

| Nr. Bedingung | Ausgabe | Stop | Ok |
|---------------------|-------------|------|----|
| 1 Status = D | Druck/Druck | STOP | + |
| 2 Creator = 'MARKR' | Druck/Druck | | + |
| 3 Pagecount > 250 | Druck/Druck | | + |

The action profile of the application and the relevant action definitions are displayed in the top left-hand screen area above. The conditions that were assigned to the action definitions appear right of this in the list. The overview displays a number, the description of the condition, the assigned action (here: 2 expressions), the stop flag, and a check symbol that displays whether the condition definition is consistent. The stop flag means that once the condition has been met, further conditions are ignored.

There are 2 types of conditions: Conditions with an action and process conditions. Conditions with an action are always assigned one action as a result. The process conditions do not have action templates. They have a controlling character. The other conditions are only evaluated if the process condition returns FALSE as the return value.

An action template (bottom right) can be assigned multiple processing options. If an invoice is to be sent by post, for example and also printed out, simply assign the mail and fax as the processing here and perform

the corresponding settings. If the condition is fulfilled, an action template with 2 processing options is returned. The runtime system creates 2 separate actions from this.

The current conditions are created by an administrator in the production system since they are generally based on application data that is only known in the productive system. A transport of conditions and thus a preconfiguration provided by SAP is not possible with this tool.

2.3.2 Action Merging

Merging is always necessary if existing, unprocessed actions have to be mixed with newly found actions. This is the case, for example, if an existing document is changed. Since all data has changed, actions, that were found when creating the document can be omitted, or new ones can be found. A new logic is required for the merging of new and old actions.

The PPF currently provides three standard logics.

2.3.2.1 One unprocessed action only

This logic only permits one unprocessed action for each action type. The first unprocessed action remains and all other actions are deleted. If the action is processed and another unprocessed action is found, then this one remains.

2.3.2.2 One unprocessed action only per processing type

This logic only permits one unprocessed action for each processing type, that is one print, fax, or mail action. Therefore, there can be multiple actions, but only one per processing type. The first unprocessed action of the processing type remains, all further actions are deleted.

2.3.2.3 Only one action

This logic only permits one action for each action type. The first action remains, all further actions are deleted.

2.3.2.4 Configurable action merging

This logic can be configured using a separate user interface. It can be set whether altogether one unprocessed action may be produced or whether one unprocessed action may be produced for each processing type.

In addition, the number of unprocessed actions can be restricted. For example, as shown here, you can set that an action may only be successfully executed three times. Afterwards, the action is no longer scheduled.

The screenshot displays the SAP 'Display View Action Definition: Details' dialog box. The interface includes a menu bar (Table View, Edit, Goto, Selection Criteria, Utilities, System, Help) and a toolbar. The left pane shows the 'Dialog Structure' with 'Action Profile' expanded, containing 'Action Definition' and 'Processing Type'. The main area shows the following details:

- ActionProfile:** BOOK
- Description:** Book context
- Action Definition:** END
- Description:** Book sales

Below these fields are three tabs: 'Action Definition', 'Action Description' (active), and 'Action Summarizatr'. The 'Action Description' tab contains two sections:

- Number of Unprocessed Actions:**
 - ☒ One Unprocessed Action for Each Action Definition
 - ☐ One Unprocessed Action for Each Processing Type
- Number of Processed Actions:**
 - ☐ Allow Any Number of Actions

At the bottom of the 'Action Description' tab, there is a 'Max.' field set to '1' and three radio buttons:

- ☒ Successful Action(s)
- ☐ Action(s) with Errors
- ☐ Total Action(s)

The bottom status bar shows 'AEC (1) (000)', 'pwdf0313', and 'INS'.

3 Interaction Between Application and PPF at Runtime

3.1 Calling the PPF

The following displays all the necessary steps for starting the PPF. The preliminary steps (see section 2) must be completed.

We again use the demo application from the application class SPPF_DEMO (TA SPPFDEMO). What has to be done?

1. Data declarations

DATA:

```
* reference to application/proxy object
  appl_object TYPE REF TO cl_book_ppf,           "application defined class
* reference to context object
  context TYPE REF TO cl_demo_context_ppf,       "application defined class
* reference to partner object
  partner TYPE REF TO cl_book_partner_ppf,       "application defined class
* reference to partner collection
  partner_coll TYPE REF TO cl_partner_coll_ppf,   "PPF defined class
* reference to PPF manager (interface to PPF services)
  manager TYPE REF TO cl_manager_ppf,           "PPF defined class
* determination protocol
  determination_protocol TYPE balloghndl
  book_id TYPE CHAR10.
```

Get an instance of class CL_MANAGER_PPF. This class displays the interface for PPF. All service methods are called by it.

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

Generate the application object and set the key fields so that the application object can be found or generated again. If the application object is a BOR object then its ID is set here. In our case, we set the key fields of table PPFTBOOK so as to be able to access the entry later.

```
* create application object
  CLASS ca_book_ppf DEFINITION LOAD.

  appl_object ?=
    ca_book_ppf=>agent->if_os_factory~create_persistent_by_key(
      i_key    = book_id ).
```

Generate an object of the action profile class

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

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

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

Generate a partner object and append it to the collection. You perform 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 = 'Lieferant Meier'
          ip_zav_addressno = '0000015762'
          ip_zav_persno    = '0000015763'
          ip_zav_addr_type = '3'.

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

* create another partner object
CREATE OBJECT partner
EXPORTING ip_partner_role = 'WE'
          ip_partner_no   = '0987654321'
          ip_partner_text = 'Ship-to party 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 ).
```

The action profile object that encapsulates all information for the PPF. The name of the application and the action profile that were defined in Customizing are transferred. The reference to the application is transferred and also the partner collection. In addition, the application-specific fields are set for the action profile that are relevant for determination. Conditions for an action are defined in these fields.

```
* set context attribute
context->applctn = 'B00K'.
context->name     = 'B00K'.
context->appl     = appl_object.
context->partner  = partner_coll.

* additional context fields
context->ID = '1234'.
```

```
context->creator = sy-uname.  
...
```

Determination is started. As a return value, the application receives a determination log that displays how many actions were found and why some actions were found. The determination log 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.
```

COMMIT WORK

The PPF uses the persistence services of Object Services. These services run after a COMMIT WORK, that is, a COMMIT WORK must take place at the end so that the generated actions can be written to the database.

3.2 Processing Actions

There are three fundamental possibilities for processing:

- Immediate processing
- With document posting
- Later, by a batch report
- Manual triggering from the dialog

This processing time is defined in action type Customizing and can be overridden in the configuration of conditions.

3.2.1 Immediate Processing

The action is executed as soon as it is found. Execution takes place during document editing. This can make sense if you want an item in the document to be generated automatically, for example.

3.2.2 Processing with Document Posting

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

Example: After the order is booked, an order confirmation is immediately sent by mail.

3.2.3 Later Processing

In later processing, the application itself must trigger the processing of actions. This generally occurs using a selection report that selects the actions to be processed and triggers their processing. The PPF already offers such a report (RSPFPPROCESS or transaction SPPFP). The report can also be triggered in the background (without user interface).

Example: All faxes are to be sent at night in a batch run.

