

# Developers Guide

29/10/08 15:51:48

[Subscribe](#) to receive email when this article changes.

Search this wiki

RSS

## Table of Contents

### [1 Preliminary form creation tasks](#)

- 1.1 Define Form Statuses
- 1.2 Define Form Actions
- 1.3 Define Form Categories

### [2 Creating a Form Data Schema in the Wizard](#)

- 2.1 Specify Form Type Details
- 2.2 Form Data Structure
- 2.3 Specify Location for XML Data Definition File
- 2.4 Alternate Adobe Designer Templates
- 2.5 Form Options
- 2.6 Summary

### [3 Field User-exits](#)

- 3.1 Field-level prepopulation
- 3.2 F4 Possible entries
- 3.3 Derivation
- 3.4 Substitution
- 3.5 Validation

### [4 Form User-exits](#)

- 4.1 Form-level pre-population
- 4.2 E-mail address derivation
- 4.3 Workflow (FLM Routing Server)
- 4.4 Version
- 4.5 Language
- 4.6 Indexing
- 4.7 Enqueue/Dequeue

### [5 Default Form](#)

- 5.1 Maintain Default Form
- 5.2 Preview Default Form

### [6 Form Template and Package Handling](#)

- 6.1 Exporting and Importing Form Packages

## **7 Form Routing, Configuration and Email Setup**

- 7.1 Form Routing
- 7.2 Form Status Determination
- 7.3 E-mail Reminder Settings
- 7.4 E-mail Settings
- 7.5 Define Approved Email Addresses

## **8 IMG Execution Tools**

- 8.1 System Log
- 8.2 Preview Default Form
- 8.3 ADS Performance testing
- 8.4 List CMS Documents

## **9 Form Posting Engine (FPE)**

- 9.1 Processing Control
- 9.2 Valid FPE status.
- 9.3 FPE function control
- 9.4 Process Functions
- 9.5 Invoking FPE
- 9.6 Posting adapter coding

## **10 FLM Routing Server**

- 10.1 Using the Routing Tables for Form Submission.
- 10.2 FLM Routing Server for triggering Off-line forms.
- 10.3 FLM Routing Server for Form Escalation.
- 10.4 FLM Routing Server for Reminder E-mails.
- 10.5 Portal task instructions

## **11 Correspondence Generation**

- 11.1 Developing a letter template and texts

## **12 Form Structure**

- 12.1 Data hierarchy
- 12.2 Subform definition and binding
- 12.3 Subform look and feel hints and tips.

## **13 Methods for Form Data Handling**

- 13.1 Get the complete address details from an address number
- 13.2 Get the complete address details from a partner number
- 13.3 Get e-mail address from partner number
- 13.4 Get address from address number into single text field
- 13.5 Get standard text into single text field

- 13.6 Prepopulate field within a subform
- 13.7 Add parent paths to form data xml table
- 13.8 Get HR Personnel number from user id
- 13.9 Get User ID from HR Personnel number
- 13.10 Get E-mail address from user id
- 13.11 Get E-mail address from HR Personnel number
- 13.12 Navigate HR organisational structure
- 13.13 Get previous form owner
- 13.14 Get previous form actioner
- 13.15 Get form name
- 13.16 Get form current owner
- 13.17 Get form current status

## **14 [Web Services](#)**

- 14.1 Designer-based web services
- 14.2 FLM-based Web Services

---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)

# 1 Preliminary form creation tasks

07/05/09 10:26:11

[Subscribe](#) to receive email when this article changes.

- 1.1 Define Form Statuses
- 1.2 Define Form Actions
- 1.3 Define Form Categories

For each form you will need to have the required statuses, actions and categories set up. This may have already been done for past form development, but it is worth checking so that you can create extra categories if necessary.

## 1.1 Define Form Statuses

This section displays all the available statuses that can be assigned to each form. A separate status is available according to customer code, description and category.

Form statuses refer to the stage of the workflow, or **routing**, that the form is in at any one time.

Customer	Status	Description	Category
ACL	*	All Statuses	All Statuses
ACL	A	Approved	Intermediate Status
ACL	I	Initial	Initial Status
ACL	P	Posted	Final Status
ACL	R	Rejected	Intermediate Status
ACL	S	Submitted	Intermediate Status
BCC		Approved not posted	Intermediate Status
BCC	*	All Statuses	All Statuses
BCC	A	Approved	Intermediate Status
BCC	I	Initial	Initial Status
BCC	P	Posted	Final Status
BCC	R	Rejected	Intermediate Status

An **initial** status defines a form at the first stage of a routing, i.e. **initial**.

An **intermediate** status defines a form in the middle stages of a routing, e.g. **approved**, **rejected**, **submitted**, or **approved**

A **final** status defines the form at the end of a routing, e.g. **End Of Routing**

**All statuses** displays all available statuses

You can create a new form status tag by going to the 'new entries' menu and entering new criteria for customer code, status code, description, and category. Once this new status has been created in this

activity, it will then become available to select in the routing configuration table (see 1.3.2).

NB. There are 2 statuses which are pre- defined and reserved by the system.

D Draft

L Deleted

## 1.2 Define Form Actions

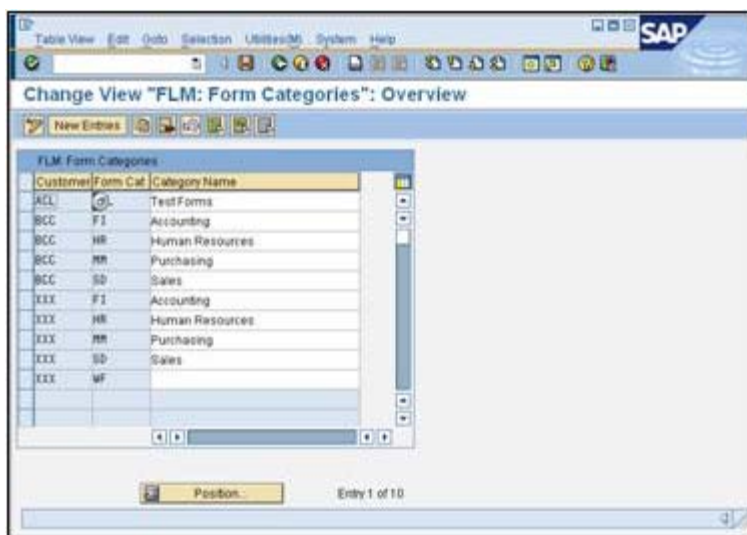
Form Actions list the options available to select the next position in the form routing, e.g. submit, approve, post etc. You can enter a new action by going to 'new entries' and entering a customer code, action code and description for each new action. Actions Y, Z , G, D, L and C pre-defined and reserved by the system. These cannot be reassigned. All other letters of the alphabet are available to assign to different stages of a workflow as you wish. The stage names themselves are only suggestions and are completely customisable.



## 1.3 Define Form Categories

In this activity you define the set of available Form Categories in your system. A Form Category is a logical grouping of Form Types that is used as a part of the user authorisation concept in FLM. You must define your form categories before you can begin creating logical forms via the FLM Form Wizard, as each form must be assigned to a 2-character category.

The groupings shown in the demo below are examples of typical form categories: 'Purchasing' 'Human Resources' and 'Accounting'.



To create a new form category, go to 'New Entries' and define a two-character form category code with an associated description for each form category required. Form Category codes are only applicable for the customer code within which they were set up.

[Table of Contents](#) - Next -> [Creating a Form Data Schema in the Wizard](#)

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 2 Creating a Form Data Schema in the Wizard

04/11/08 16:53:15

[Subscribe](#) to receive email when this article changes.

- 2.1 Specify Form Type Details
- 2.2 Form Data Structure
- 2.3 Specify Location for XML Data Definition File
- 2.4 Alternate Adobe Designer Templates
- 2.5 Form Options
- 2.6 Summary

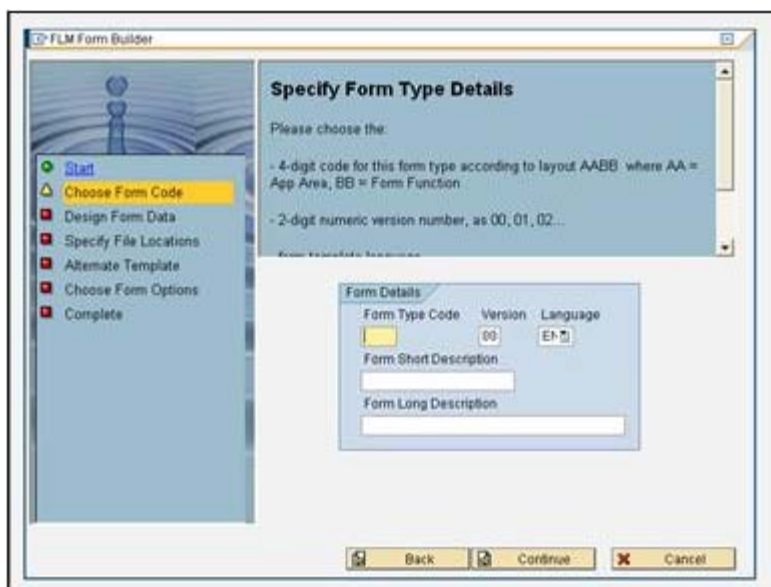
Create the logical definition of a new interactive form, including field names, field types and form templates. This section will take you through an example form creation to explain all the processes involved. The process accesses some



On the welcome page, click 'Continue' to begin creating your form.

## 2.1 Specify Form Type Details

This section of the form wizard defines how the form will be identified within SAP. Each form is defined by a four-character code, version number and language. You can enter a short description and long description of the form here.



### 1. Form type code.

Invent a form type code here: the first two letters define the application area while the second two denote the form's function. For example, a sales order form may have the code SAOR. It is a good idea to decide and agree on a universal form naming system so that each form code accurately describes the form's application area and function. This will be helpful, for example when managing form routings.

### 2. Version.

One of the features of FLM is the support of concurrent form versions. So it is possible to create a form with the same function and 4-character code, but with a different version number. This facilitates forms management, because the 4-digit character code can be fixed for a specified form type and application, and does not need to be altered with successive versions of the form.

Use this field to enter the version number of the form.

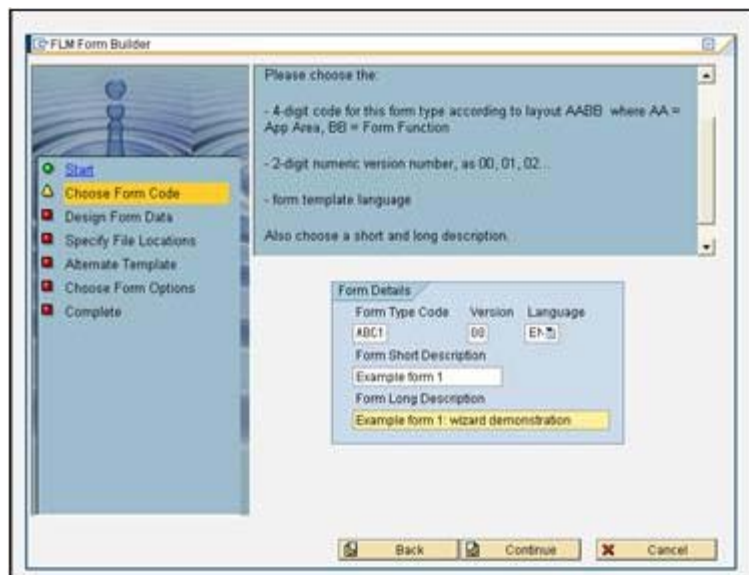
### 3. Language.

Each form must be generated in English before a copy can be made in another language. Once a form has been designed in English though, any language can be selected for a successive version of the form. Note that in a new language version, the data designer cannot be edited and is a read-only version of that created for the English form.

### 4. Form Description

Enter a short and long description of the form to easily identify it in the forms list.





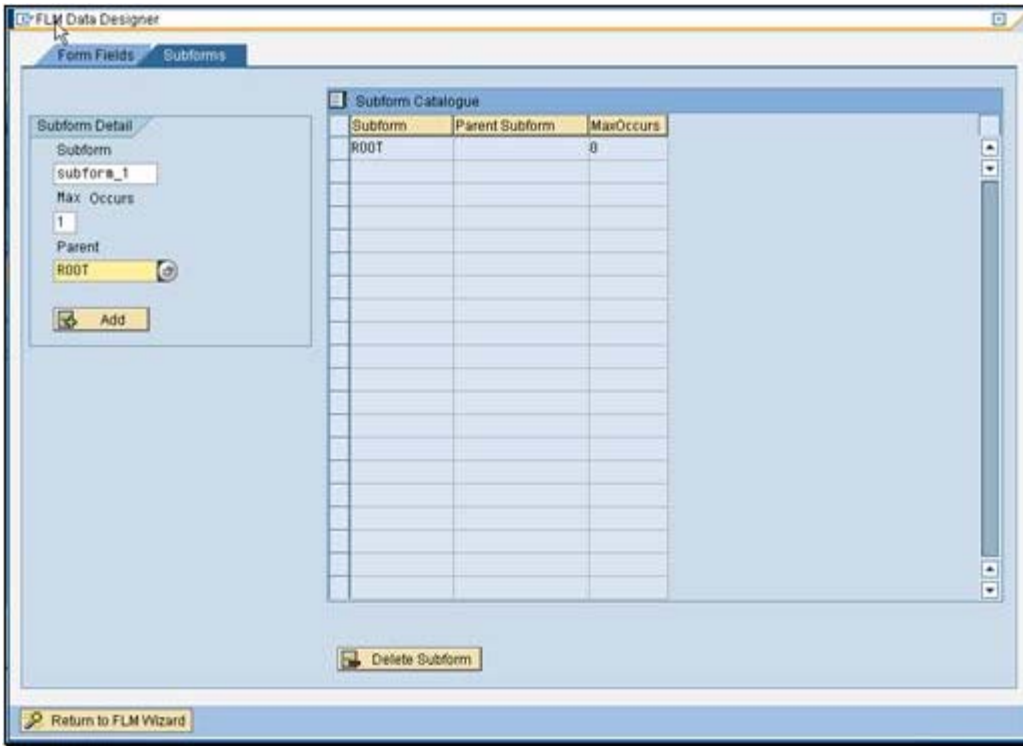
## 2.2 Form Data Structure

Launch form data designer at this stage of the wizard to input the required fields and subforms.

### 2.2.1 Subforms

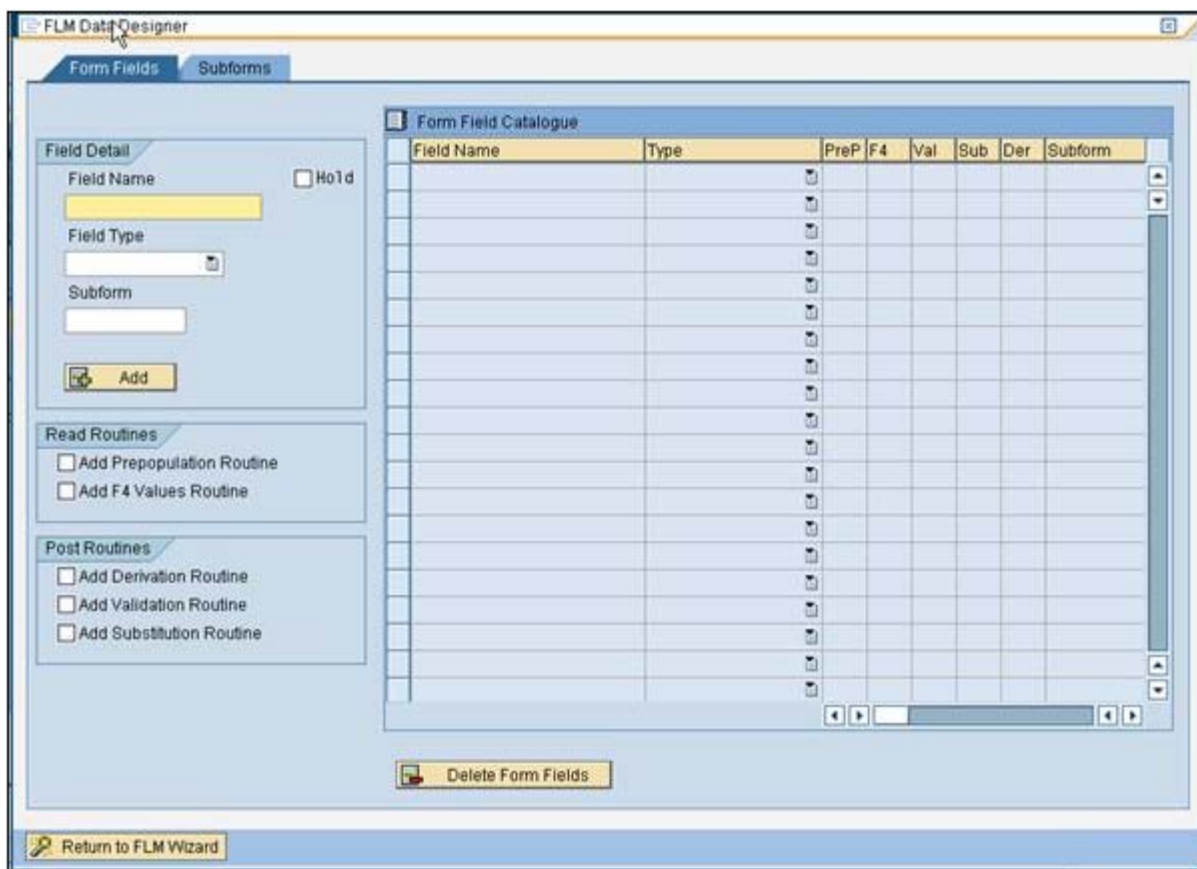
Begin the new form creation by creating a Subform. This is a logical or physical grouping of fields that defines their characteristics on the form. It is a feature of Adobe interactive forms that some fields may be copied according to how many times it is required. For example, on a sales order form, an item description/quantity field may be copied according to the number of discrete items to be ordered.