3.2.4 Manual Triggering of Processing in the Dialog

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

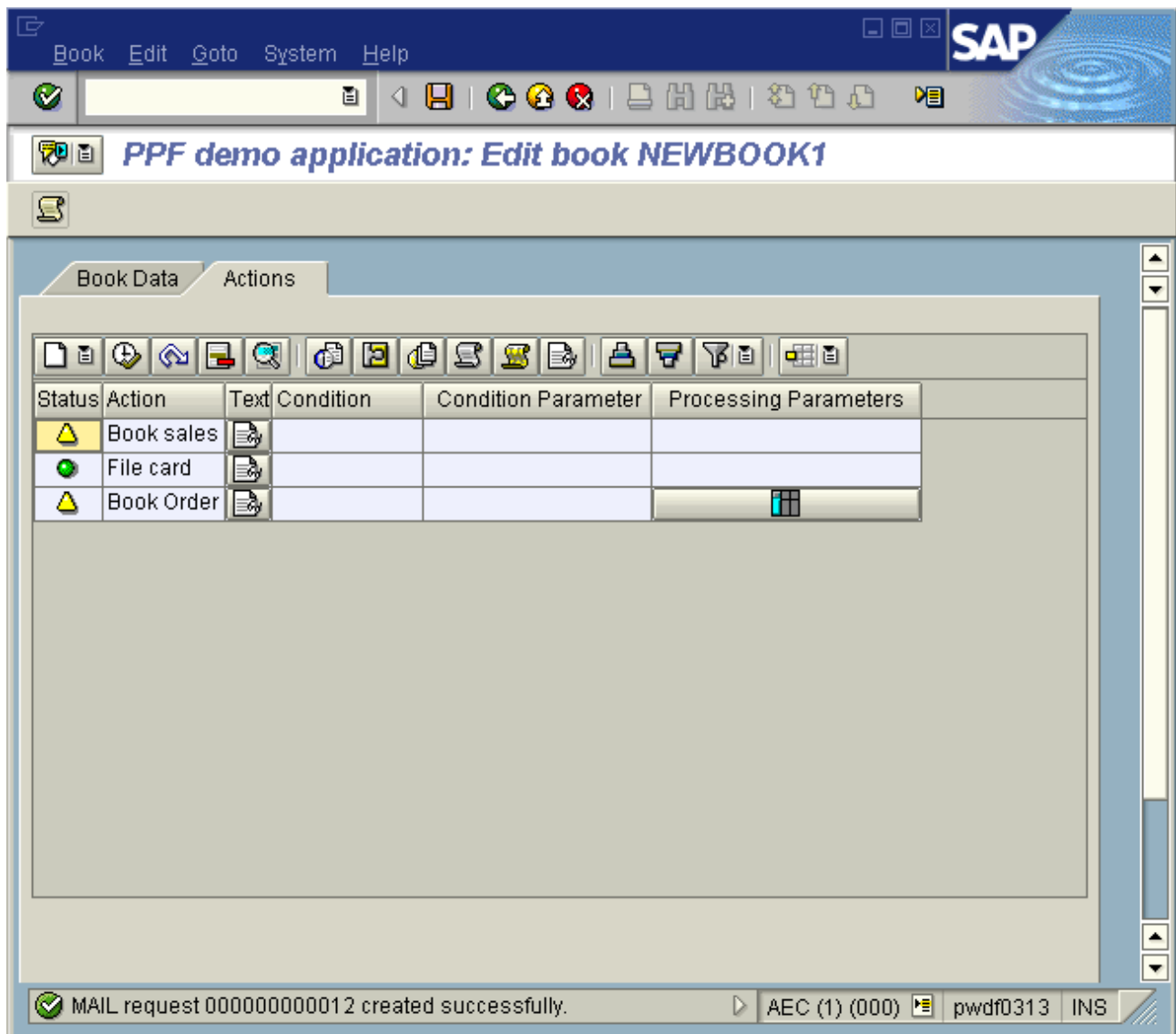
Example: A letter of invitation should be printed during the maintenance of a contact.

3.3 User Interface at Runtime

3.3.1 Standard User Interface

The PPF provides a standardized user interface to display the found and manually added actions. In addition, you can of course create your own user interface for your application. The APIs provide all functions that are needed for this. The necessary function modules and subscreens are available in the function group SPPF_VIEW_CRM. The standard UI is highly configurable, you can exactly determine which columns and functions are to be displayed.

Standard user interface as a subscreen:



The user interface displays the existing actions and their status for a document. Newly found actions are also displayed. A determination log documents in detail which actions have been found this time.

Example of a determination log:

The screenshot shows the SAP 'Display logs' window. The title bar includes 'Log', 'Edit', 'Goto', 'Settings', 'System', and 'Help'. The main window displays a table of log entries with columns: Date/Time/User, Jumb, External ID, Object txt, Sub-object text, and Tran. The log entries are as follows:

| Date/Time/User | Jumb | External ID | Object txt | Sub-object text | Tran |
|-------------------------------|------|-------------|-------------------|--------------------|----------|
| 03.04.2002 14:19:44 MARKR | 32 | | Post-Processin... | Determination I... | SPPFDEMC |
| Problem class medium | 20 | | | | |
| Problem class Additional info | 12 | | | | |

Below the table, there is a list of messages with a 'Ty...' column and a 'Message Text' column. The messages are as follows:

| Ty... | Message Text |
|-------|---|
| 2 | conditions for action END were read |
| 1 | Condition 1 returns an action template |
| 2 | Condition 2 returns an action template |
| 2 | Number of action templates from the determination: 2 |
| | Action is partner dependent, partner determination is performed |
| | A partner was found |
| | Action (MAI) Was Generated and Added |
| | Action is partner dependent, partner determination is performed |
| | A partner was found |
| | Action (BCS) Was Generated and Added |
| | Action Merging: Only 1 Successfully Processed Action(s) Allowed |
| | Action of type END (MAI) is scheduled |
| | Action END of type (BCS) is deleted |
| | Action FILE (File card) |
| 1 | conditions for action FILE were read |
| 1 | Condition 1 returns an action template |
| 1 | Number of action templates from the determination: 1 |
| | Action is partner dependent, partner determination is performed |
| | A partner was found |
| | Action (MAI) was generated and added to the worklist |
| | Action Merging: Only 999.999.999 Successfully Processed Action(s) Allowed |
| | Action of type FILE (MAI) is scheduled |
| | Action BOOK_ORDER (Book Order) |
| 1 | conditions for action BOOK_ORDER were read |
| 1 | Condition 1 returns an action template |
| 1 | Number of action templates from the determination: 1 |
| | Action (MET) Was Generated and Added |
| | Action Merging: Only 1 Successfully Processed Action(s) Allowed |
| | Action of type BOOK_ORDER (MET) is scheduled |

The status bar at the bottom shows 'AEC (1) (000)' and 'pwdf0313 INS'.

Additional functions:

Actions can be added manually

There is a processing log for every processed action

The processing can be repeated (if allowed)

Unprocessed actions can be deleted (if allowed)

Unprocessed actions can be processed (if allowed)

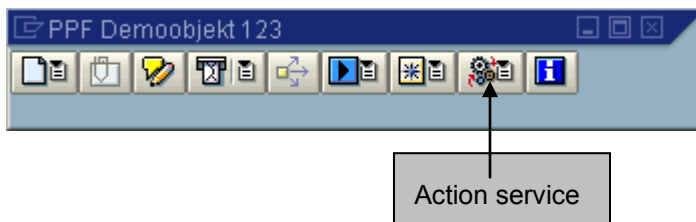
Settings can be overridden (printer, e-mail address, fax number, processing time, action processing, and so on)

A describing text of the action can be displayed

Scheduling of actions from a generated worklist

3.3.2 Connection of Generic Object Services (GOS)

Using this service, actions can also be provided from the worklist for scheduling from here.