However it would be undesirable for some fields, for example 'name' and 'address' to be allowed to appear more than once on the form. MaxOccurs allows you to set the maximum number of times a subform field may occur on the form. Following the same example, it would be advisable to set Header fields as MaxOccurs: 1, and Item fields as Maxoccurs: 10. When these fields are eventually created, they can then be assigned to the relevant subform group to maximize the functionality of the fields according to requirements .



### 2.2.2 Fields

To create a field, begin by entering the field name in the 'Field' section of the page. The Field Type can be assigned as character, numeric, date etc. The field can then be assigned to one of the subforms you have created under 'Parent'. Please note that the field name must consist of alphanumeric characters only; underscores are allowed but spaces are not. Though the field names should resemble their content, they need not be the label of the field as it will appear on the form. This is done via form design at a later stage.



### 2.2.2.1 Field Types

CHAR – Character field; will allow free text to be input to the field

NUMC – Numeric field; only numeric characters are allowed.

DATE – Date field

TIME – Only allows times to be input

BOOL – Checkbox; will produce a labelled checkbox which can either be ticked or unticked.

### 2.2.2.2 Read and Post Routines

Read and Post routines can be assigned to each field:

#### Read Routines

Read Routines will apply to the field in any reading instance of the form.

A Prepopulation Routine will cause the field to be prepopulated and will not be amendable by the form user. This could be used, for example, for the Form ID field.

An F4 Values Routine limits the field input to options selected from a drop-down menu. This could be used in the case of there being a limited number of field options to select from, such as Country of Residence.

A **Sticky** dropdown will retain the values populated into it at the first render throughout its routing.

☐ Add F4 Values Routine    ☐ Sticky

If the Sticky function is not enabled, the form will re-render using live data at a later stage of the routing, which may mean that an approver may not have access to the same dropdown data as the initiator who accessed the form, say, a couple of days previously. If live data is always required, DO NOT enable a sticky dropdown.

## Post Routines

Post routines apply to the transfer of data back into SAP via the Posting Engine.

- A Derivation Routine allows the data on the form to be used create a new field before the data is input to SAP.
- A Validation Routine checks the data in the field for validity before the form can be submitted. For example, a user-entered code might be checked against the codes in the database to see whether it is in fact valid.
- A Substitution Routine substitutes the data input to the field for another value. For example, an item option featured as full text in the form might be substituted for an item code as the data is input to SAP.

Please see the section on 'Field-level business logic' for further details.

### 2.2.2.3 Field Editing

To edit or cut a field after it has been entered, first select the action you wish to perform (e.g. update field), then select its row in the Field Name table. Re-click on the desired action to make changes.

Once all the fields have been entered, click on 'Return to FLM Form Wizard' and proceed to the next step.

## 2.3 Specify Location for XML Data Definition File

The form data definitions you have just created will be stored as an XML file during the generation of the FLM interface. Here you can specify the location in which to store that XML file. You must store it to an accessible location such that it can be imported into the Adobe Lifecycle Designer Tool. Click 'Set Directory' and select a new location if the one displayed is not suitable.

## 2.4 Alternate Adobe Designer Templates

Here, you can choose whether to:

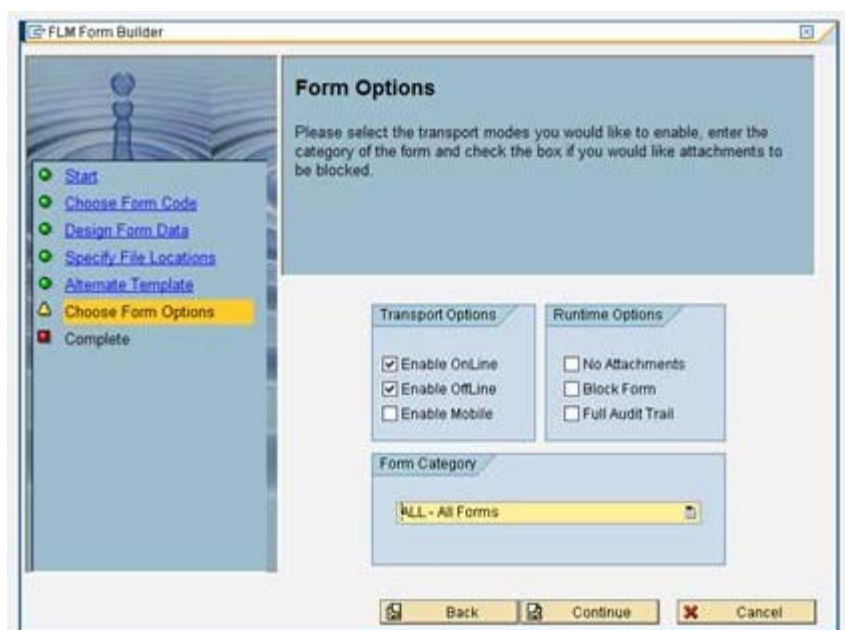
- use the FLM master template, which is uploaded on install
- use a user-specified template, selected as an .xdp file from your local machine

- use a previously uploaded template (useful if you are using the wizard to modify a schema after the template has been configured and re-uploaded into the system)



## 2.5 Form Options

Under Form Options, you can specify the required form category, transport options, and whether an audit trail should be created.



### 2.5.1 Transport Options

With Offline Transport the form can be sent via email as an attachment.

With Online Transport the form is available via the online form portal.

Mobile Transport is not available in FLM version 290.

## 2.5.2 Runtime Options

'No attachments' will disable the ability for form users to add attachments to the form.

Blocking the form will prevent it from appearing in the portal, useful if you would like to prevent others from accessing it in a development environment before it is ready.

An Audit Trail (Variant Trail) can be created as the form is modified through its routing. Before selecting this option however, it is worth noting that an audit trail generates a significantly larger file size attached to each form, which may not be desirable if the form is to be transported offline.

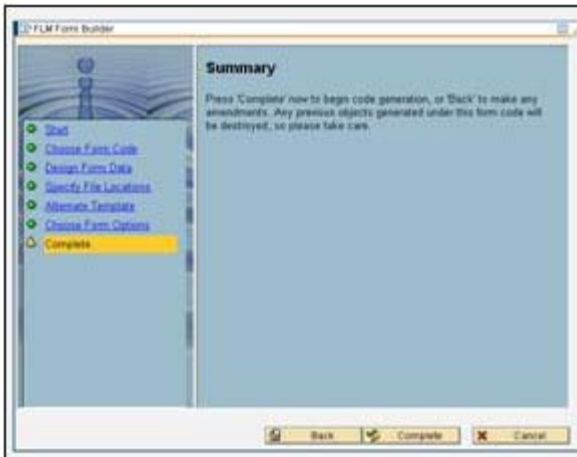
## 2.5.3 Form Category

Here, select the form category under which you would like the form to be stored. You can use different form categories to control user access to forms in the portal, or just to provide logical groupings of forms with a similar function or target area.

## 2.6 Summary

You can now click on 'complete' to generate your form, or 'back' to make any amendments. On clicking 'complete' the components of the form will automatically be saved in the system.

Once the wizard has been closed, you can still make amendments by initiating the form wizard for a form of the same code, and if necessary updating the version number of the form.



[Preliminary form creation tasks](#) <- Previous - [Table of Contents](#) - Next -> [Field User-exits](#)

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 3 Field User-exits

05/11/08 01:20:37

[Subscribe](#) to receive email when this article changes.

---

- 3.1 Field-level prepopulation
- 3.2 F4 Possible entries
- 3.3 Derivation
- 3.4 Substitution
- 3.5 Validation

In order to use field-level User-exits, you must select the type of Userexit required in the wizard. Please see the 'first form' section for details on how to do this.

There are two types of field-level Userexit: Read Routines and Post Routines.

Read routines occur when the form is rendered: prepopulating a field or creating a dropdown based on data from SAP.

Post routines are concerned with manipulating the data on the form before it is posted, i.e. Substitution, Derivation and Validation. These routines are executed when the user clicks 'submit' on the form, and, if a validation fails, the user will be able to re-enter the faulty data.

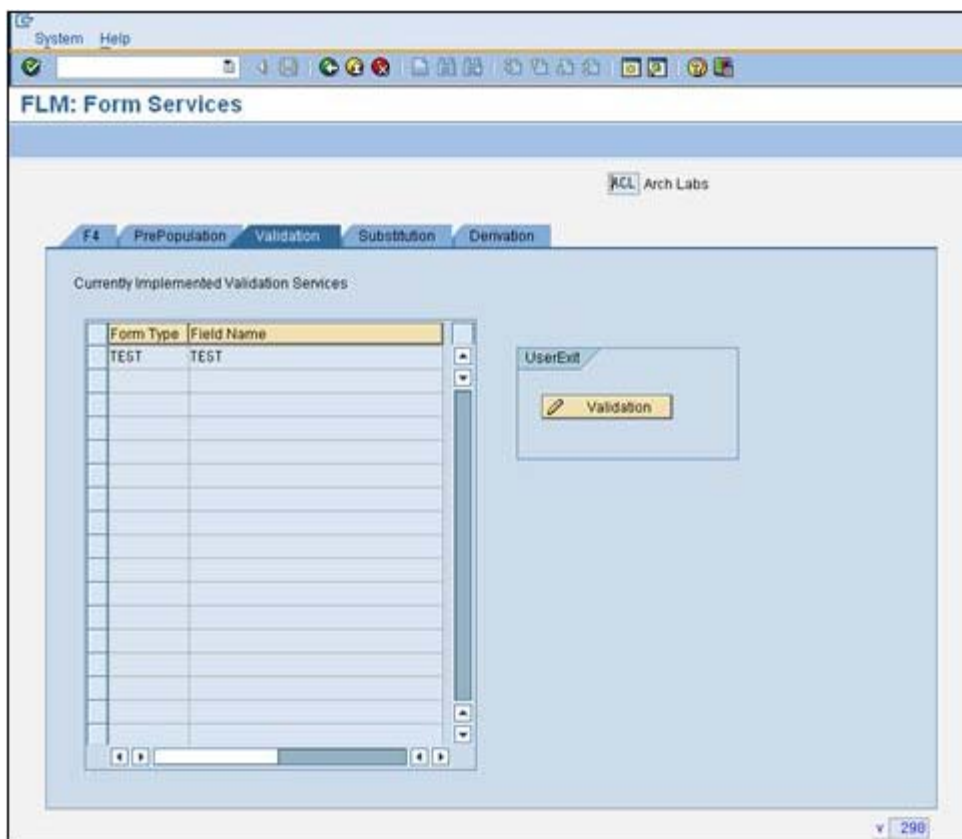
The order in which FLM processes Post routines is:

1. Substitution
2. Derivation
3. Validation

Select the required Userexit type from the tabs, then the required field, and then the 'Userexit' button to access the code editor.

In other words, you can substitute or derive values from user-entered data before the final value is validated.

Once the field has a Userexit assigned to it, go into **Forms Lifecycle Manager> Interactive Forms> Business Logic> Field User-Exits** and enter the relevant Form ID.



Here, you can enter ABAP code to execute the required Userexits. Below are guidelines and hints for creating useful functions.

The following data is available within all field-level user-exits:

**<g\_data>** Internal table of type /FLM/XML\_TAB\_T storing all current form data and one instance of each field.

**<g\_ccode>** 3-character customer code.

**<g\_ftype>** 4-character form type.

**<g\_field>** The name of the currently selected field.

### 3.1 Field-level prepopulation

Prepopulation = defaulting the field's value to a predefined value when the form is rendered, and allowing it to be seen by the user. Often used in conjunction with the Read-Only property. For example, you might want to enter the name of the user based on the login credentials they entered in the portal.

In addition to the core data, the following fields are available:

**<g\_value>** The value of the currently selected field.

**<g\_doc>** 10-character document number if passed in.

This is used for the email form scenario triggered by application document output.



**<g\_user>** The user id.

For email processing this is the user determined by /FLM/CORE->GET\_OFFLINE\_USER (stored on the customer code table.)

For online processing this is the user id passed in by FLM Portal.

The export parameter is ex\_value, which has type 'string'.

## 3.2 F4 Possible entries

Use this function to create a drop-down menu on a field, with values based on live SAP data. For example, you might want to allow an employee to select a current client against which to bill their expenses from the live list held in SAP. The advantage of using this User-exit as opposed to hard-coding the dropdown possible entries on the Designer template is that the values will automatically update as the values in the system change, without having to be edited manually. If the possible entries are not subject to ongoing change though (for example, select 'rare', 'medium', 'well-done', then it is easier to use the dropdown functionality in Adobe Designer when creating the form template.

In addition to the core data, the following fields are available:

**<g\_doc>** 10-character document number if passed in. This is used for the email form scenario triggered by application document output.

**<g\_user>** The user id. For email processing this is the user determined by /FLM/CORE->GET\_OFFLINE\_USER (stored on the customer code table.

For online processing this is the user id passed in by FLM Portal.

The export parameter is ex\_form\_data, which is an internal table with two fields, name and value.

**Note: In FLM version 261, the derived data value needs to be written to the 'name' field and the data description needs to be written to the 'value' field.**

The required syntax is of the form:

```
DATA: l_form_data TYPE /flm/form_data.
MOVE '0' TO l_form_data-name.
MOVE 'OFF' TO l_form_data-value.
APPEND l_form_data TO ex_form_data.
```

## 3.3 Derivation

Use this Userexit to derive a SAP-relevant value based on user-entered data. For example, you might want to post the number of hours an employee has worked to SAP, when they are entering their start and finish times. To do this, you might create an invisible field on the form: 'total hours', whose value would be calculated from the start and finish time fields.

In addition to the core data, the following fields are available:

**<g\_return>** The 'return' field submitted back from the form. This has the structure:

**<ACTION> + <CUSTOMER CODE> - <FORM TYPE> - <LANGUAGE> - <VERSION> - <FORM ID> - <VARIANT> + <RECIPIENT EMAIL ADDRESS>**

**<g\_path>** The path of the currently selected field

**<g\_value>** The value of the currently selected field

Changes to <g\_value> cause the field value to change before posting.

All fields need to already exist on the form – we cannot derive a field in a new instance of a subform through the derivation user-exit.

To read values in the <g\_return> field we need to split the field as follows:

```
SPLIT <g_return> AT '+' INTO l_action l_cms_doc l_rec_email.
```

Then we can split the cms document reference using the method

```
/FLM/CORE-> SPLIT_XDP_CMS_DOC.
```

## 3.4 Substitution

Used when the user-entered value is to be substituted for another, related to it.

In addition to the core data, the following fields are available:

**<g\_return>** The 'return' field submitted back from the form. This has the structure:

**<ACTION> + <CUSTOMER CODE> - <FORM TYPE> - <LANGUAGE> - <VERSION> - <FORM ID> - <VARIANT> + <RECIPIENT EMAIL ADDRESS>**

**<g\_path>** The path of the currently selected field

**<g\_value>** The value of the currently selected field

Changes to <g\_value> cause the field value to change before posting.

## 3.5 Validation

Used to check the validity of user-entered data against SAP back-end data, and allow user to re-enter data if necessary. You might want to use a message box to prompt the user as to the nature of the validation error if it occurs.

In addition to the core data, the following fields are available:

**<g\_return>** The 'return' field submitted back from the form. This has the structure:

<ACTION> + <CUSTOMER CODE> - <FORM TYPE> - <LANGUAGE> - <VERSION> - <FORM ID> - <VARIANT> + <RECIPIENT EMAIL ADDRESS>

<g\_path>     The path of the currently selected field

<g\_value>    The value of the currently selected field

The export parameters are:

- ex\_response       String composed of one or several of the following codes:
  - A     On-Line - Error - reject form
  - B     On-Line - Warning - log event
  - C     Off-Line - Warning - log event
  - D     Off-Line - Error - return form
  - E     Off-Line - Error - delete form
- ex\_mess\_num       Message number from class /FLM/SFS
- ex\_msgvar1        Error variable 1
- ex\_msgvar2        Error variable 2
- ex\_msgvar3        Error variable 3
- ex\_msgvar4        Error variable 4

The syntax should take the following form:

```
if <g_value> is INITIAL.
    ex_response = 'A'.
    ex_mess_num = '999'.
    ex_msgvar1  = 'Initial field not permitted'.
    ex_msgvar2  = <g_field>.
    ex_msgvar3  = space.
    ex_msgvar4  = space.
endif.
```

[Creating a Form Data Schema in the Wizard](#) <- Previous - [Table of Contents](#) - Next -> [Form User-exits](#)

# 4 Form User-exits

05/11/08 01:22:37

[Subscribe](#) to receive email when this article changes.