A BOR object is needed to connect the GOS toolbar with the action service. The BOR object must implement the BOR Interface IFGOSPPF (see TA SWO1). The interface defines 2 methods that are needed:

GET_MODE read mode (change or display mode)

GET_CONTEXTS read action profile

The service is contacted in Generic Object Services using the name PPFACTION. This name must be transferred for the toolbar constructor. For more information on the Generic Object Services, read the corresponding online documentation.

3.4 Transaction Concept

3.4.1 Overview

Transactions ensure that a certain number of editing steps are either completely executed or not executed at all. When the transaction ends, the transaction is either ended regularly and all data is written to the database or the transaction is rolled back and none of the changes made reach the database.

The PPF essentially supports the transaction concept of the object services. For more documentation on this, see [Object Services](#). The object services do not support any parallel transactions, that is, two objects (documents) cannot be edited in parallel, one is to be canceled and the other is to be saved. This is not covered by the transaction model of object services. Since CRM really needs such a transactions concept, an extension of the existing transaction model, the Object Pool, was implemented in cooperation with the object services.

3.4.2 Object Pool

The Object Pool enables the parallel editing of documents. Here, the Object Pool is always notified which document is currently active. Only one document can be edited for each session. Individual documents can be selected for either saving or canceling (roll back of changes) after editing.

Object Pool methods

SET_GUID (sets the GUID of the document that is currently edited)

All persistent objects that are generated, loaded, or changed from this time on, logically belong to the transaction with the set document GUID. They can later be saved using `save_guids` or reset using `reset_guids`.

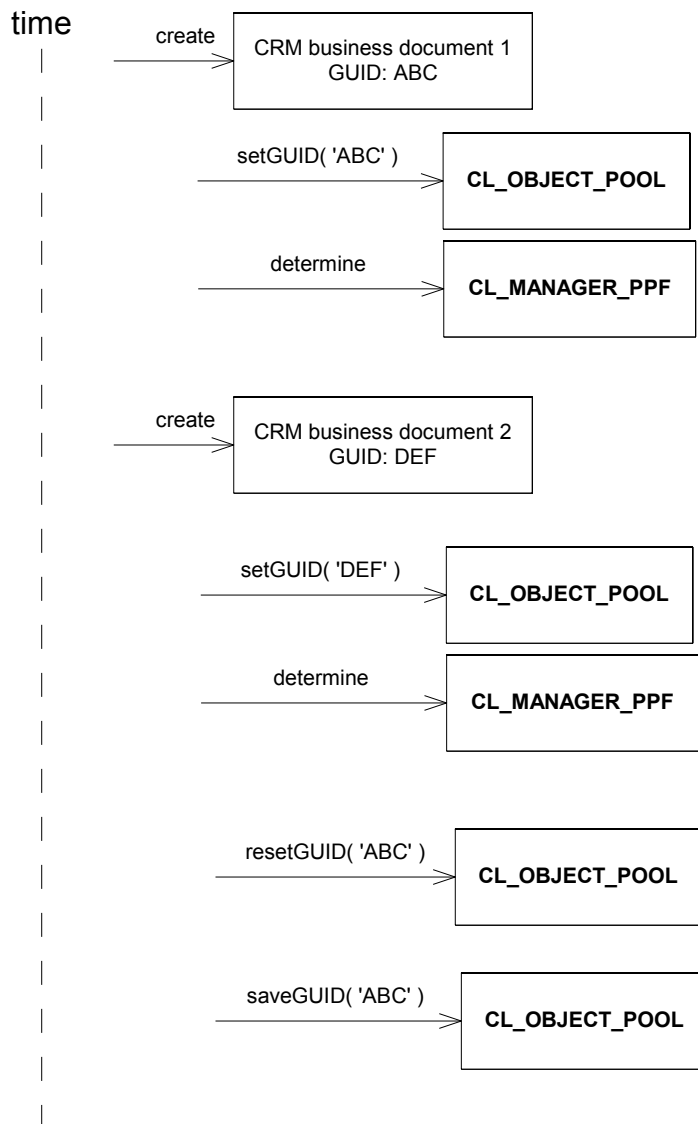
SAVE_GUIDS (transfers the GUIDs of the document that are provided for saving)

All objects that belong to the transferred GUIDs are written to the database.

RESET_GUIDS (transfers the GUIDs of the document that are provided for canceling)

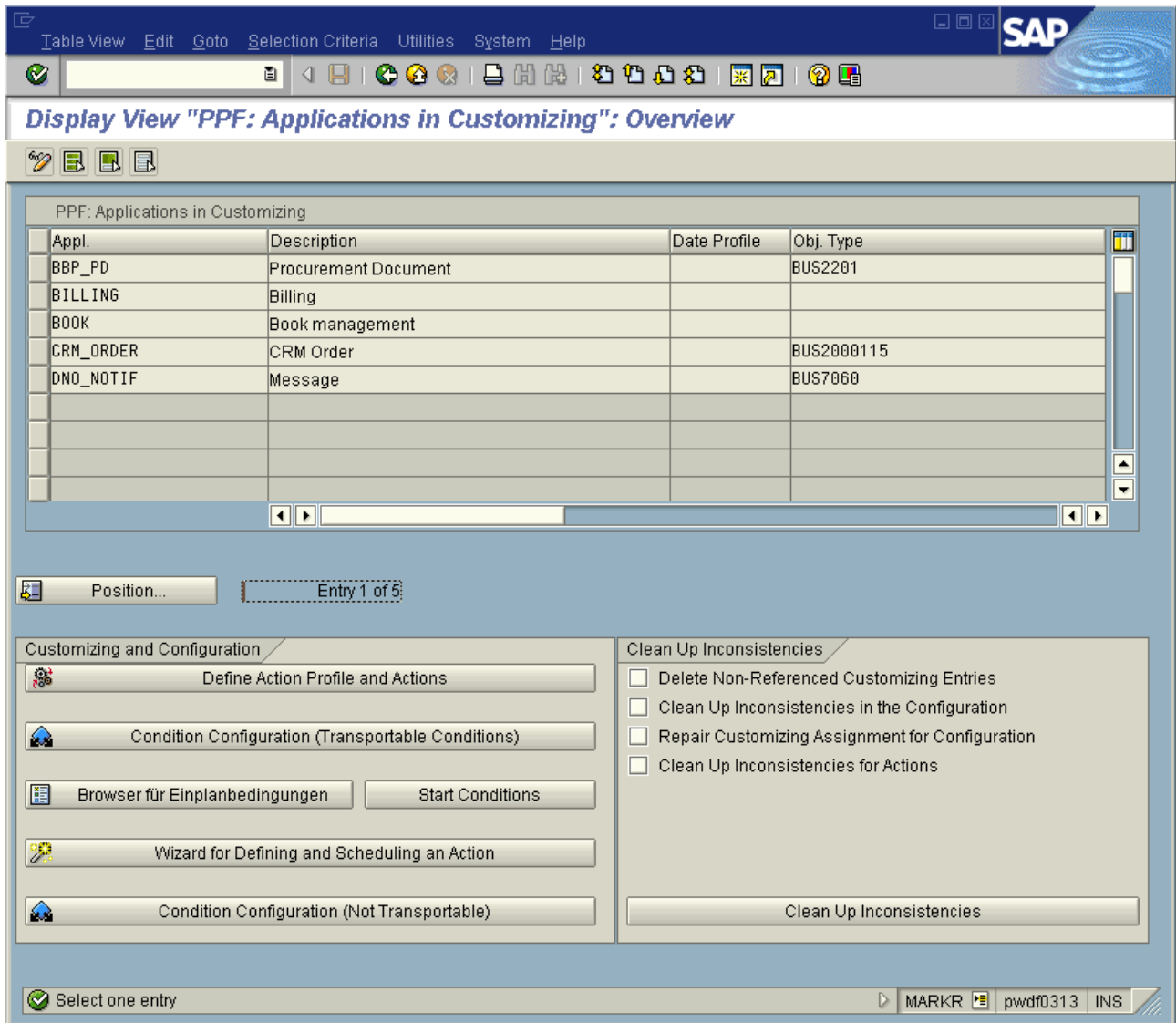
All objects that belong to the transferred GUIDs are rejected.