---

- 4.1 Form-level pre-population
- 4.2 E-mail address derivation
- 4.3 Workflow (FLM Routing Server)
- 4.4 Version
- 4.5 Language
- 4.6 Indexing
- 4.7 Enqueue/Dequeue

## Forms Lifecycle Manager> Interactive Forms> Business Logic> Form User-Exits

Form-level user exits are accessed via transaction /FLM/FORM\_MANAGER.

All user-exits are available to all form types; there is no dependency on settings selected in the New Form Wizard.

### 4.1 Form-level pre-population

The following data is available within pre-population user-exits:

**<g\_data>** Internal table of type /FLM/XML\_TAB\_T storing all current form data and one instance of each field.

**<g\_ccode>** 3-character customer code.

**<g\_ftype>** 4-character form type.

**<g\_doc>** 10-character document number if passed in. This is used for the email form scenario triggered by application document output.

**<g\_user>** The user id.

For email processing this is the user determined by /FLM/CORE->GET\_OFFLINE\_USER (stored on the customer code table.)

For online processing this is the user id passed in by FLM Portal.

The form level pre-population user-exit can be used to prepopulate any field on the form.

It can also be used to create instances of repeating subforms to pre-populate fields within row data – or nested subforms down to 3 levels of nesting.

The return parameter is 'ex\_data', which is an internal table of type /FLM/XML\_TAB\_T. This parameter is set to equal <g\_data> before the user-exit is called.

It is easier to update fields that are in non-repeating subforms with a field user-exit, as in this case there is no need to handle the internal table.

However, if the same table selections or programming logic is required to determine several form fields, then it is easier to use the logic just once, at the form level.

The syntax for updating fields in non-repeating subforms is:

```
READ TABLE ex_data WITH KEY name = 'EBELN' INTO l_data.
IF sy-subrc eq 0.
    MOVE wa_header-ekko-ebeln TO l_data-value.
    MODIFY ex_data INDEX sy-tabix FROM l_data.
ENDIF
```

The syntax for filling fields within a repeating subform called 'ITEM' from an internal table 'itab' is:

```
DATA: l_index TYPE sytabix,
      l_item_subform TYPE string,
      l_data_value TYPE string,
      l_data TYPE /flm/xml_tab.

*
FIELD-SYMBOLS:
<itab> TYPE itab.

*
LOOP AT itab ASSIGNING <itab>.
    l_item_row = sy-tabix.

*
    move <itab>-ebelp to l_data_value.
    CALL METHOD /flm/sfs=>field_prepopulate
        EXPORTING
            im_data      = ex_data
            im_subform   ``= 'ITEM' "Subform name
            im_row       = l_item_row
            im_field     = 'EBELP' "Field name
            im_value     = l_data_value

        IMPORTING
            ex_data      = ex_data.

*
    ...
ENDLOOP.
```

## 4.2 E-mail address derivation

This user-exit returns an internal table of e-mail addresses and is used when an email form is required to be dispatched to multiple recipients.

The following import parameters are available:

- IM\_EMAIL\_STAT-CCODE                      Customer code
- IM\_EMAIL\_STAT-FTYPE                    Form type
- IM\_EMAIL\_STAT-FLANG                    Language
- IM\_EMAIL\_STAT-FVER                     Form version

- IM\_EMAIL\_STAT-STAT\_IN                      Form status
- IM\_EMAIL\_STAT-RECEIV\_ADDR              Receiver's e-mail address
- IM\_EMAIL\_STAT-EMAIL\_TITLE              E-mail title text
- IM\_EMAIL\_STAT-EMAIL\_BODY              E-mail body text
- IM\_EMAIL\_STAT-EMAIL\_ATT\_NAME          E-mail attachment text
- IM\_DOCUMENT                                  Application document number

The export parameter is EX\_EMAIL\_ADDRS which is an internal table with structure /FLM/EMAIL. We can update the following fields only within this structure:

- RECEIV\_ADDR              Receiver's e-mail address
- EMAIL\_TITLE              E-mail title text
- EMAIL\_BODY              E-mail body text
- EMAIL\_ATT\_NAME          E-mail attachment text

In this user-exit we can read the document data (using the document number) to find the partner number and then read the e-mail address from the partner's address details.

The syntax required is:

```
Data:      wa_email      TYPE /flm/email,
          l_smtp_addr    TYPE ad_smtpadr.

call method
/flm/sfs-> GET_PARTNER_ADDR_SMTP

EXPORTING
  im_parvw = im_parvw
  im_parnr = im_parnr
IMPORTING
  ex_smtp_addr    = l_smtp_addr.

wa_email = IM_EMAIL_STAT.

wa_email-RECEIV_ADDR = l_smtp_addr

APPEND wa_email TO ex_email_addrs.
```

**Note: In FLM version 261 we cannot pass in the partner from the NAST record for email forms. Instead we need to start with document data in the user-exit.**

### 4.2.1 Offline form triggered for form distribution list

The email user-exit described above is always used to determine a distribution list

The email form will be triggered by the FLM email submissions utility or by a custom program that calls function module /FLM/OFFLINE\_FORM\_SUBMIT.

### 4.2.2 Offline form triggered by FLM Routing Server

The normal email user-exit is triggered for the determination of e-mail recipients for email forms triggered by FLM Routing configuration or user-exit.

## 4.3 Workflow (FLM Routing Server)

The workflow user-exit can be used to determine the subsequent form owner, version, status and set flags to trigger the sending of an email form or notification e-mail in the case of online forms.

The following import parameters are available:

- im\_action            Action
- im\_instance        Form instance (contains form type, form id etc)
- im\_ftransport       Online/Offline flag
- im\_owner           Form owner
- im\_remind          E-mail notification flag
- im\_status          Form status

The following export parameters are available:

- ex\_owner           New owner
- ex\_ftransport       Online/Offline flag
- ex\_remind          E-mail notification flag
- ex\_status          New status

Typically the logic for determining the new workflow options will be driven by custom tables or by navigating the HR organisational structure.

## 4.4 Version

A new version can be determined prior to form rendering using the version user-exit. This user-exit is only triggered when the form is first created.

The import parameters are:

im\_document

im\_email\_rece

im\_user

The export parameter is ex\_version.

## 4.5 Language

A new language can be determined prior to form rendering using the language user-exit. This user-exit is only triggered when the form is first created.

The import parameters are:

im\_document

im\_email\_rece

im\_user

The export parameter is ex\_lang.

## 4.6 Indexing

Global data is available in this userexit as follows:

- <g\_data> Form data
- <g\_ccode> Customer Code
- <g\_ftype> Form Type
- <g\_doc> Application document reference [form render only]
- <g\_return> Return field [form submit only]
- <g\_user> Logged in user

Update index data in the structure **ex\_findex**

- ex\_findex-ind01 char12
- ex\_findex-ind02 char12
- ex\_findex-ind03 char12
- ex\_findex-ind04 char40
- ex\_findex-ind05 char40
- ex\_findex-ind06 char40

Use this user-exit to write values to six index fields.

These fields are not written to by any other process and are reserved for customer indexing of forms, in order to enable form selection for reporting purposes.

This user exit is called during initial form render AFTER pre-population at each form submission BEFORE form routing.

It is recommended to consider the current action from the FLM RETURN field in order to closely control this customer index update.

## 4.7 Enqueue/Dequeue

Use this user-exit to lock tables and objects when a form is rendered so that it cannot be updated through the lifecycle of the form process. Normally code a matching dequeue user-exit, and call this from the appropriate workflow step or posting adapter.

[Field User-exits](#) <- Previous - [Table of Contents](#) - Next -> [Default Form](#)



---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)

# 5 Default Form

05/11/08 01:24:50

[Subscribe](#) to receive email when this article changes.

---

5.1 Maintain Default Form

5.2 Preview Default Form

This menu compiles a number of functions that allow easy manipulation of the form which is currently set as default.

## 5.1 Maintain Default Form

This contains a list of all forms stored in the system, from which you can set one as default using the tickbox next to its name.

## 5.2 Preview Default Form

In the activity 'Form Types Configuration', one of the options allows you to set a form as default.

In 'preview default form', that form will be displayed as a pdf as it currently appears in the system.

You cannot enter any data into this preview form, nor open it for editing.

[Form User-exits](#) <- Previous - [Table of Contents](#) - Next -> [Form Template and Package Handling](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 6 Form Template and Package Handling

05/11/08 01:37:58

[Subscribe](#) to receive email when this article changes.

---

## 6.1 Exporting and Importing Form Packages

### 6.1 Exporting and Importing Form Packages

FLM packages, i.e. a combination of the form template and data schema (with or without its associated business logic) can be transported between non-productive systems using this facility. To transport a form through to productive systems, you should use SAP transport requests as usual.

#### 6.1.1 Export Form Package

In this activity you can export form definitions to an external file. You can subsequently import these definitions into any FLM system which is of equal or later version. You can export business logic in non-productive systems by clicking the checkbox, but business logic cannot be transported in productive systems.

When exporting to a different system make sure to export using a compatible code page encoding. For example on a Unicode system with a 4103 (UTF-16 Big Endian) and the target system for the export is a Unicode system with 4102 (UTF-16 Little Endian). Leave blank to use the system default. You can check your system code pages in transaction: i18N

To use this activity, select the form you wish to export using the selections at the top of the screen, click execute and select a directory to save the file.

FLM: Form Export

Select Form to Export

Customer Code: ACL

Form Type: ☒

Form Version: 88

Language: E

☒ Include blocked forms

If you wish to export as a different encoding type (for example to allow import to a non-Unicode system) specify the target type here. For the default encoding type to be applied, keep this field blank.

Export Encoding Type:

Business Logic

☐ Also include business logic

Choosing to include business logic in your form package means that all of the user-exit ABAP code and routing behind your forms will also be included. However, you can only import such packages into "Development" systems, and so do not try to use this for transporting forms through your landscape.

The mechanism is provided for importing form definitions from form-sharing websites and also to facilitate remote support.

## 6.1.2 Import Form Package

In this activity you can import a form package into your system. A form package consists of one or more forms in a bundle. During the import you will be given chance to select which forms you wish to import, and also you are given the opportunity to allocate new form types, versions and languages as required.

## 6.1.3 Upload Form Template

This activity is used to upload an Adobe Designer template into FLM. The Master template (Form Code XXXX) should be uploaded at install, and it is used during the form design process to upload a customised template. During the activity, it will check whether there are digital signatures on the template, and ask you to select whether they are client- or server-side. Only one server-side signature is allowed per template.

## 6.1.4 Download Form Template

This activity can be used to download a form-associated template from FLM onto any machine. Select the location of the .xdp file and click save.

[Default Form](#) <- Previous - [Table of Contents](#) - Next -> [Form Routing, Configuration and Email Setup](#)

---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)

# 7 Form Routing, Configuration and Email Setup

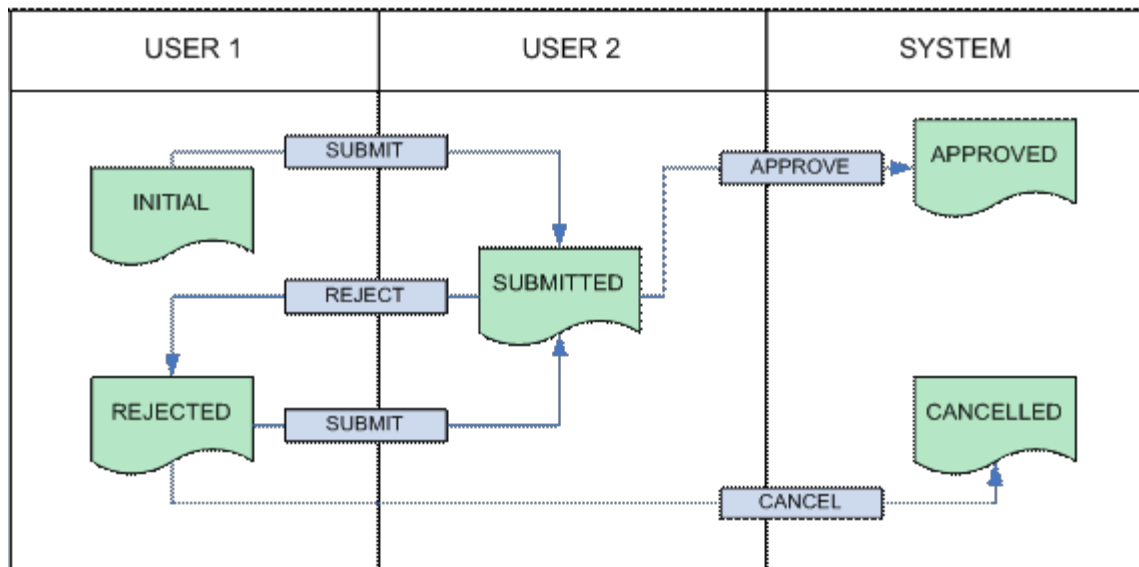
06/03/09 14:29:48

[Subscribe](#) to receive email when this article changes.

- 7.1 Form Routing
- 7.2 Form Status Determination
- 7.3 E-mail Reminder Settings
- 7.4 E-mail Settings
- 7.5 Define Approved Email Addresses
- 7.6 Setting up of standard texts for e-mail notifications

## 7.1 Form Routing

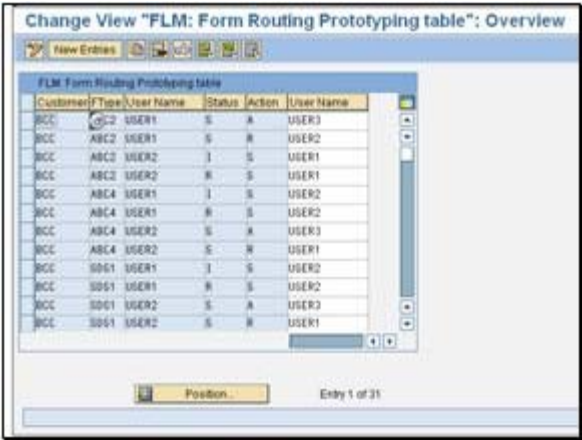
This section allows you to define all aspects of the form workflow, or routing. e.g:



### 7.1.1 Form Owner Determination

In this activity you determine which user will become the new form owner after an action by a specified user on a form of a specified status.

This activity is typically only relevant for workshops and rapid forms prototyping.



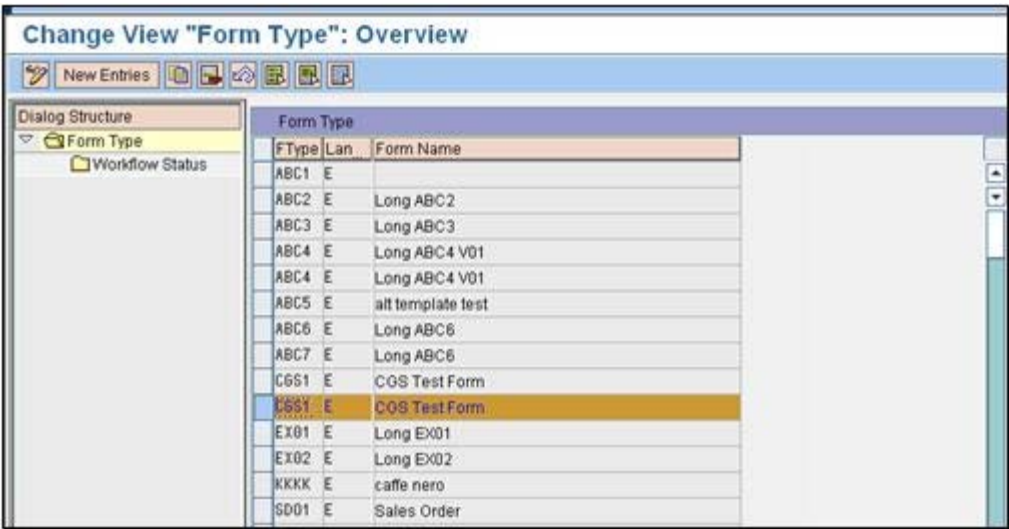
Stages in the form routing are listed as a function of customer code, form type, form status, action and user.

To add a stage in the form routing, go to ‘new entries’ and enter a new sequence of form ownership transfer.

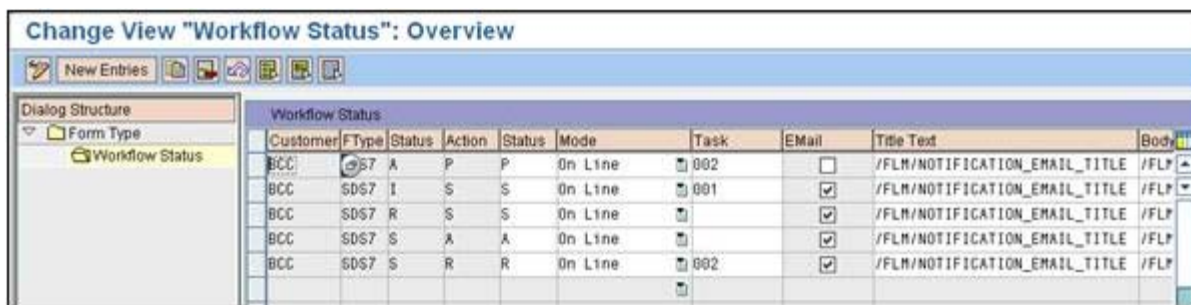
- 1. Enter the customer code
- 2. Enter the form type code
- 3. Enter the first form owner
- 4. Enter the form status code to which the action will be applied
- 5. Enter the desired action code
- 6. Enter the name of the user that will become the owner of the form once steps 1-5 have been applied..