The following example shows how a call of the object pool can appear. First a document 1 is generated. Here, the GUID of this document is used to start a transaction using `set_guid`. All persistent objects (in the sense of Object Services) that are created from here on are linked to this transaction and can later be saved or rejected. A document 2 can now be created in this mode. During its editing this document GUID must now be set. Like before, all of the objects created are linked to the transaction with this GUID. After the document is edited, the first document is saved here and the second document and the objects (actions) created are rejected.



4 Administration User Interface

Transaction SPPFCADM provides a practical entry point for cross-application administration of PPF.



All applications are displayed and there are links to the action definition, the condition configuration, and the Wizard. You can also correct possible inconsistencies.

5 Extensibility

5.1 Business Application Add Ins (BADIs)

The PPF provides various BADIs that permit manipulation at defined times. There are currently 12 BADIs:

5.1.1 Exit for the printer determination (PRINTER_DETERM_PPF)

| | | | | |
|-----------------|-----------|-------------|----------------|-----------------------------|
| FLT_VAL | Importing | Type | PPFDTT | PPF: Name of a trigger type |
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | PPF: Action profile class |
| CP_PRINTER_DATA | Changing | Type | PPFSPRINT | PPF: Spool data |

The printer is generally returned by the determination with the template for the action. The determination returns an action template with action processing. A printer can be entered in action processing.

It is often the case that a specific printer is to be used due to the application data. In this way, for example, the printer can be determined depending on the sales organization. At definition time of the action template, the sales organization is naturally unknown. For this reason, the following BADI is called provided the template or the determination does not return a printer.

The BADI has the name of the action type as a filter value. As a further import parameter the implementing class has a reference to the actions profile object and thus access to all document data. A printer can now be found and returned using the document data.

Example implementation: PRINTER_DET_SUSR_PPF (transaction SE19)

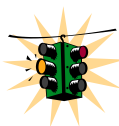
The example implementation reads the master data of the current user and returns the printer maintained there.

5.1.2 Exit After Generated Action (TRIGGER_EXECUTED)

| | | | | |
|------------|-----------|-------------|----------------|------------------------|
| FLT_VAL | Importing | Type | PPFDAPPL | PPF: Application |
| IO_TRIGGER | Importing | Type Ref To | CL_TRIGGER_PPF | PPF: Trigger reference |

This BADI is called after the action is completed, that is, after the action is processed. As a result, the action can perform subsequent actions. If processing is incorrect (that is, there were errors with the processing of the action) a routine or transaction can be executed for error handling, for example.

The BADI has the application names as the filter value. The action is also transferred and can deliver its processing status (successful, with error) or other information.



Take care of the performance. An implementation of this Badi will be called after the execution of any action within the application, as the application serves as filter value here. A better option might be to listen to the executed event of CL_TRIGGER_PPF.

5.1.3 Exit for Context Extension (CONTEXT_EXTEND_PPF)

| | | | | |
|---------------------|-----------|-------------|----------------|------------------------------|
| FLT_VAL | Importing | Type | PPFDTT | PPF: Name of trigger type |
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | PPF: Action profile class |
| RO_EXTENDED_CONTEXT | Returning | Type Ref To | CL_CONTEXT_PPF | PPF: Extended action profile |

The BADI is used for potential extensions of the action profile class by the customers. If the attributes defined in the action profile of the application are not sufficient for the customer's determination, then he or she can extend the application using action profile class using inheritance. The additional attributes must be

provided with values. This BAdI is used for this. The implementations are created for each trigger type. The application should deliver example implementations.

5.1.4 Exit for Completion of Processing Options (COMPLETE_PROC_PPF)

| | | | | |
|----------------|-----------|-------------|-------------------|---------------------------------------|
| FLT_VAL | Importing | Type | PPFSCONACT | PPF: Name of trigger type |
| IO_CONTEXT | Importing | Type | CL_CONTEXT_PPF | PPF: Context Super ClassSmart |
| IP_PROTOCOL | Importing | Type | BALLOGHNDL | Application Log: Log Handle |
| IP_NO_PROTOCOL | Importing | Type | BOOLE_D | Do Not Write a Log |
| IO_MEDIUM | Importing | Type Ref To | Processing Object | Reference to corresponding Processing |

The BAdI is called after the action determination has taken place and before the created action is checked. Here additional data might be set for the processing of the action. The following methods are available depending on the type of processing:

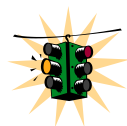
- COMPLETE_MAIL Completes Mail Processing
- COMPLETE_FAX Completes Fax Processing
- COMPLETE_PRINT Completes Print Processing
- COMPLETE_METHOD Completes BAdI Processing
- COMPLETE_WORKFLOW Completes Workflow Processing
- COMPLETE_SEND Completes Transmission Processing
- COMPLETE_ALERT Completes Alert Processing

The interface of the methods is always the same as above described.

5.1.5 Extend Container for Condition Evaluation (CONTAINER_PPF)

| | | | | |
|--------------|-----------|-------------|----------------------|---------------------------|
| FLT_VAL | Importing | Type | OJ_NAME | Name of business object |
| CI_CONTAINER | Changing | Type Ref To | IF_SWJ_PPF_CONTAINER | Container with BOR object |
| CI_PARAMETER | Changing | Type Ref To | IF_SWJ_PPF_CONTAINER | Parameter container |

The BAdI is called whenever a condition for the given BOR object type has to be evaluated. The CI_CONTAINER contains the BOR object and the CI_PARAMETER the uses parameters. Additional objects or parameters might be added to the container for condition evaluation.



An implementation will only work for business object repository (BOR) objects and not for class objects. As the filter is only the business object type, the BAdI will be called whenever this object type is used in a workflow condition. This BAdI should be only used after consulting PPF development.