## 7.2 Form Status Determination

This menu allows you to view Form Statuses and Actions for Form Routing.



To view the workflow status of a form, select its name from the list by clicking on the blue bar to the left of the relevant row, then double-click on ‘Workflow Status’ under Dialog Structure. This will bring up a screen detailing the workflow of that form, including whether that stage of the routing is to be conducted online or offline, email settings and links to any email title and body text.



Customer	FType	Status	Action	Status	Mode	Task	Email	Title Text	Body
BCC	SD\$7	A	P	P	On Line	002	<input type="checkbox"/>	/FLM/NOTIFICATION_EMAIL_TITLE	/FLP
BCC	SD\$7	I	S	S	On Line	001	<input checked="" type="checkbox"/>	/FLM/NOTIFICATION_EMAIL_TITLE	/FLP
BCC	SD\$7	R	S	S	On Line		<input checked="" type="checkbox"/>	/FLM/NOTIFICATION_EMAIL_TITLE	/FLP
BCC	SD\$7	S	A	A	On Line		<input checked="" type="checkbox"/>	/FLM/NOTIFICATION_EMAIL_TITLE	/FLP
BCC	SD\$7	S	R	R	On Line	002	<input checked="" type="checkbox"/>	/FLM/NOTIFICATION_EMAIL_TITLE	/FLP

A new status routing can be created by selecting 'New Entries' and entering the Customer Code, Form Code, Version, Status, Action on Status, Status After Action and Online/Offline settings.

To insert a newly created status into a routing, you can click on a status field and the new status will automatically appear in the selection table. The new status configuration will have to be saved using the floppy disc icon.

ABAP code can be entered to define the form owner in the next stage of the routing by accessing data from the SAP database. Please see the Developer Guide for further details.

## 7.3 E-mail Reminder Settings

Email reminders can be automatically sent to the relevant user if the form spends too long at their stage in the form routing. You can set which form statuses have automatic reminders associated with them, the number of days that are allowed to pass before a reminder is sent and whether a reminder will be resent if no action is taken.

This option allows you to create and modify email reminder settings for every individual form in the system, identified by its customer code and form type. Reminders can be configured according to form status; e.g. different reminder settings can be made depending on whether the form is in initial or rejected status. Links can be made to the desired email title text and body text, and an automatic action should reminders be ignored (e.g. rejection) can be set up.



The screenshot shows the SAP 'FLM: Routing Server settings: Overview' table. The table has columns for Customer, Type, Status, Rems, Rem Da, Rese, Title Text, Body Text, Escalate, Esc Days, Action, Posting Ok, and NoHistory. The data is organized into groups for customers ABC2, ABC3, ABC4, and ABC5, each with various status types (A, I, P, R, S, Z). Some rows are highlighted in blue, and some have checkboxes checked in the 'Posting Ok' column. The 'Title Text' and 'Body Text' columns contain references to text objects like /FLN/REM\_IN\_TITLE\_TES and /FLN/REM\_IN\_BODY\_TEST. The 'Escalate' column has a value of 2, and the 'Esc Days' column has a value of 2. The 'Action' column has a value of R. The 'Posting Ok' column has checkboxes checked for some rows. The 'NoHistory' column has checkboxes checked for some rows. The table is displayed in a standard SAP table format with a menu bar at the top and a status bar at the bottom.

Customer	Type	Status	Rems	Rem Da	Rese	Title Text	Body Text	Escalate	Esc Days	Action	Posting Ok	NoHistory
BCC	ABC2	A	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC2	I	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC2	P	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input checked="" type="checkbox"/>	<input type="checkbox"/>
BCC	ABC2	R	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC2	S	<input type="checkbox"/>	1	<input type="checkbox"/>	/FLN/REM_IN_TITLE_TES	/FLN/REM_IN_BODY_TEST	<input type="checkbox"/>	2	R	<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC2	Z	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	A	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	I	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	P	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input checked="" type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	R	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	S	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC3	Z	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	A	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	I	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	P	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input checked="" type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	R	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	S	<input type="checkbox"/>	1	<input type="checkbox"/>	/FLN/REM_IN_TITLE_TES	/FLN/REM_IN_BODY_TEST	<input type="checkbox"/>	2	R	<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC4	Z	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>
BCC	ABC5	A	<input type="checkbox"/>		<input type="checkbox"/>			<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>

To make a new email reminder entry, go to ‘new entry’ at the top of the screen. Here you can enter the 3-letter customer code, four-digit form type, and form status at the time the reminder should be sent.

Under ‘FLM routing server settings’ you can check options to:

**Send reminder email.** The ‘reminder window’ is the number of days allowed after the recipient has received the form for their completion or authorization before a reminder is sent.

**Resend reminder:** here enter the name of the text object used to store the title of the reminder e-mail, and the name of the text object used to store the email body text.

**Escalate form.** If the user has still not passed the form on after all reminders have been sent, the system can automatically take action on the form. You can enter the number of days that make the escalation window, and the required action code, e.g. R.

**Posting OK** Tick this box if posting is required at the given status/stage in the workflow. The actual posting configuration takes place in the Forms Posting Engine, under ‘Define valid FPE statuses’, but this box must also be checked to allow posting to take place.

**No History** Allows you to select whether the form history is viewable by the owner in that status/stage of the workflow. For example, it may be desirable to hide the form history from a user.

New Entries: Details of Added Entries

Customer

Form Type

Status

FLM: Routing Server settings

☐ Send reminder email

Reminder Window

☐ Resend reminder

Title Text

Body Text

☐ Escalate Form

Esc Win [Days]

Action

☐ Posting Ok

☐ NoHistory

7.4 E-mail Settings

This facility allows you to view and create email sender and receiver settings. This defines the email addresses assigned to particular stages in an offline form routing. For example, a form with status I (initial) may be submitted to an approver at the first stage of the form routing. In the example below, an Initial form ABC3 will be routed to [rg@arch.co.uk](mailto:rg@arch.co.uk) once submitted by the initiator.

Change View "FLM: Offline form settings": Overview

New Entries

FLM: Offline form settings								
Customer	Type	Language	Version	Status	Receiver Email	Title Text	Body Text	Attachment Name
BCC	ABC2	E	00	*				
BCC	ABC3	E	00	*				
BCC	ABC3	E	00	I	rg@arch.co.uk	/FLR/OFFLN_TITLE_TEST	/FLR/OFFLN_BODY_TEST	/FLR/OFFLN_ATT
BCC	ABC4	E	00	*				

7.5 Define Approved Email Addresses

This facility allows you to create a list of Safe Email Recipients recognized by the system to be used in form routings in non-productive systems.

7.6 Setting up of standard texts for e-mail notifications

SAP standard texts (maintained in transaction SO10, with a name beginning /FLM/, text ID ‘ST’) hold the text for the e-mail title and e-mail body. Variable substitution is allowed inside these texts as follows:

- &FORMNAME& Description of form
- &SYST-DATUM& Formatted System date
- &SYST-UZEIT& Formatted System time

&<variable from syst>& Any variable from table SYST

&<variable from fpe>& Any variable from table /FLM/FPE

&<form field>& Any form field not a repeating subform

You must schedule report RSCONN01 in order to physically dispatch the emails at the frequency you require.

[Form Template and Package Handling](#) <- Previous - [Table of Contents](#) - Next -> [IMG Execution Tools](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 8 IMG Execution Tools

07/11/08 10:50:38

[Subscribe](#) to receive email when this article changes.

- 8.1 System Log
- 8.2 Preview Default Form
- 8.3 ADS Performance testing
- 8.4 List CMS Documents

## 8.1 System Log

The FLM system log captures various events that occur in the system - the log is a tool supplied to help with the initial system configuration and also to help when problem solving. Among other things, the storage of form attachments (other than the form history) is recorded here



## 8.2 Preview Default Form

This activity will open a preview of the form which is currently set as default within SAPGUI. Useful to check the appearance of a form's template after uploading.

## 8.3 ADS Performance testing

Use this activity to test the speed of form rendering. Go into the activity, enter the number of times you want the form to render (e.g. 10), and then select the form type you would like to test. The program will render the form the number of times you have dictated in succession, and return an average time lapse per render. A one-page form should take in the region of one second to render.

ADS performance is a function of hardware, software and basis settings. The overwhelming majority of processing in ADS rendering is done on the java stack.

Performance improvements fall into 2 main categories, firstly making an individual CPU thread run more quickly, and secondly, providing more parallel CPU threads for ADS processing:

### Individual CPU thread Tuning

Check there is no memory limitation on the machine at the OS layer. Processes to monitor

are "PDFManipulation.exe" and "XMLForm.exe".

Run the ADS system on native 64-bit hardware. Even though some ADS modules are 32-bit, the overall performance can be improved by building on 64-bit hardware. Check SAP note 925741 for details of supported platforms.

Make the form templates as small as possible. Do not embed logos and use non-embedding fonts [TNR, Times, Arial, Helvetica and Courier] only.

## Parallelising ADS Processing

Java stacks can be clustered.

In the Visual Administrator, <SID> / ServerXX / Services / PDF Manipulation module - Low encryption, change the PoolMax setting to allow more CPU cores to run ADS process threads. Obviously this can have a detrimental effect on other services provided by the java stack. Similarly adjust 'XML Form Module' service via the VA. Always restart the java stack after these changes.

Please see SAP's "Sizing for Adobe Document Services" guide for more details on performance tuning. This can be downloaded from OSS

## 8.4 List CMS Documents

This simple report lists form data and also checks that the form data is stored correctly in the CMS database. Make your selections of the form types you would like to view on the initial screen, and click 'execute' to produce the report. It is useful for investigations during system setup, as it links the Forms Posting Engine with the record in the CMS database.

[Form Routing, Configuration and Email Setup](#) <- Previous - [Table of Contents](#) - Next -> [Form Posting Engine \(FPE\)](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 9 Form Posting Engine (FPE)

01/10/09 10:16:35

[Subscribe](#) to receive email when this article changes.

---

- 9.1 Processing Control
- 9.2 Valid FPE status
- 9.3 FPE function control
- 9.4 Process Functions
- 9.5 Invoking FPE
- 9.6 Posting adapter coding
  
- 9.7 Posting adaptor code to save flat version of the form

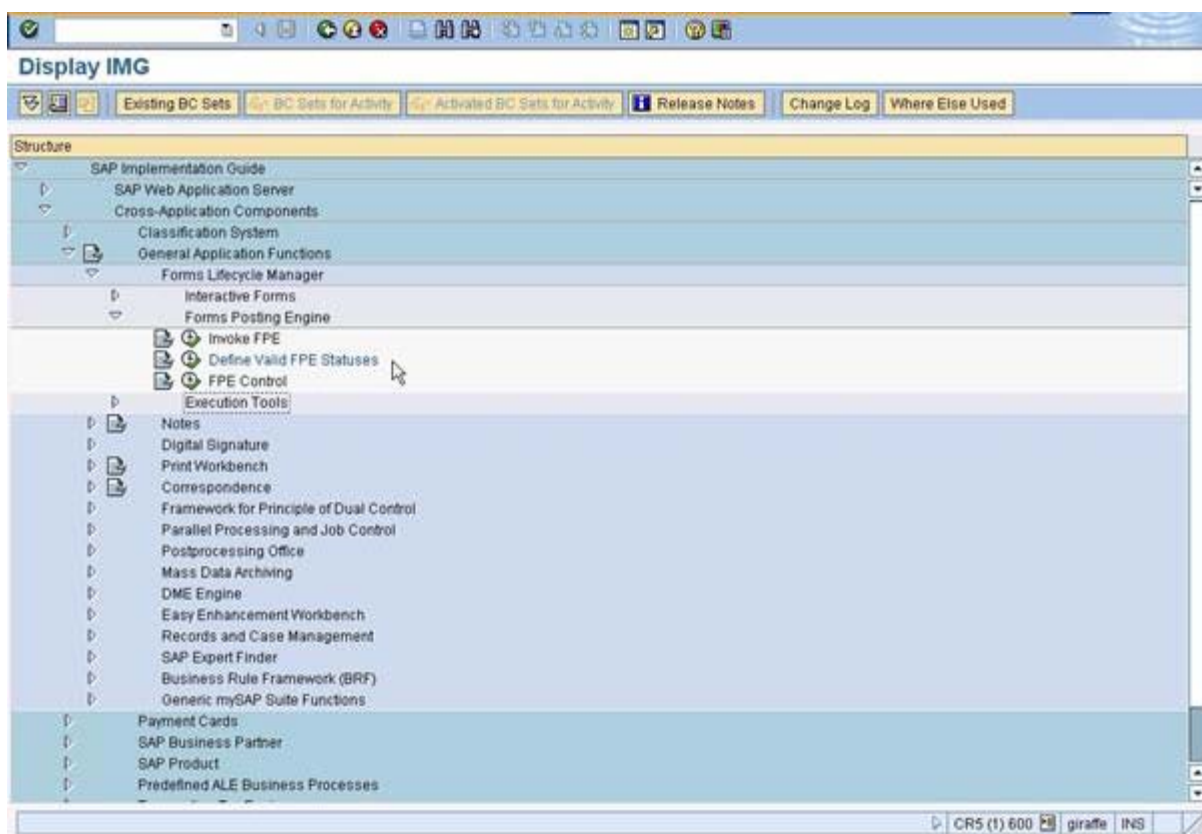
The Forms Posting Engine [FPE] is the component of FLM that controls the posting of data captured on an interactive form into an SAP back end. FPE provides the framework to control when a form is applicable for posting. FPE also provides the facility to handle errors in posting and to repost the data as required once a potential problem has been corrected.

FPE can be run interactively in foreground or scheduled as a batch job to automatically pick up and post forms on a regular basis - but still allowing foreground intervention for problem resolution.

FPE also provides the facility to display the form as it was posted inside the SAP GUI.

## 9.1 Processing Control

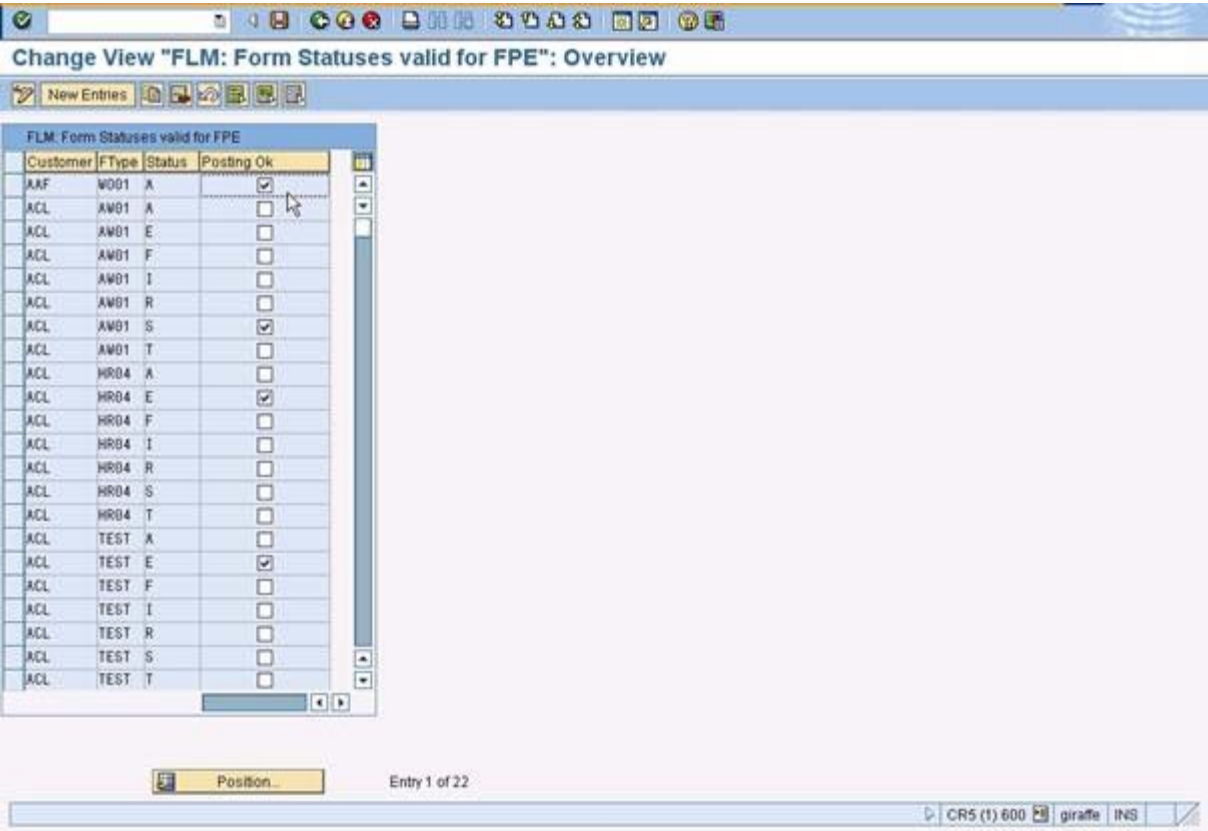
FPE processing is controlled by 2 tables. They can be accessed from menu items inside the FLM section of the IMG as shown:



The first table defines what Forms status is applicable for FPE posting. In other words at what stage in the Forms life do we want to persist some data from it into an SAP back end. The other table controls what function module will be invoked to extract data from the Forms and post it into SAP.

## 9.2 Valid FPE status.

The control table view behind this option is called /FLM/FPE\_STAT\_V. On execution the following screen is displayed:

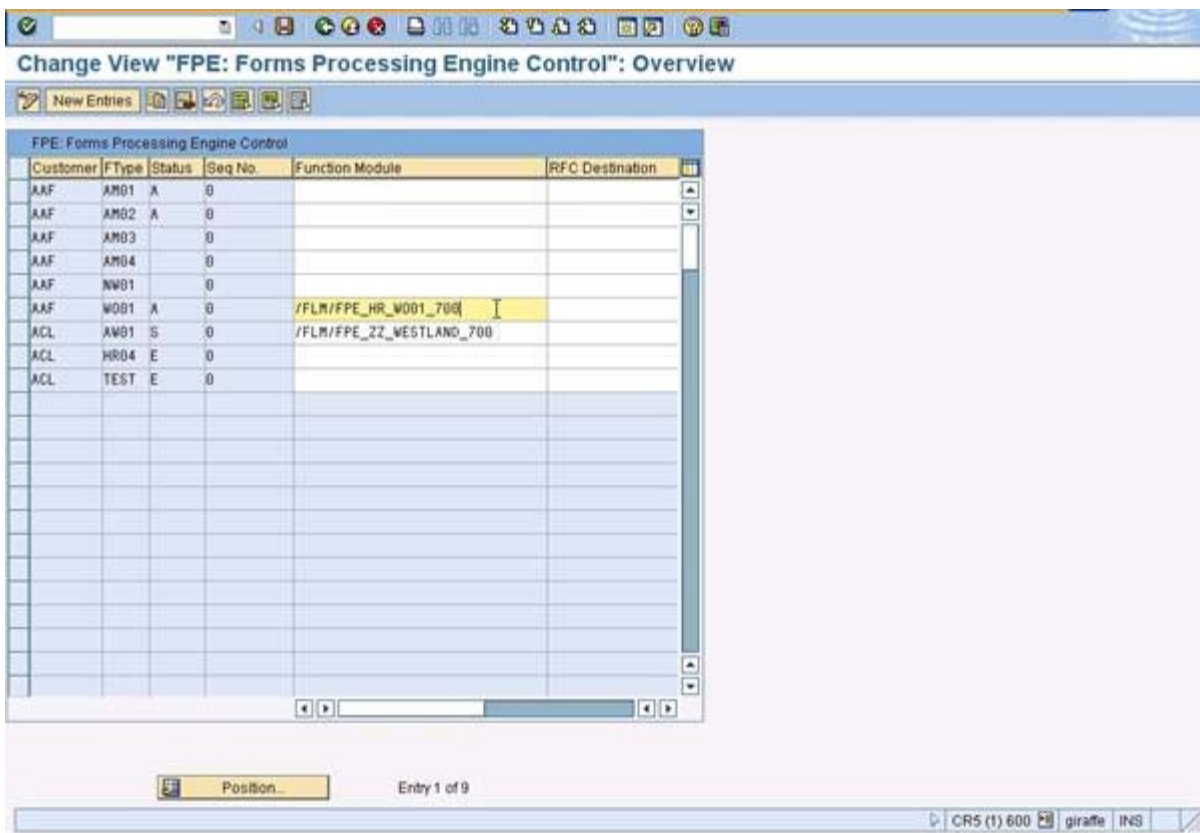


The 'Posting ok' check box makes a particular form type and status applicable for a posting attempt.

### 9.3 FPE function control

The control table view behind this option is called /FLM/FPE\_CNTRL. On execution the following screen is displayed:





Customer	FType	Status	Seq No.	Function Module	RFC Destination
AAF	AMB1	A	0		
AAF	AMB2	A	0		
AAF	AMB3		0		
AAF	AMB4		0		
AAF	NWB1		0		
AAF	WOB1	A	0	/FLM/FPE_HR_WOB1_700	
ACL	AMB1	S	0	/FLM/FPE_ZZ_WESTLAND_700	
ACL	HR04	E	0		
ACL	TEST	E	0		

This controls the function module invoked by FPE when processing is initiated by the main program /FLM/FPE\_INVOKE. Note that there is a sequence number in this table as multiple processing functions are allowed. It is preferable that the function modules use the naming convention as follows. /FLM/FPE\_XX\_YYYY\_VVV where

- XX is the SAP module where possible or ZZ if the processing is not module specific eg. SD
- YYYY is the 4 character form ID the posting module relates to
- VVV is the minimum SAP version required eg. 700

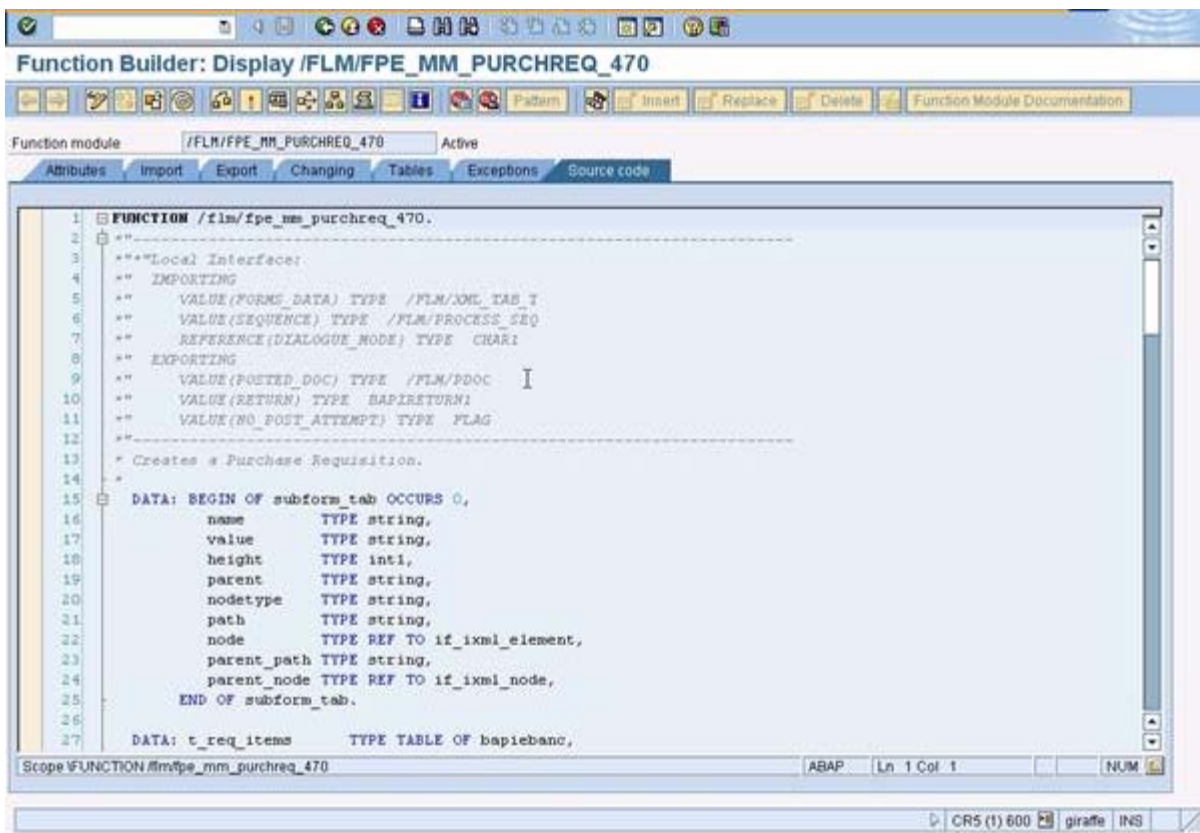
The FPE framework checks the SAP version is at the minimum level required to run the posting function so the last 3 characters of the function module **must** be the SAP version of the machine that will process the form.

The next field on the attached screen shot is for an RFC destination. FPE can post forms data to functions residing on remote SAP systems. So FLM can be running on one SAP instance but data is posted to a separate SAP instance. This is to allow for the scenario where an installation eg. is running a 4.7 ERP system so FLM is installed on its own Netweaver system and posts forms data to the 4.7 box.

## 9.4 Process Functions

Forms data is posted into a back end SAP system by normal SAP function module developed in SE37 like any other function. Its is required to use a predefined set of import and export parameters.

A screen shot of the parameters in an existing posting function is shown below:



The import parameters are:

- Forms\_Data – An internal table of the data extracted from the form so that the function can extract / convert it prior to a posting attempt.
- Sequence – The sequence number passed in from the control table /FLM/FPE\_CNTRL. Its is possible for one posting function to be invoked multiple times with each call being distinguished by a change in sequence number. One posting function could therefore be written to perform multiple functions. This is not generally used and is usually 0.
- Dialogue\_mode – Some posting functions can be run interactively to facilitate the ease of identifying a posting problem. To allow this the dialogue mode is passed into the adaptor from the /FLM/FPE\_INVOKE program selection screen. [Note that this facility is not possible for postings into SAP by non screen related functions such as BAPI's.]

The export parameters are:

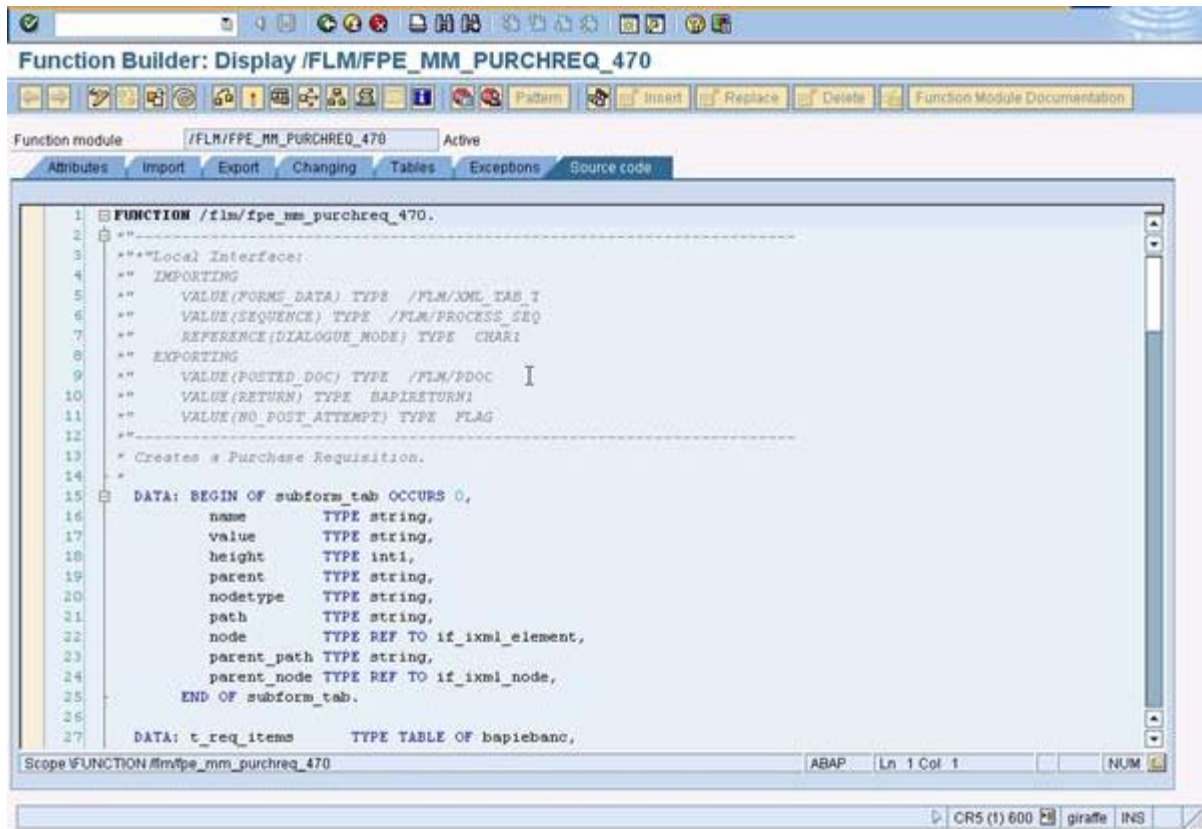
- Posted\_doc – the document number or reference related to a successful posting is assigned to this parameter. The FPE framework checks this variable for a non initial value on return from invocation. A value indicates a successful posting. Initial value indicates the posting failed.
- Return – SAP standard BAPI return structure for the return of error messages to the FPE framework.
- No\_post\_attempt – indicates to the FPE framework that no attempt to post was made – so neither success or failure.

A full example of a posting adaptor is attached at the end of this document as Appendix A.

## 9.5 Invoking FPE

Forms data is posted into a back end SAP system by normal SAP function module developed in SE37 like any other function. Its is required to use a predefined set of import and export parameters.

A screen shot of the parameters in an existing posting function is shown below:



The import parameters are:

- Forms\_Data – An internal table of the data extracted from the form so that the function can extract / convert it prior to a posting attempt.
- Sequence – The sequence number passed in from the control table /FLM/FPE\_CNTRL. Its is possible for one posting function to be invoked multiple times with each call being distinguished by a change in sequence number. One posting function could therefore be written to perform multiple functions. This is not generally used and is usually 0.
- Dialogue\_mode – Some posting functions can be run interactively to facilitate the ease of identifying a posting problem. To allow this the dialogue mode is passed into the adaptor from the /FLM/FPE\_INVOKE program selection screen. [Note that this facility is not possible for postings into SAP by non screen related functions such as BAPI's.]

The export parameters are:

- Posted\_doc – the document number or reference related to a successful posting is assigned to this parameter. The FPE framework checks this variable for a non initial value on return from invocation. A value indicates a successful posting. Initial value indicates the posting failed.
- Return – SAP standard BAPI return structure for the return of error messages to the FPE framework.
- No\_post\_attempt – indicates to the FPE framework that no attempt to post was made – so neither

success or failure.

## 9.6 Posting adapter coding

All posting adapters must have the following import parameters:

```
FORMS_DATA TYPE    /FLM/XML_TAB_T
SEQUENCE TYPE    /FLM/PROCESS_SEQ
DIALOGUE_MODE TYPE  CHAR1
```

And the following export parameters:

```
POSTED_DOC TYPE    /FLM/PDOC
RETURN TYPE  BAPIRETURN1
NO_POST_ATTEMP TYPE  FLAG
```

Normally the posting adapter will loop around the FORMS\_DATA internal table, taking the data from the form and filling other internal tables and/or structures required as import parameters by BAPIs to make the final SAP update.

Any BAPI returns are passed back and generated document number is also returned.

```
TYPES: t_return TYPE TABLE OF bapireturn,

DATA: subform_tab TYPE /FLM/XML_TAB_T,
      subform_wa_t TYPE /FLM/XML_TAB_T.
      w_return TYPE t_return,
      path_tab TYPE TABLE OF string,
      l_path_part TYPE string,
      l_parent_path TYPE string,
      l_parent_path_c(80) TYPE c,
      l_parent_path_len TYPE i,
      l_subform(3) TYPE n,
      t_lines TYPE i.
```

Call method **/FLM/SFS-> DATA\_ADD\_PARENT\_PATH** passing in FORMS\_DATA and receiving back SUBFORM\_TAB.

Now we have all the parent paths we can loop at this to get all the form data for a particular instance of a subform.

Use the following syntax for fields in non-repeating subforms:

```
READ TABLE forms_data ASSIGNING <f_formfld>
  WITH KEY name = 'DELIV_EXT'.
  <bapi_import_structure-field> = <f_formfld>-value.
```

Use the following syntax for fields in repeating subforms:

```
* Get the first occurrence of an item field:

READ TABLE subform_tab ASSIGNING <subform> WITH KEY name = 'MATNR'.

  l_parent_path_c = <subform>-parent_path.
  l_parent_path_len = STRLEN( l_parent_path_c ) - 3.
  l_subform = 1.

  WHILE l_subform LT 4.
    MOVE l_subform TO l_parent_path_c+l_parent_path_len(3).
    MOVE l_parent_path_c TO l_parent_path.
    CLEAR: subform_wa_t, wa_inb_del_item.
```

```

*
LOOP AT subform_tab ASSIGNING <subform>
WHERE parent_path = l_parent_path.
    APPEND <subform> TO subform_wa_t.
ENDLOOP.

DESCRIBE TABLE subform_wa_t LINES t_lines.

IF t_lines GT 0.

* Now we have a table of the fields in just one row.

    READ TABLE subform_wa_t ASSIGNING <subform> WITH KEY name = 'MATNR'.
    <bapi_import_item_wa-field> = <subform>-value.

READ TABLE subform_wa_t ASSIGNING <subform>
    WITH KEY name = ...
...

* Now append the item row to the BAPI import internal table parameter

APPEND <bapi_import_item_wa> TO <bapi_import_item>.

    ENDIF.

ELSE.

    EXIT.

ENDIF.

ADD 1 TO l_subform.

ENDWHILE.