5.1.6 Exit for Execution of Actions (EXEC_METHODCALL_PPF)

| | | | | |
|--------------------|-----------|-------------|----------------------|---------------------------------|
| FLT_VAL | Importing | Type | OJ_NAME | Name of business object |
| IO_APPL_OBJECT | Importing | Type Ref To | OBJECT | Reference to Application Object |
| IO_PARTNER | Importing | Type Ref To | CL_PARTNER_PPF | Message Partner |
| IP_APPLICATION_LOG | Importing | Type | BALLOGHNDL | Processing Protocol |
| IP_PREVIEW | Importing | Type | CHAR1 | Preview Mode |
| II_CONTAINER | Importing | Type Ref To | IF_SWJ_PPF_CONTAINER | Processing Parameters |
| IP_ACTION | Importing | Type | PPFDTT | Name of Action Definition |
| RP_STATUS | Returning | Type | PPFDSTAT | Processing Status |

The BAdI is a standard exit for enhancing business logic with dynamic processing. Via the implementation PPF can trigger arbitrary business logic. After implementing the BAdI can be assigned in PPF customizing as methodcall processing. Internal development is encouraged to use this processing type for generic processing. Customers may also use this feature for customer specific processing.

**5.1.7 Exit for Getting Possible Partner Functions of an Application
(GET_PARTN_ROLES_PPF)**

5.1.8 Exit for Double Clicking on Values in the Display (GRID_CLICK_PPF)

**5.1.9 Exit for Checking if Deletion of Action Profile is allowed
(CONTEXT_DELETE_PPF)**

5.1.10 Exit for evaluation of schedule conditions (EVAL_SCHEDCOND_PPF)

5.1.11 Exit for evaluation of start conditions (EVAL_STARTCOND_PPF)

5.1.12 Exit for Adding further data to workflow container (WF_CONT_MODIFY_PPF)

5.2 PPF Interface

The PPF provides three interfaces in three places that allow any extension:

5.2.1 Connection of Your Own Processing Options

The PPF provides standard processing options. This is currently Smart Forms Print, Smart Forms Fax, Smart Forms Mail, and generic action processing. In the future, even more processing options like XML or workflow will be supported.

If this is not sufficient to you, you can add your own action media at any time using the action media provided. By implementing the interfaces you automatically integrate the processing options without the need for coding changes inside the PPF.

Further information on this topic is available on request.

5.2.2 Connection of a Logic for the Determination

If the standard determination technologies are insufficient or if any specific determination technologies are necessary, your own determinations can be connected using the IF_DETERMINATION_PPF interface. The new determination class implements the interface and the methods of the interface. Thus the new determination is integrated in the framework and can be used. You can select it using the possible entries in the Customizing of the action definition.

- GET_PERSISTENCY_TABLE

Delivers the table in which persistent data for the determination object is to be stored. The information is needed for the creation of the transport request in Customizing.

- DETERMINE

| | | | | |
|-----------------------|-----------|-------------|---------------------------|--------------------------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile class |
| IP_TTYPE | Importing | Type | PPFDTT | Action type |
| IP_DETLOG | Importing | Type | BALLOGHNDL | Log handle |
| RO_TRIGGER_TEMPL_COLL | Returning | Type Ref To | CL_TRIGGER_TEMPL_COLL_PPF | Collection of output templates |

The actual determination method receives a reference to an actions profile object as the import parameter, the action type, and a handle to a determination log that the condition is to fill. Using this information, the determination returns one or multiple action templates.

5.2.3 Connection to a Separate Logic for Action Merging

Many different logics are also conceivable for the merging of existing and newly found actions. We offer various standard logics. A self-defined logic is very easy to integrate. You must create a class for this that implements the interface IF_MERGE_PPF. As a template, you can view the following classes:

- CL_MERGE_MAX1_FOR_TYPE_PPF max. 1 unprocessed action for each action definition
- CL_MERGE_MAX1_PPF max. 1 unprocessed action

Interface methods:

- GET_PERSISTENCY_TABLE

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

- MERGE

The merge method is called after determination to merge newly found actions with possible existing actions using determination.

| | | | | |
|---------------------|-----------|-------------|---------------------|---------------------|
| IO_NEW_TRIGGER_COLL | Importing | Type Ref To | CL_TRIGGER_COLL_PPF | Newly found actions |
| IO_OLD_TRIGGER_COLL | Importing | Type Ref To | CL_TRIGGER_COLL_PPF | Existing actions |
| IP_DETLOG | Importing | Type | BALLOGHNDL | Log handle |
| RO_TRIGGER_COLL | Returning | Type Ref To | CL_TRIGGER_COLL_PPF | Merged actions |

- **MERGE_SINGLE**

The method **MERGE_SINGLE** is called after the manual generation of an action (see User Interface section) to merge this with possible existing actions.

| | | | | |
|---------------------|-----------|-------------|---------------------|------------------|
| IO_NEW_TRIGGER | Importing | Type Ref To | CL_TRIGGER_PPF | New action |
| IO_OLD_TRIGGER_COLL | Importing | Type Ref To | CL_TRIGGER_COLL_PPF | Existing actions |
| RO_TRIGGER_COLL | Returning | Type Ref To | CL_TRIGGER_COLL_PPF | Mixed proposals |

6 Appendix

6.1 Description of Interfaces

6.1.1 CL_MANAGER_PPF

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

- GET_INSTANCE (Static Method)

Parameter: None

Effect: Delivers an instance of the class (singleton)

- ADD_TRIGGER (Instance Method)

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

| | | | | |
|------------|----------|-------------|----------------|--------|
| CO_TRIGGER | Changing | Type Ref To | CL_TRIGGER_PPF | Action |
|------------|----------|-------------|----------------|--------|

Effect:

Adds a new action for the action profile supplied

- REPEAT_TRIGGER (Instance Method)

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

| | | | | |
|------------|-----------|-------------|----------------|-----------------------|
| IO_TRIGGER | Importing | Type Ref To | CL_TRIGGER_PPF | Action to be repeated |
|------------|-----------|-------------|----------------|-----------------------|

| | | | | |
|------------|-----------|-------------|----------------|-----------------|
| RO_TRIGGER | Returning | Type Ref To | CL_TRIGGER_PPF | Repeated action |
|------------|-----------|-------------|----------------|-----------------|

Effect:

Repeats processing of the action transferred

- DELETE_TRIGGER (Instance Method)

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

| | | | | |
|------------|-----------|-------------|----------------|--------|
| IO_TRIGGER | Importing | Type Ref To | CL_TRIGGER_PPF | Action |
|------------|-----------|-------------|----------------|--------|

Effect:

Deletes the unprocessed action transferred

- DELETE_ALL_TRIGGERS (Instance Method)

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

| | | | | |
|----------|-----------|------|---------|----------------|
| IO_FORCE | Importing | Type | BOOLE_D | Without checks |
|----------|-----------|------|---------|----------------|

Effect:

Deletes the actions of the transferred action profile. The method should be called when the document is called.