```

Once all the BAPI import parameters are filled then the BAPI is called and the results passed back to the calling program.

## 9.7 Posting adaptor code to save flat version of the form

The following code will save a copy of the current form onto a specified LAN location.

**\*\*NOTE :** The method /flm/save\_pdf is available on FLM release 295. For eariler versions please see

"1. Saving a flat PDF to a LAN location" under [ABAP Examples](#). \*\*

```

DATA: l_data    TYPE /flm/xml_tab,
      l_action  TYPE c,
      l_return  TYPE string,
      l_cms_doc TYPE /flm/cms_doc,
      l_email   TYPE string,
      l_filename TYPE filename,
      l_filepath TYPE char50.

```

```

*
CLEAR l_data.
*
* Look for FLM_RETURN in data table to find form details string

```

```

*
  READ TABLE forms_data INTO l_data WITH KEY name = 'FLM_RETURN'.
  l_return = l_data-value.
*l_return = 'S+ACL-FIMS-E-00-1000003313-0006+na@arch.co.uk'.
*
* Structure of return parameter is:
*
*   ACTION + CCODE - FTYPE - FLANG - FVER - FID - FID_VAR + REC_EMAIL
*
* Split string at + separators
*
  SPLIT l_return AT '+' INTO l_action
                        l_cms_doc
                        l_email.
*
* FLM_RETURN HAS OLD VARIANT. USE FUNCTIN TO GET NEW VARIANT.
*
  CALL METHOD /flm/core=>get_next_variant
    CHANGING
      ch_cms_doc = l_cms_doc.
*
* Call method to split string further and return individual components
*
  CALL METHOD /flm/core=>split_xdp_cms_doc
    EXPORTING
      im_cms_doc = l_cms_doc
    IMPORTING
      ex_ccode   = g_fpe-ccode
      ex_ftype   = g_fpe-ftype
      ex_flang   = g_fpe-flang
      ex_fver    = g_fpe-fver
      ex_fid     = g_fpe-id
      ex_fid_var = g_fpe-id_var.
*
* Set the location where the PDF needs to be saved
*
  l_filepath = 'C:\Users\nahmad\Desktop\'.
*
* Define a name for the PDF to be saved as
*
  l_filename = g_fpe-id.
*
* Call function which will produce and save the PDF
*
  CALL FUNCTION '/FLM/SAVE_PDF'
    EXPORTING

```

```

    im_filename      = l_filename
    im_filepath      = l_filepath
    im_ccode         = g_fpe-ccode
    im_ftype         = g_fpe-ftype
    im_fver          = g_fpe-fver
    im_flang         = g_fpe-flang
    im_fid           = g_fpe-id
    im_fid_var       = g_fpe-id_var
EXCEPTIONS
    multiple_forms   = 1
    no_forms_found   = 2
    error_saving_file = 3.
*
* Handle return
*
IF sy-subrc <> 0.
    CLEAR posted_doc.
    IF return-type = ".
        return-type = 'E'.
    ENDIF.
*
    EXIT.          "Bail out of any more attempts
*
ELSEIF sy-subrc = 0.
    posted_doc = 'SAVED'.
ENDIF.
ENDFUNCTION.

```

[IMG Execution Tools](#) <- Previous - [Table of Contents](#) - Next -> [FLM Routing Server](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 10 FLM Routing Server

09/06/09 09:27:01

[Subscribe](#) to receive email when this article changes.

- 10.1 Using the Routing Tables for Form Submission.
- 10.2 FLM Routing Server for triggering Off-line forms.
- 10.3 FLM Routing Server for Form Escalation.
- 10.4 FLM Routing Server for Reminder E-mails.
- 10.5 Portal task instructions

The FLM Routing Server is a service program that runs as a background job, and performs actions on forms based on their system status. The set-up required for the routing server depends on the business process to be mapped. This document describes the set-up for four types of functionality:

- Using the Routing Tables for Form Submission.
- FLM Routing Server for triggering Off-line forms.
- FLM Routing Server for Form Escalation.
- FLM Routing Server for Reminder E-mails.

## 10.1 Using the Routing Tables for Form Submission.

When a form is submitted back to SAP, either for an on-line scenario or for an off-line scenario, the form is returned with its original status plus the 'action' selected by the submitting user. The system follows the routing server logic to determine a new status, new user etc. for the submitted form during the update to SAP.

### 10.1.1 Table /FLM/WF\_STAT: Form Status Derivation

Activity: Routing Status Table



Cust	FType	Status	Action	Status	New Trans	Task	Render Options	Send Notif	Notif Tbl
Arch	Test of DM1	Initial	Submt	S	[icon]	[icon]	Render Options	<input type="checkbox"/>	[icon]
[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]
[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]	[icon]

The form derivation table is keyed on customer, form type, form status and action.

The following information is derived from this table:

- New form status
- *For example an initial form with an action 'submit' may derive the new status 'submitted'*
- New form mode



- This can be set to 'on-line' or 'off-line'.
- E-mail notification settings
- There is an e-mail notification flag, plus standard texts for the e-mail subject and e-mail body.
- Render options. This can be set to:
  - Dynamic: allows the user to use dynamic template functions such as repeating rows
  - Static: allows user to put 'stamps' and annotations on the form, but does not support dynamic properties.
- Form Tagging: allows data to be read by other programs, e.g. JAWS software, but increases form size.

Within the form status derivation table we need to map every possible status of the form, and every possible action that can be performed for each status, such that a business process is mapped; it is this table that defines the form's lifecycle.

***Example of a simple off-line form routing without approval:***

Form type	Status IN	Action	Status OUT	Version OUT	Mode	E-mail Notification
AB01	Initial	Submit	Submitted			

Note that as there are no approval steps, the final form status is 'Submitted'.

***Example of a simple on-line form routing with approval:***

Form type	Status IN	Action	Status OUT	Mode	E-mail Notification
AB01	Initial	Submit	Submitted	On-line	X
AB01	Submitted	Approve	Approved	On-line	
AB01	Submitted	Reject	Rejected	On-line	X
AB01	Rejected	Submit	Submitted	On-line	X

This example represents an on-line form, with an approval step. When the form is submitted or rejected, an e-mail notification will be triggered to the next person in the business process.

***Example of an on-line form with off-line approval:***

Form type	Status IN	Action	Status OUT	Mode	E-mail Notification
AB01	Initial	Submit	Submitted	Off-line	

AB01	Submitted	Approve	Approved	On-line	
AB01	Submitted	Reject	Rejected	On-line	X
AB01	Rejected	Submit	Submitted	Off-line	

Note that an e-mail notification is not sent to an off-line user, since they will already be receiving an e-mail with the PDF form.

### 10.1.2 Table /FLM/WF\_USER: Form owner Derivation

The user derivation table is keyed on form type, status, current owner and action. This table is used to derive the new form owner only.

FLM: Form Routing Prototyping table					
FType	User Name	Status	Action	User Name	
ADBE		P	Reject	USER3	
ADBE	USER2	P	Reject	USER3	

This table can be used in scenarios where the number of users is very low, or when there is no other source of organizational data available to determine who should approve a form. Its main use might be for workshopping or prototyping a business process without the need to develop code to determine the user.

### 10.1.3 User-exits

In practice the new form owner is likely to be derived from an existing data source such as the SAP HR Organisational structure.

A user-exit will be available to determine the next form owner, based upon any SAP data including the form posting table (FLM/FPE) and form history table (FLM/FPE\_H) such that previous form owners can be determined. *(This is of particular importance when deriving the new owner in the case of form rejection.)*

### 10.1.4 Variable substitution in e-mail standard texts

See:

[Variable Substitution in Texts and Messages](#)

for details of variable substitution.

## 10.2 FLM Routing Server for triggering Off-line forms.

During a status change as defined within table /FLM/WF\_STAT, if the mode flag is set to 'off-line' then this will trigger the render and distribution of an off line form. The settings for the off-line form e-mail are stored in table /FLM/EMAIL.

### 10.2.1 Table /FLM/EMAIL: Off-line form e-mail settings

This table stores the standard texts for the subject, body and attachment name, plus the recipient e-mail address for a form. The full key includes the status and version of the form, so that different versions of the form at different statuses would be sent to different recipients.

Change View "FLM: Offline form settings": Overview

New Entries

FLM: Offline form settings								
Customer	FType	Langua	Version	Status	Receiver Email	Title Text	Body Text	Attachment Name
ACL	ADBE	EN	00		ag@arch.co.uk	/FLM/OFFLINE_EMAIL_TITLE	/FLM/OFFLINE_EMAIL_BODY	/FLM/OFFLINE_EMAIL_ATT_NAME
ACL	ERCS	EN	00		ag@arch.co.uk	/FLM/OFFLINE_EMAIL_TITLE	/FLM/OFFLINE_EMAIL_BODY	/FLM/OFFLINE_EMAIL_ATT_NAME
ACL	LGCA	EN	00		ag@arch.co.uk	/FLM/OFFLINE_EMAIL_TITLE	/FLM/OFFLINE_EMAIL_BODY	/FLM/OFFLINE_EMAIL_ATT_NAME

## 10.3 FLM Routing Server for Form Escalation.

The FLM Routing Server program /flm/wf\_engine should be scheduled as a background job and performs two functions: (a) Escalate forms, (b) Send e-mail reminders.

FLM Routing Server

Restrict Form Selections

Customer Code

Form Type

to

to

Perform Routing Actions

☒ Escalate

☒ Send E-mail reminders

☐ Test only

The form escalation job should be run at least once per day. For form escalation the routing server checks on all forms to check whether they need to be escalated to another user. The form escalation settings are stored in table /FLM/WF\_ESCA.

### 10.3.1 Table /FLM/WF\_ESCA: Form Escalation settings

The form escalation table is keyed on customer, form type and status. For any form at any particular

status an escalation window (in days) can be set, plus an escalation action, which will be taken for any form that exists in the status for the escalation window.



Cust	FType	Status	Remind	Days	Hrs	Mins	Resend	Rem Title Text	Rem Body Text
ACL	ST	Initial	<input type="checkbox"/>	0	0	0	<input type="checkbox"/>		
ACL	TEST	R	<input type="checkbox"/>	0	0	0	<input type="checkbox"/>		
ACL	TEST	S	<input type="checkbox"/>	0	0	0	<input type="checkbox"/>		
ACL	TEST	vender details	<input type="checkbox"/>	0	0	0	<input type="checkbox"/>		

The escalation action pushes the form one step along its route, for example, a submitted form could be automatically rejected [or approved] if no approval was granted within 5 days.

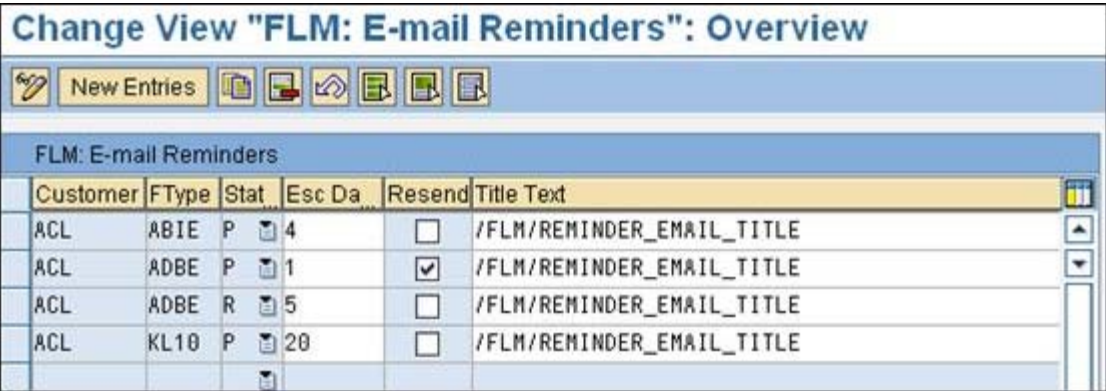
The effect of the action is to trigger five potential updates:

- Change the form owner
- Change the form status
- Change the form version
- Trigger an offline form
- Trigger an e-mail notification

These actions are configurable using tables /flm/wf\_stat, /flm/wf\_user and /flm/email as described in sections 10.1 and 10.2.

### 10.4 FLM Routing Server for Reminder E-mails.

The second task the routing server provides is to send out reminder e-mails to users who have not processed a form. Normally this would be used in an approval scenario; if the form has been neither approved or rejected then a reminder is sent out. The functionality is similar to the escalation functionality, since a ‘window’ of days is defined after which a reminder e-mail will be sent out.



Customer	FType	Stat	Esc Da	Resend	Title Text
ACL	ABIE	P	4	<input type="checkbox"/>	/FLM/REMINDER_EMAIL_TITLE
ACL	ADBE	P	1	<input checked="" type="checkbox"/>	/FLM/REMINDER_EMAIL_TITLE
ACL	ADBE	R	5	<input type="checkbox"/>	/FLM/REMINDER_EMAIL_TITLE
ACL	KL10	P	20	<input type="checkbox"/>	/FLM/REMINDER_EMAIL_TITLE

If both an escalation window and a reminder window are defined for the same form and same status,

then the reminder window will be set to be less than the escalation window. For example, a submitted for that is waiting approval might trigger a reminder after 2 days, and then be escalated after 4 days.

### 10.4.1 Table /FLM/WF\_REMI: Form Reminder settings

The e-mail reminder window is defined by form status on table /FLM/WF\_REMI

Standard texts for the e-mail subject and e-mail body are also derived from this table, and variable substitution is possible as described in section 10.1.4.

There is also a 'Resend' flag, which controls whether several e-mail reminders will be sent out for the same form/status.

FLM Routing Server must only be run once per day for any form type, as running this multiple times on a single day would generate multiple reminders on the same day regardless of the 'Resend flag'.

The logic behind the selection of the form for a reminder e-mail is as follows:

[1] Is today the last day of the reminder window?

[2] Has the reminder window passed AND is the Resend flag set?

If the answer to either of these is yes then a reminder e-mail is generated.

For example, if the reminder window was set to be 2 days and the resend flag was not set then the owner would receive a reminder on day 2 only. However, if the reminder window was set to be 2 days and the resend flag was set, then the owner would receive a reminder on day 2 and on each subsequent day until he/she processed the form, or the form was escalated by FLM Routing Server.

## 10.5 Portal task instructions

Portal Task Instructions messages can contain variables &1, &2, &3 and &4. The values from the first 4 Index fields will be substituted in at runtime and hence appear in the Portal.

Asset Incident Log	Form submitted for your...	Initial	USER1	3/7/2008	3/7/2008	1000002439
--------------------	----------------------------	---------	-------	----------	----------	------------

To use this facility, use transaction SE91 and message class /FLM/[3digit cust code] e.g. /FLM/ACL. Enter the required portal message (such as 'Please approve this form').

To make use of the variable capability, use '&' signs to denote an index field. (e.g. 'please approve a form for &1'. You can then set up &1 to contain the name of the person who submitted the form:

- In the 'Forms Type Configuration' activity, enter the name of the form fields you wish to index and click save to prompt a customising request.

Table ViewEditGotoSelectionUtilities(M)SystemHelp

Change View "FLM: Form Types": Details

New Entries

Index Options

Customer	ACL
Form Type	TEST
Language	E
Form Version	00

Choose from the drop downs up to 6 form fields you wish to index form data with. 3 fields can hold up to 12 chars, 3 up to 40 chars

Index 1 (12)	NAME
Index 2 (12)	
Index 3 (12)	
Index 4 (40)	
Index 5 (40)	
Index 6 (40)	

Alternatively you can populate the indexes programmatically using the Form User Exit Index

[Form Posting Engine \(FPE\)](#) <- Previous - [Table of Contents](#) - Next -> [Correspondence Generation](#)

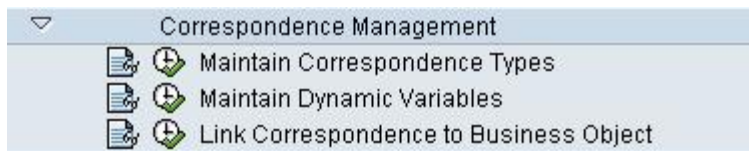
# 11 Correspondence Generation

07/11/08 10:59:07

[Subscribe](#) to receive email when this article changes.

## 11.1 Developing a letter template and texts

Correspondence generation allows the creation of letters, based on a template created in Adobe Designer. The text comes from SAP standard texts, which can contain variables that take on values from the SAP database at runtime.



## 11.1 Developing a letter template and texts

A developer creates the template and texts, which an administrator can then use to create a standard letter.