- GET_TTYPES (Instance Method)

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

| | | | | |
|-----------|-----------|-------------|-------------------|------------------------------------|
| RO_TTYPES | Returning | Type Ref To | CL_TTYPE_COLL_PPF | Action types of the action profile |
|-----------|-----------|-------------|-------------------|------------------------------------|

Effect:

Delivers the action types for the action profile transferred

- **GET_TRIGGERS (Instance Method)**

Parameter:

| | | | | |
|-------------|-----------|-------------|---------------------|----------------------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
| RO_TRIGGERS | Returning | Type Ref To | CL_TRIGGER_COLL_PPF | Actions for action profile |

Effect:

Delivers all possible actions for the action profile

- **CREATE_TRIGGER (Instance Method)**

Parameter:

| | | | | |
|---------------|-----------|-------------|----------------|-------------------|
| IP_TTYPE_NAME | Importing | Type | PPFDTT | Action definition |
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
| RO_TRIGGER | Returning | Type Ref To | CL_TRIGGER_PPF | Action |

Effect:

Generates an action of the transferred type for this action profile

- **DETERMINE (Instance Method)**

Parameter:

| | | | | |
|-------------|-----------|-------------|----------------|-------------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
| EP_PROTOCOL | Exporting | Type | BALLOGHNDL | Determination log |

Effect:

Starts the determination for the action profile transferred and returns the determination log

- **SET_APPLKEY (Instance Method)**

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|-----------------|
| IP_APPLKEY | Importing | Type | PPFDAPPKEY | Application key |
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |

Effect:

Sets the field APPLKEY in all actions of the action profile with which the system can later search and sort by

- **REFRESH (Instance Method)**

Parameter: None

Effect: All data of the PPF manager is lost

Resets the PPF manager, allocated memory is released

- **LOCK_TRIGGERS (Instance Method)**

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

Effect:

Locks the triggers of this action profile for processing, recommended for incomplete documents

- **UNLOCK_TRIGGERS (Instance Method)**

Parameter:

| | | | | |
|------------|-----------|-------------|----------------|----------------|
| IO_CONTEXT | Importing | Type Ref To | CL_CONTEXT_PPF | Action profile |
|------------|-----------|-------------|----------------|----------------|

Effect:

Unlocks triggers of this action profile for processing

Events:

| | |
|--------------------|------------------------------------|
| DETERMINATION_DONE | Determination was performed |
| CHANGED | Changes were performed |
| REFRESHED | Manager was reset |
| GRID_CHANGED | Changes in the interface were made |

Attribute:

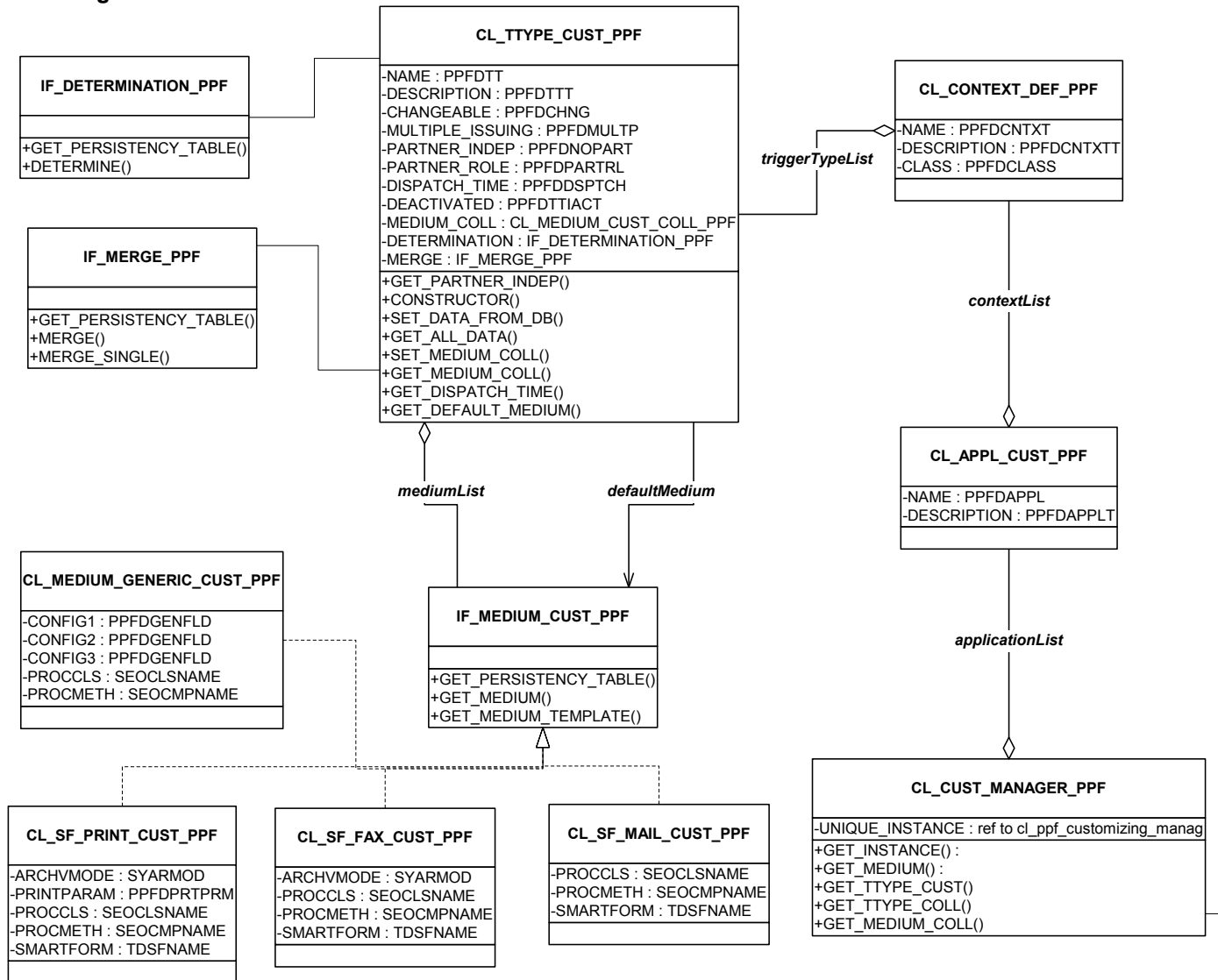
LOCALE_UPDATE Boolean variable default = 'X'

The effect of the attribute is that actions that are to be processed when the document is saved are processed synchronously. If the flag is set to SPACE, processing takes place asynchronously using RFC.