1. Create Letter Template in Adobe Designer
2. Create Standard Texts (letter paragraphs) in transaction SO10. Define variables by '&' characters. Any FPE or SYST variable can be accessed, as well as '&FORMNAME&' (description of correspondence type) and '&URL&' (encrypted hyperlink for notification and reminder emails)
3. Go to 'maintain dynamic variables' and enter business logic to derive each of the variables (listed by customer code, not letter type)
4. Go to 'maintain correspondence types' and select a letter template to view and arrange its constituent paragraphs (standard texts). Here you can configure the permissions the administrator will have on each paragraph; for example you can make them read-only, editable, or editable (but with warning)
5. (Go to transaction SE91 to create any paragraph warning messages)
6. Select the message text in 'maintain correspondence types' if required
7. Each correspondence can update a SAP object. The object is determined by its SAP Business Object name and so can be any object in the SAP system.
8. Go to the 'link correspondence to business object' activity and enter the properties of any system table you wish to update
9. Create a routing for the letter if the template will be contributed to by more than one person. This is done using the standard FLM routing activities; see the section on form routing.

[FLM Routing Server](#) <- Previous - [Table of Contents](#) - Next -> [Form Structure](#)

---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)



# 12 Form Structure

07/11/08 11:00:18

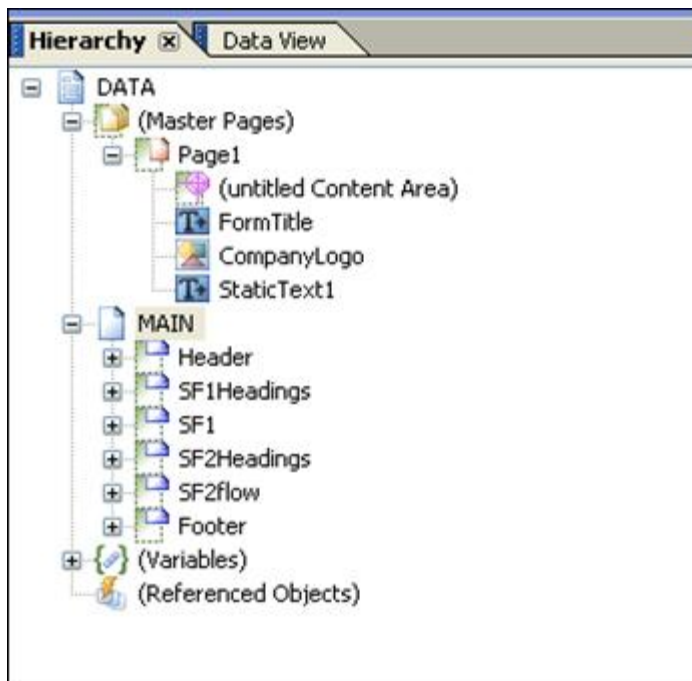
[Subscribe](#) to receive email when this article changes.

- 12.1 Data hierarchy
- 12.2 Subform definition and binding
- 12.3 Subform look and feel hints and tips.

## 12.1 Data hierarchy

The top-level node in the data hierarchy represents the form. There are no settings to configure on this node. It is suggested that this node is called the same name as the 4-digit 'Form Type' in FLM.

### A. Data hierarchy in Adobe Designer



Beneath the Data hierarchy are three types of node:

- **Master pages**

At least one master page must be defined. Typically the form header/footer/logos – anything that should be repeated on each page, should be put inside the master page. Also a Content Area must be defined, which is basically a box defining the boundaries for all the other form fields. This ensures that form fields do not overlap with headers and footers on the master. The Content area by default does not have a name it should be named and by default this should be called 'content'. This is critical when coding as un-named objects or subforms make referencing in script very difficult.

- **Subforms**

There are two types of subform; 'Flowed' and 'Positioned'.

The 'Flowed' subforms generally follow the hierarchy represented by the xsd data schema. The 'Positioned' subforms represent physical groupings of fields at the same logical level within the xsd data schema. A flowed approach will allow the form to work in a dynamic nature. For example, for an invoice you would define one header, one detail line, and one footer. The data would then be bound and for each line on the invoice a new detail line will be created, and the Reader will then flow all trailing subforms correctly.

#### □ **Variables**

There are two types of variable; 'Variables' and 'Script Objects'.

Variables are used for storing any data string. You can use Javascript to read or write to these variables. 'Script Objects' are (groups of) Javascript sub-routines that can be called at any point using Javascript behind form/field events.

#### □ **Referenced Objects**

These are objects that are not naturally occurring in the form, but may appear based upon certain events. The classic example of this is a Footer subform that should only appear if a page break occurs on another subform.

## **12.2 Subform definition and binding**

The top-level subform is always FLOWED.

As you work your way logically through the subform then each subform is FLOWED until the final subform in which data fields sit, which is POSITIONED. Fields must always sit inside POSITIONED subforms if they are to be displayed, as it is not possible to physically position a field inside a FLOWED subform.

It is essential to have this hierarchy of flowed subforms since the option to set a subform to be repeating is only available if the parent subform (the next level up) is FLOWED. However, it is possible for either a FLOWED or a POSITIONED subform to be repeating as long as the parent subform is FLOWED.

It is not necessary to have a 1:1 match between the subforms in the form and the subforms designed within FLM: The form will often require additional subforms in order to handle the field positions as well as the data flow.

It is not necessary to bind each subform in the data hierarchy, unless the subform has repeating rows or is nested within another subform that has repeating rows. In other cases the binding of the field is sufficient.

### **12.2.1 Binding for non-repeating subforms.**

There is no need to bind non-repeating subforms as all the fields within those subforms are effectively at root level, be they header fields, item headings, footer fields etc.

The binding of the fields inside non-repeating subforms takes the following form:

```
$record.Header.form_status
```

In this example 'Header' is the data node in the xsd data schema and 'form\_status' is the field name under the 'Header' node in the xsd data schema. If there were more sub-nodes in the data schema then they would all appear in the binding path.

12.2.2 Binding for repeating subforms.

It is essential to bind subforms that can repeat.

The binding of the subform takes the following form:

```
$record.Header.form_status
```

In this example, 'EmployeeUtil' is the data node in the xsd data schema that sits underneath the root ('DATA') node. 'ResUtilisation' is the data node in the xsd schema that contains repeating fields. The notation of the [\*] means for all nodes that match this Xpath query.

If the repeating subform was defined with a parent of 'ROOT' then the binding would take the following form, as no path of the node hierarchy would be necessary:

```
$record.ITEM[*]
```

In this example, of course, the node is called 'ITEM' and its parent is the root 'DATA' node.

The binding of a repeating subform is different depending on whether there are any nested subforms within the subform to be bound.

If there are no nested subforms then the POSITIONED subform is set to have repeating rows, and is bound to the data node.

If there are further nested subforms within the subform that contain data to be bound, then the FLOWED subform is set to have repeating rows and is bound to the data node.

This often means that there is a POSITIONED subform without binding (ie 'Normal' binding) that sits in-between the bound FLOWED subform and the fields within that subform. This in turn means that the binding for those fields is takes a different form.

Note that there is an option to define a data schema with separate nodes for the repeating subform and for all the fields at the level of repeating subform:

Workorder	FLOWED, REPEATING
Workorder_info	POSITIONED
<fields>	
Workorder_details	FLOWED

Employee_details <fields>	POSITIONED, REPEATING
Plant_details <fields>	POSITIONED, REPEATING

In this scenario the subforms 'Workorder', 'Employee\_details' and 'Plant\_details' must be bound, but it is not necessary to bind the subform 'Workorder\_info'. (While it is not needed it is best practice to always bind subforms to the corresponding node.

However, if there is a corresponding data node (ie. There are no data fields defined under the 'workorder' node, just nodes for info, employees and plants) then the Workorder\_info subform should be bound, and this means that the binding is the same as for a normal POSITIONED subform regardless that there are nested subforms.

In the scenario where the POSITIONED subform is bound then the binding for fields within a repeating subform takes the form:

EBELP

The immediate parent of the field defines the full data path so there is no need to define it again. Binding always work on a relative path unless they start with record in which case they become absolute paths.

In the scenario where the FLOWED subform is bound but the child POSITIONED subform is not bound, then the binding for fields within a repeating subform takes the form:

`$record.ITEM[ * ].EBELP`

This is because the immediate parent of the field (the POSITIONED subform) has no binding, so the full data path is required.

Do not attempt to bind both the FLOWED and POSITIONED subforms to the same data node.

### 12.2.3 Binding for nested subforms.

The binding for nested subforms follows exactly the same pattern as above. This means that if there are further nested subforms then the FLOWED subform must be bound, otherwise the POSITIONED subform should be bound.

Since nested subforms always sit inside FLOWED subforms that are bound, then the binding of the nested subform is not fully declared, but instead it is just the subform name:

`plantdata[*]`

The binding of the fields is the same as described in the previous section.

**Note that in all cases the bound subform is the subform that is set to have repeating rows.**

## 12.3 Subform look and feel hints and tips.

All subforms in the data hierarchy should be set to 'Allow page breaks within content' except for the bottom-level positioned subforms where it may be desirable to keep the form fields together. The 'allow Page break' will allow a set of flowing subforms to natural flow over a page and then you can define a trailer (footer) and header subform to be placed on the next page.

These can include Referenced Subforms. If you have a group of subforms that you want to bind together then set the Page Break option of and then use the 'Keep With' Flag. An example of this is an invoice line with special details section which could be long text that could span one or more lines. Therefore running of the Page Break option here and setting the keep with options will ensure if the detail line fits but the special instructions do not it will place both on a new page.

[Correspondence Generation](#) <- Previous - [Table of Contents](#) - Next -> [Methods for Form Data Handling](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# 13 Methods for Form Data Handling

29/10/08 16:26:39

[Subscribe](#) to receive email when this article changes.

---

## Table of Contents

1. Get the complete address details from an address number
2. Get the complete address details from a partner number
3. Get e-mail address from partner number
4. Get address from address number into single text field
5. Get standard text into single text field
6. Prepopulate field within a subform
7. Add parent paths to form data xml table
8. Get HR Personnel number from user id
9. Get User ID from HR Personnel number
10. Get E-mail address from user id
11. Get E-mail address from HR Personnel number
12. Navigate HR organisational structure
13. Get previous form owner
14. Get previous form actioner
15. Get form name
16. Get form current owner
17. Get form current status

## Index of methods for form data handling

This section describes the other methods delivered as part of FLM that can be used for data handling in user-exits. Note the methods within the '/FLM/SAMPLE' class are commented out to enable FLM to be installed on NetWeaver servers without any SAP Application component installed.

### 1 Get the complete address details from an address number

**/FLM/SAMPLE=> ADRNR\_TO\_ADDR\_COMP**

IM_ADRNR	TYPE ADRNR	Address number
EX_ADDR_COMPLETE	TYPE SZADR_ADDR1_COMPLETE	Partner complete address

This method reads the complete address details from an address number.

## 2 Get the complete address details from a partner number

/FLM/SAMPLE=> GET\_PARTNER\_ADDR\_COMP

IM_PARVW	TYPE PARVW	Partner type
IM_PARNR	TYPE PARNR	Partner number
EX_ADDR_COMPLETE	TYPE SZADR_ADDR1_COMPLETE	Partner complete address

This method reads the complete address details from a partner type and number.

## 3 Get e-mail address from partner number

/FLM/SAMPLE=>GET\_PARTNER\_ADDR\_SMTP

IM_PARVW	TYPE PARVW	Partner type
IM_PARNR	TYPE PARNR	Partner number
EX_SMTP_ADDR	TYPE AD_SMTPADR	Partner e-mail address

This method reads the e-mail address from a partner type and number.

## 4 Get address from address number into single text field

/FLM/SAMPLE=>ADRNR\_TO\_TEXT\_FIELD

I_ADRNR	TYPE ADRNR	Address number
O_ADDRESS	TYPE STRING	Formatted address

Note that in FLM version 261 we do not include the country in the formatted address; this method should be cloned if any address lines are required that are missing from the returned address.

## 5 Get standard text into single text field

/FLM/SAMPLE=>READ\_TEXT\_TO\_TEXT\_FIELD

IM_TDID	TYPE TDID	Text ID
IM_SPRAS	TYPE SPRAS	Language
IM_TDNAME	TYPE TDOBNAME	Name
IM_TDOBJECT	TYPE TDOBJECT	Object
EX_TEXT	TYPE STRING	Output text

This method reads the contents of a standard text and concatenates them into a single string, adding in carriage return codes at the end of each line so that the standard text is easily formatted when bound to a form.

## 6 Prepopulate field within a subform

/FLM/SAMPLE=>FIELD\_PREPOPULATE

IM_DATA	TYPE /FLM/XML_TAB_T	Table type for XML table
IM_SUBFORM	TYPE STRING	Parent subform name
IM_ROW	TYPE INT3	Parent subform row instance
IM_FIELD	TYPE STRING	Field name
IM_VALUE	TYPE STRING	Field value
EX_DATA	TYPE /FLM/XML_TAB_T	Table type for XML table

This methods is used for repeating subform handling within form prepopulation.

## 7 Add parent paths to form data xml table



**/FLM/SAMPLE=>DATA\_ADD\_PARENT\_PATH**

IM_DATA	TYPE /FLM/XML_TAB_T	Table type for XML table
EX_DATA	TYPE /FLM/XML_TAB_T	Table type for XML table

This method is used for repeating subform handling within posting adapters.

**8 Get HR Personnel number from user id**

**/FLM/SAMPLE=>UNAME\_GET\_PERNR**

IM_UNAME	TYPE UNAME	User Name
IM_SUBTY	TYPE SUBTY DEFAULT '0001'	Sub type
IM_DATUM	TYPE DATUM	Date
IM_OBJPS	TYPE OBJPS DEFAULT "	Object ID
IM_SPRPS	TYPE SPRPS DEFAULT "	Lock indicator
EX_PERNR	TYPE PERSNO	Personnel number

The link between a user name and the personnel number is stored in info type 0105, subtype 0001. The method does a simple selection on table PA0105.

**9 Get User ID from HR Personnel number**

**/FLM/SAMPLE=>PERNR\_GET\_UNAME**

IM_PERNR	TYPE PERSNO	Personnel number
IM_SUBTY	TYPE SUBTY DEFAULT '0001'	Sub type
IM_DATUM	TYPE DATUM	Date
IM_OBJPS	TYPE OBJPS DEFAULT "	Object ID

IM_SPRPS	TYPE SPRPS DEFAULT "	Lock indicator
EX_UNAME	TYPE UNAME	User Name

The link between a user name and the personnel number is stored in info type 0105, subtype 0001. The method does a simple selection on table PA0105.

## 10 Get E-mail address from user id

/FLM/CORE=>GET\_USER\_EMAIL

IM_USER	TYPE UNAME	FLM: Form Owner
EX_EMAIL	TYPE AD_SMTPADR	FLM: Form Action

This method returns the e-mail address from a user’s default data.

## 11 Get E-mail address from HR Personnel number

/FLM/SAMPLE=>PERNR\_GET\_EMAIL

IM_SUBTY	TYPE SUBTY DEFAULT '0010'	Sub type
IM_DATUM	TYPE DATUM	Date
IM_OBJPS	TYPE OBJPS DEFAULT "	Object ID
IM_SPRPS	TYPE SPRPS DEFAULT "	Lock indicator
IM_PERNR	TYPE PERSNO	Personnel number
EX_EMAIL	TYPE /FLM/EMAIL_RECE	E-mail address

The Link between a personnel number and their e-mail address is stored in info type 0105, subtype 0010. This method performs a simple selection on table PA0105.

## 12 Navigate HR organisational structure

/FLM/SAMPLE=>PERNR\_GET\_MANAGER

IM_PERNR	TYPE HROBJID	Personnel number
IM_PLVAR	TYPE PLVAR  DEFAULT '10'	Plan Version
IM_DATUM	TYPE DATUM	Date
IM_PERNR_PROLE_RELAT	TYPE RELAT  DEFAULT '008'	Relationship Between Objects
IM_PROLE_DEPT_RELAT	TYPE RELAT  DEFAULT '003'	Relationship Between Objects
IM_DEPT_SROLE_RELAT	TYPE RELAT  DEFAULT '012'	Relationship Between Objects
IM_SROLE_SPERNR_RELAT	TYPE RELAT  DEFAULT '008'	Relationship Between Objects
EX_SPERNR	TYPE HROBJID	Manager Personnel number

This method performs several selections on table HRP1001 passing in relationships to find an employee’s supervisor.

Note that this works only with the structure desribed below, and a check is required afterwards in case the employee passed in was a supervisor: in practise we may need to clone this method depending on the organisational structure in HR.

<b>Dept [O]</b>		
B003->	<b>Employee role [S]</b>	
	A008->	<b>Employee [P]</b>
B012->	<b>Supervisor role[S]</b>	
	A008->	<b>Supervisor [P]</b>

## 13 Get previous form owner

/FLM/CORE=>GET\_FORM\_PREV\_OWNER

IM_CCODE	TYPE /FLM/CUST_CODE	SFS: Customer Code
IM_FTYPE	TYPE /FLM/FTYPE_CODE	SFS: Form Type
IM_FLANG	TYPE /FLM/FLANG	FLM: Form Language
IM_FVER	TYPE /FLM/FVER	FLM: Form Version
IM_FID	TYPE /FLM/FID	FLM: Form ID
IM_FID_VAR	TYPE /FLM/ID_VAR	FLM: Form Variant
VALUE( RE_OWNER )	TYPE XUBNAME	User Name in User Master Record

This method finds the last previous form owner. It is useful for owner derivation within workflow user-exits for rejection actions.