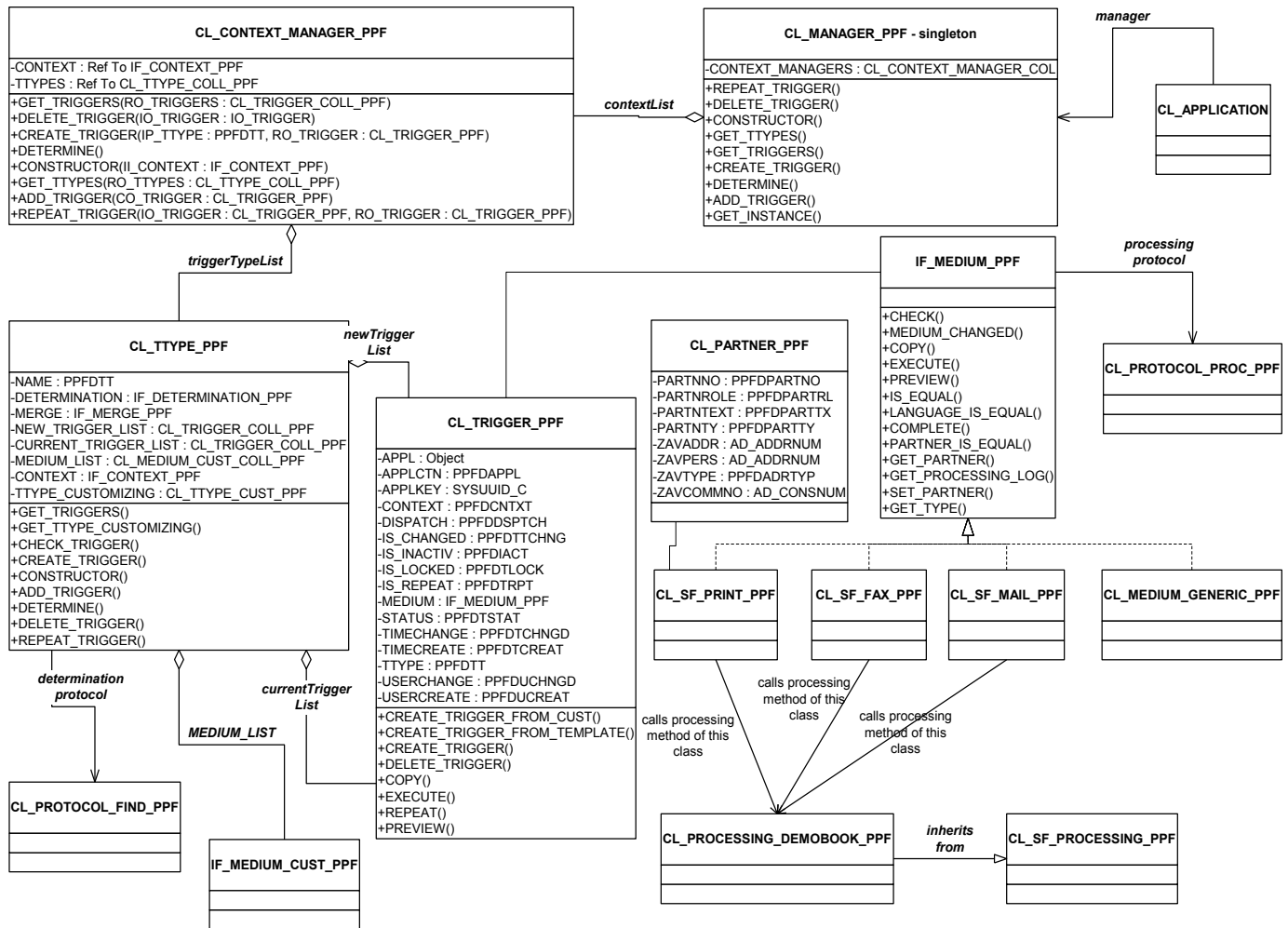
6.2 Class Diagram

6.2.1 Customizing Classes

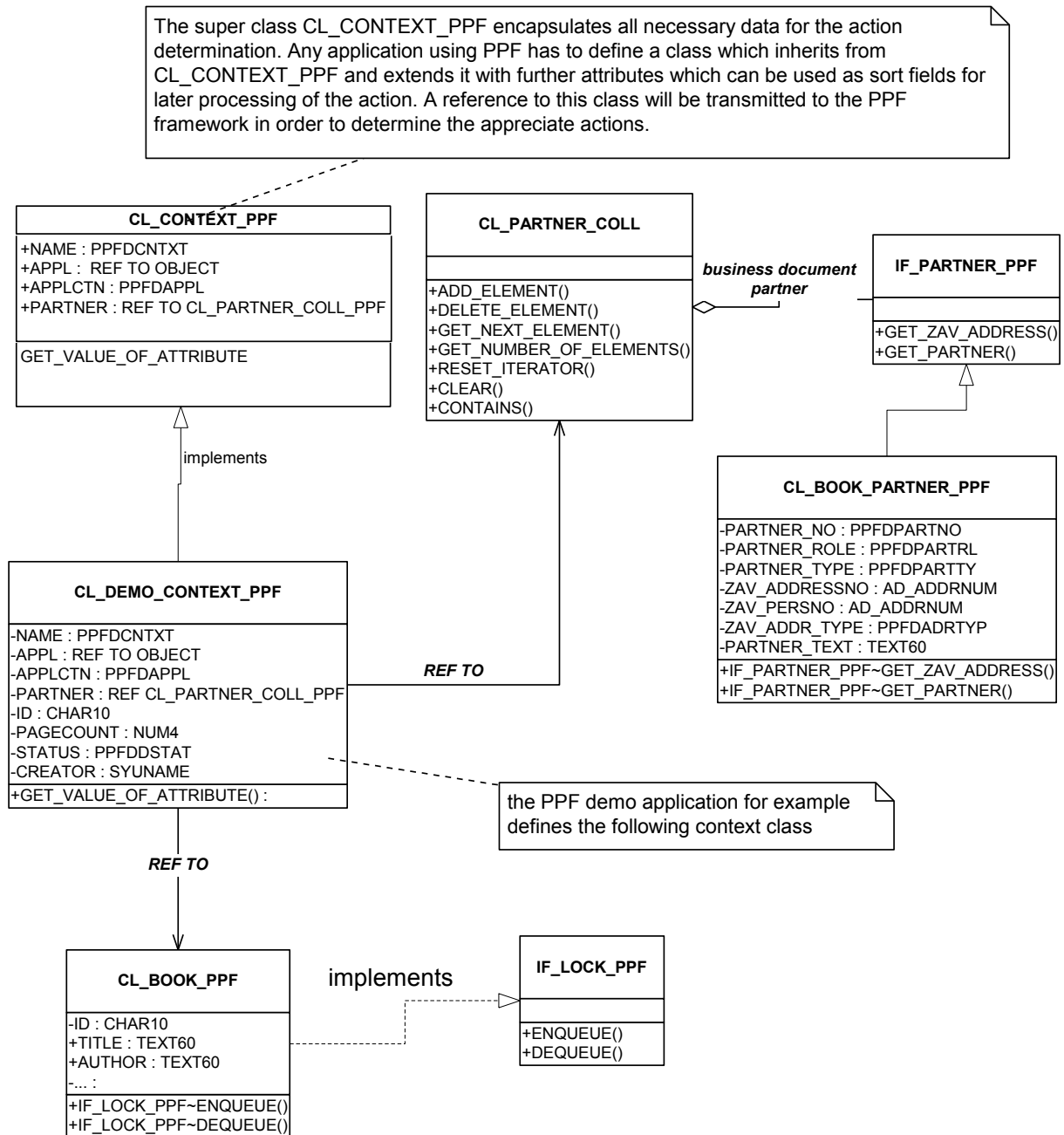
Logische Ansicht



6.2.2 Runtime Classes



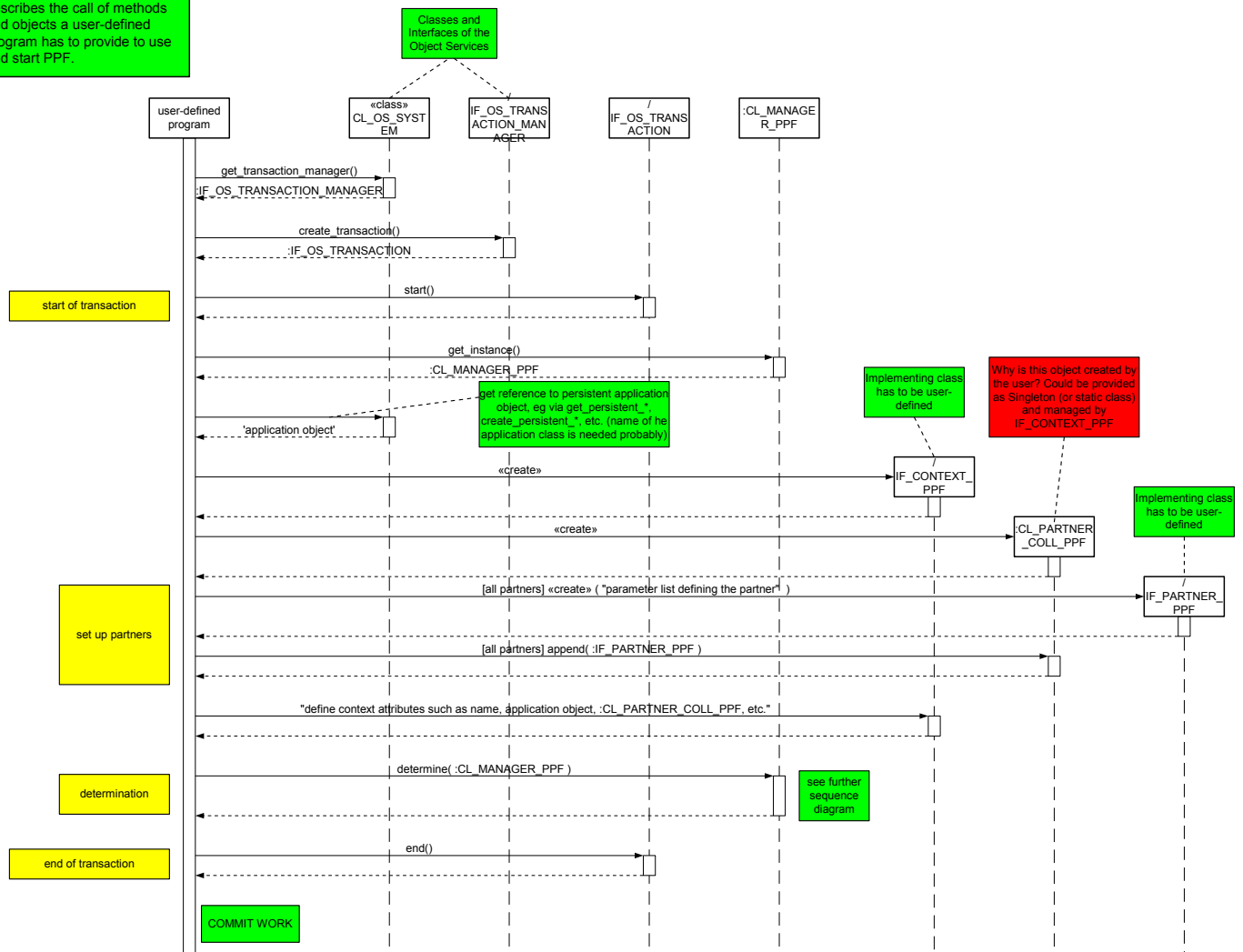
6.2.3 Service Classes



6.3 Sequence Diagrams

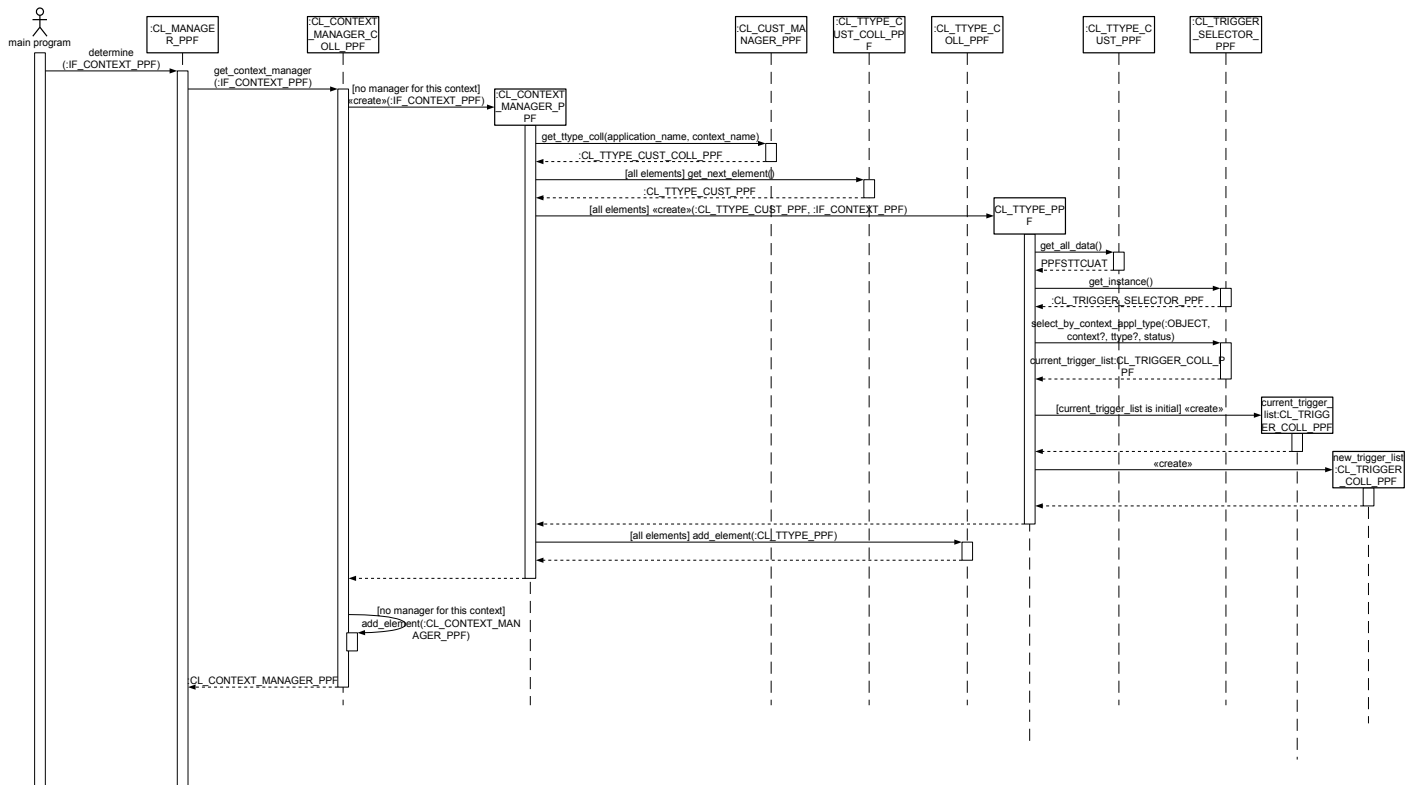
6.3.1 Calling the PPF

This sequence diagram describes the call of methods and objects a user-defined program has to provide to use and start PPF.



6.3.2 Method DETERMINE Part 1

Determination, Scenario A : part 1 - trigger type does not exist yet (AH_990909)



The second part of the DETERMINE scenario just shows the handling after or without explicit trigger creation

6.3.3 Method DETERMINE Part 2

Determination, Scenario A : part 2 - trigger type does exist now (AH_990929)