## 14 Get previous form actioner

**/FLM/CORE=>GET\_FORM\_PREV\_ACTIONER**

IM_CCODE	TYPE /FLM/CUST_CODE	SFS: Customer Code
IM_FTYPE	TYPE /FLM/FTYPE_CODE	SFS: Form Type
IM_FLANG	TYPE /FLM/FLANG	FLM: Form Language
IM_FVER	TYPE /FLM/FVER	FLM: Form Version
IM_FID	TYPE /FLM/FID	FLM: Form ID
IM_FID_VAR	TYPE /FLM/ID_VAR	FLM: Form Variant
IM_ACTION	TYPE /FLM/FACTION	FLM: Form Action
VALUE( RE_OWNER )	TYPE XUBNAME	User Name in User Master Record

This method finds the last previous form owner who performed a specific action. It is useful for owner derivation within workflow user-exits for rejection actions.

# 15 Get form name

```
/FLM/CORE=>GET_FORM_NAME
```

IM_CCODE	TYPE /FLM/CUST_CODE	FLM: Customer Code
IM_FTYPE	TYPE /FLM/FTYPE_CODE	FLM: Form Type
IM_FLANG	TYPE SPRAS	Language
IM_FVER	TYPE /FLM/FVER	FLM: Form Version
VALUE( EX_FNAME )	TYPE /FLM/FNAME_L	SFS: Long Form Name

This method returns the long name for a form type.

# 16 Get form current owner

```
/FLM/CORE=>GET_FORM_OWNER
```

IM_CCODE	TYPE /FLM/CUST_CODE	SFS: Customer Code
IM_FTYPE	TYPE /FLM/FTYPE_CODE	SFS: Form Type
IM_FLANG	TYPE /FLM/FLANG	FLM: Form Language
IM_FVER	TYPE /FLM/FVER	FLM: Form Version
IM_FID	TYPE /FLM/FID	FLM: Form ID
IM_FID_VAR	TYPE /FLM/ID_VAR	FLM: Form Variant
VALUE( RE_OWNER )	TYPE XUBNAME	User Name in User Master Record

This method returns the current owner for a form.

# 17 Get form current status

/FLM/CORE=>GET\_FORM\_STATUS

IM_CCODE	TYPE /FLM/CUST_CODE	SFS: Customer Code
IM_FTYPE	TYPE /FLM/FTYPE_CODE	SFS: Form Type
IM_FLANG	TYPE /FLM/FLANG	FLM: Form Language
IM_FVER	TYPE /FLM/FVER	FLM: Form Version
IM_FID	TYPE /FLM/FID	FLM: Form ID
IM_FID_VAR	TYPE /FLM/ID_VAR	FLM: Form Variant
VALUE( RE_FSTATUS )	TYPE /FLM/FSTATUS	User Name in User Master Record

This method returns the current status of a form.

[Form Structure](#) <- Previous - [Table of Contents](#) - Next -> [Web Services](#)

# 14 Using Web Services

25/09/09 08:20:15

[Subscribe](#) to receive email when this article changes.

---

## Introduction

Web services allow you to connect ABAP (or other) application functionality directly to your form, or directly to your FLM user exits. If you already have a wsdl (Web Service Description Language) that you've downloaded from say Google, you can just import that into your forms as described in the developer guide.

If you have an ABAP function module you do the following to convert to a Web-service interface then follow the SOA runtime setup as described in the [Web Services Config and Create](#) wiki.

[Heartbeat Web Service](#) example gives a complete guide on of how to create, test and use a Web Service.

Also an example of [How to Consume an External Web Service via an ABAP Proxy Class](#).

## 14.1 How to Publish a Function Module as a Web Service

**To turn a Function Module into a Web Service (.wsdl) do the following:**

- Make the Function Module RFC enabled on the Attributes tab in SE37,
- you may also need to change your parameters to "call by reference" if necessary
- Then in the menu select /utilities/more utilities/create web service/from the function module
- Follow the wizard through it provides details at each screen.
- Transaction SOAMANAGER, this launches a browser session. Log in as an Administrator.
- Tab 'Business Administration', choose 'Web Service Administration'
- Enter a Search pattern and click Go
  - for example, to search for Z\* function modules enter Z\*)
- Click on row needed, click 'Apply Selection'
- If you need the WSDL file for import into Designer, do the next 2 steps:
  - Click on 'Open WSDL document for selected binding', this launches a new web page
  - Right click on web page, choose View source. Save file as 'MYWEBSERVICE.wsdl' – this is what you need to import into Designer
- Back to SOAMANAGER screen, choose Configurations tab.
- Choose Webservice, click Edit
- Enter ABAP service user/password at the bottom, click Save

## 14.2 Designer-based web services (Client-side Web services)

Web Services can be accessed via the data connection functionality in Adobe Designer. You need a .wsdl file to do this which is generated after you have web-enabled an ABAP function module and done the Web Services administration steps ([Web Services Config and Create](#)).

From Designer, Data View, Right click and then New Data Connection, WSDL file, Next and select the location of the required file.

Bind the import and export parameters to your screen fields as required, then add the following line of code into the triggering event you require (this can be anywhere, make sure it's java script at client):

```
xfa.connectionset.<Web service name>.execute(false);
//or 'true' if its a repeating section, this triggers a re-merge of the data,
//so 'false' by default for performance reasons
```

## 14.3 FLM-based Web Services (ie server side Web Services)

If you wish to consume an externally based web-service via ABAP proxies, please see here. The main article here is a specific case of this for a particular web service.

FLM ships with a VAT registration validation web service stored in class /FLM/WS\_LIB\_1. This webservice can be invoked from any ABAP user exit and requires the SAP server to have www access at the moment of processing. Also you must have done the SOA technical set up ([Web Services Config and Create](#)). The client side proxy (available in SE80) is called /FLM/CO\_CHECK\_VAT\_PORT\_TYPE.

To make use of this web service, you will need to

1. create two fields on the form: one into which the VAT country code is entered, and one to contain the VAT number itself. Call these fields "COUNTRY" and "VAT\_NUMBER".
2. Create a third field, called 'IS\_VALID', type = checkbox, and assign it a derivation routine
3. In the field-level derivation userexit on the IS\_VALID field, enter the following code behind the derivation userexit:

```
DATA:  l_country(2) TYPE c,
       l_vat_number TYPE string,
       l_data        TYPE /flm/xml_tab,
       l_valid        TYPE flag,
       l_message      TYPE string,
       l_var          TYPE symsgv,
       l_vl           TYPE symsgv,
       l_succ_mess    TYPE string.
*
CLEAR: l_country,
       l_vat_number,
       l_valid,
       l_message.

READ TABLE im_data INTO l_data WITH KEY name = 'COUNTRY'.
l_country = l_data-value.

*
READ TABLE im_data INTO l_data WITH KEY name = 'VAT_NUMBER'.
l_vat_number = l_data-value.

*
CHECK l_country IS NOT INITIAL AND
      l_vat_number IS NOT INITIAL.
```



```

*
CALL METHOD /flm/ws_lib_1=>eu_vat_registration_check
EXPORTING
    im_country_code = l_country
    im_vat_number   = l_vat_number
    im_port_name    = 'DEFAULT'
IMPORTING
    ex_valid        = l_valid
    ex_message      = l_message.
*
* Dispatch log message
*

IF l_message IS NOT INITIAL.
MOVE l_message(20) TO l_succ_mess.
ELSE.
IF l_valid IS INITIAL.
MOVE 'Not Valid ' TO l_succ_mess.
ELSE.
MOVE 'Valid' TO l_succ_mess.
ENDIF.
ENDIF.
*
CONCATENATE 'EU VAT check for' l_country '/' l_vat_number ':' l_succ_mess INTO l
_v1 SEPARATED BY space.
*
CALL METHOD /flm/core=>error
EXPORTING
    im_type   = /flm/core=>c_mess_success
    im_number = '997'
    im_v1     = l_v1.

IF l_message IS NOT INITIAL.
    l_var = l_message.
    ex_mess_num = '002'.
    ex_msgvar1 = l_var.
    ex_response = 'A'.
    CALL METHOD /flm/core=>error
        EXPORTING
            im_type   = /flm/core=>c_mess_warning
            im_number = '002'
            im_v1     = l_var.
ELSE.
    ex_value = l_valid.
ENDIF.

```

## 1. Save

The checkbox will be ticked if the VAT number supplied is valid.

[Methods for Form Data Handling](#) <- Previous - [Table of Contents](#)

---

Home: [FLM Documentation](#)

What's new: [Recently changed articles](#)

# Group Ownership of Forms

19/05/09 10:08:01

[Subscribe](#) to receive email when this article changes.

---

## **Introduction**

In the standard behaviour a form is routed from person to person and is therefore 'owned' by one single person at any one time; only the owner of a form can open the form and change the contents.

However, sometimes this behaviour is not sufficient. For example if the business requires that multiple users have access to the same form, and whoever launches the form first should lock out the other users in the group. This is known as 'Group Ownership of Forms' and is supported in FLM.

## **Overview of Setup Procedure**

There are three aspects to the setup of this behaviour:

### **1. Determination of the Group User**

In the standard behaviour we determine the next user in the routing either programmatically in the Routing userexit or by table entry from FLM Configuration. In either case we ultimately determine a real SAP user name, or, for off line forms a target email address. This makes no sense for Group Ownership of forms, as there is no one user we wish to route the form to.

There are two options here to achieve Group ownership:

1.1 Determine the special user '\*' either programmatically or via the FLM Configuration. This user will allow other users to pick up this form and make changes to it. This is the simplest option for achieving group ownership of forms.

1.2 If you require to determine a more meaningful user name, perhaps one that corresponds to the name of the particular group of users, you can configure a particular form type to allow group access, and then you may derive whatever username you wish. This requires an advance correction described in [FLM Note 0007](#). The main advantage here is that you can then link the group name directly to the actual SAP usernames of the group members, see 2) below.

### **2. Assignment of users into groups**

How you achieve this is not enforced by the FLM framework, and so here we list some of the most common ways of doing this:

1.1 via z-tables. Build a table which links the form type and status to a list of user master records in SAP. This can be maintained regularly as the groups change

1.2 via User Authorisations. Build a profile for each group you wish model in the system, attach this to users as required.

1.3 via User Parameters. Create a new parameter in table TPARA (use SM31), attach this parameter, with a value, to each user in the group.

1.4 via Substitutions in standard SAP Workflow. Here you can maintain a list of alternate processors for your workitems in the Business Workplace

However you achieve this link, you must be able to programmatically read the group in ABAP from inside the FLM InBox user exit, as described in 3)

### 3. Modification of the FLM Inbox contents

The standard FLM Inbox contains only those forms that you, the logged on user, are the owner of. Obviously in the case of the Group Ownership situation, this no longer applies, as a form is owner either by the special user '\*' or if you implemented FLM note 0007 by another group user.

In pseudo-code here is the process for redetermining the contents of the FLM Inbox programmatically, and below we will give a concrete example of some code using one of the assignment techniques from 2) above:

1. Determine if the logged-on user/form type/form status correspond to a group ownership situation, by reference to your mapping techniques defined in 2) above. *Note// it is up to you to design if this is a function of form type/form status or other variables.*
2. If this is *not* a group ownership situation, exit the User Exit with the standard Inbox contents intact
3. Decide if you need to delete any or all of the standard Inbox contents.
4. Rebuild the contents of the Inbox by reading the table /FLM/FPE to include all of those forms that you wish the group to see. Normally this would be a function of form type and form status, but again, the design is yours and may well be a function of other variables.

#### 3.1) Example of Inbox Redetermination

This example assumes only the status 'Approved' for form type 'ABCD' is to be shared amongst the groups users. The form owner is defined as '\*'. The groups are defined by users having a form type set against one of 5 user parameters (ie an example of section 1.3 in 2) above):

```
DATA:      w_newentry      TYPE /flm/get_inbox_user_1,
           w_group_access  TYPE flag,
           w_fpe           TYPE /flm/fpe,
           w_ccode         TYPE /flm/cust_code.
*
* Call Function for my form type to see if form 'ABCD' is shared for this user:
*
CALL FUNCTION 'Z_FLM_SHARED_USERS'
  EXPORTING
    requestor_id = im_user
    form_type    = 'ABCD' "<---call this several times other forms shared..
  IMPORTING
    group_acces  = w_group_access.
*
* If this user doesn't have group access for the form type ABCD (and there are no other
* form types to check), then we can leave without altering the contents of the Inbox
*
```

```

if w_group_access is initial.
*
    exit.
*
endif.

*
* My form owner for form type 'ABCD' is defined as '*', so the standard Inbox won't
* contain any forms for that user and so I don't have to go deleting anything from the
* existing list. What I have to do now is to add in the forms that are shared by
* reading table /flm/fpe:
*

w_ccode = /flm/core=>get_default_customer( ).

*
select * from /flm/fpe into w_fpe
        where ccode = w_ccode
        and ftype = 'ABCD'
        and fowner = '*'.    "may have fstatus in this select also.
*

clear w_newentry.
w_newentry-fname          = 'My Form Description'.    "or read from /flm/ftype
w_newentry-fstatus        = 'Approved'.              "or read from /flm/fstatt
w_newentry-fuser          = '*'.
w_newentry-form_id        = w_fpe-id.
w_newentry-ftype          = w_fpe-ftype.
w_newentry-fccode         = w_fpe-ccode.
w_newentry-fver           = w_fpe-fver.
w_newentry-flang          = w_fpe-flang.
w_newentry-fcreated_date  = w_fpe-created_date.
w_newentry-fsaved_date    = w_fpe-change_date.
w_newentry-finitiated_user = w_fpe-finitiator.
w_newentry-form_id_var    = w_fpe-id_var.
*

w_newentry-ftask =
    /flm/core=>get_portal_instruction( im_ccode = w_ccode
                                       im_flang = w_fpe-flang
                                       im_msgnr = w_fpe-ftask_msg
                                       im_fpe   = w_fpe ).

append w_newentry to ch_inbox.

endselect.
*

*-end of user exit-----*

function z_flm_shared_users.
*-----
***"Local Interface:
*   IMPORTING
*       VALUE(REQUESTOR_ID) TYPE SYUNAME
*       VLAUE(FORM_TYPE) TYPE /FLM/FTYPE_CODE
*   EXPORTING
*       VALUE(GROUP_ACCESS) TYPE FLAG
*-----
*
tables: usr05.
clear group_access.
*
* Users can share up to 5 form types in this scenario:
*
select single * from usr05
        where bname = requestor_id
        and ( parid = 'Z_FLM_SHARED_FORM_1' OR
              parid = 'Z_FLM_SHARED_FORM_2' OR
              parid = 'Z_FLM_SHARED_FORM_3' OR
              parid = 'Z_FLM_SHARED_FORM_4' OR
              parid = 'Z_FLM_SHARED_FORM_5' )
        and parva = form_type.
*
if sy-subrc is initial.
*
    group_access = 'X'.
*
endif.
*
endfunction.

```

---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)

# Render Options

09/12/09 17:00:27

[Subscribe](#) to receive email when this article changes.

This page describes the functioning of the 'Render options' popup in the routing table in FLM configuration:

The screenshot shows a 'Render Options' dialog box with a title bar and a close button. Inside, there's a 'Form Type and Status' section with buttons for 'ACL', 'CORR', and 'I'. Below that is an 'Options' section with various settings:

- Render Type:** A dropdown menu set to 'Static for Annotations'.
- Form Tagging:** A dropdown menu set to 'Untagged form'.
- Initial View:** A dropdown menu set to 'Default'.
- Initial Layout:** A dropdown menu set to 'Single'.
- Initial Magnification:** A dropdown menu set to 'FitWidth'.
- Start Page:** A text input field containing '1'.
- Checkboxes:**
  - ☐ Resize Window Allowed
  - ☐ Center Window Allowed
  - ☐ Full Screen
  - ☐ Display Filename in Title
  - ☒ Hide Window Menus
  - ☒ Hide Toolbars
  - ☒ Hide Window Container

At the bottom, there is a 'Save and Return' button.

Render type => dynamic or static. Dynamic can change template in response to user actions (eg add row), static allows the markup options to be set (sticky notes etc). This is a PDF limitation not an FLM limitation that you can't have both

Form tagging => adds metadata to allow screen reader programs to function with form. Makes it slightly larger and therefore slower

Initial view=> determines which sidebar options are initially open (eg attachments)

Initial magnification=> overwrites the PDF default

%=> linked to above if you choose a certain numeric percentage

Start Page=> where on the document to initially display

Resize window allowed=> ADS option with no effect implemented in Reader as yet

Centre window allowed=> ADS option with no effect implemented in Reader as yet

Full screen=>maxims the PDF on the screen, typically hides useful toolbars, so not recommended

Display filename in Title=> ADS option with no effect implemented in Reader as yet

Hide window menus=> hides the sidebar options (eg attachments, security, bookmarks). Can be restored by pressing F4 in reader

Hide toolbars=>hides standard Adobe toolbars (can be restored by pressing F8 in reader)

Hide windows container=> ADS option with no effect implemented in Reader as yet

---

*Home:* [FLM Documentation](#)

*What's new:* [Recently changed articles](#)