



FLM Developer Guide

Version History

Version	Date	Reason for update
1	10/10/2007	Initial release of the document
2	7/7/2008	Combining Starter Guide, Dev Guide, and Advanced Functions

Table of Contents

1	First Form	4
1.1	Creating the Data Definition of a Form in SAPGUI	
1.2	Designing the layout in Adobe Designer	
1.3	Adding Business Logic	
1.4	Posting Data	
1.5	Form Routing	
1.6	Launching the form in the portal	
2	Form User-exits	11
2.1	Form-level pre-population	
2.2	E-mail address derivation	
2.3	Workflow (FLM Routing Server)	
2.4	Version	
2.5	Language	
2.6	Indexing	
2.7	Enque / Deque	
3	Field User-exits	15
3.1	Field-level prepopulation	
3.2	Possible Entries	
3.3	Derivation	
3.4	Substitutions	
3.5	Validation	
4	Form Posting Engine (FPE)	19
4.1	Processing Control	
4.2	Valid FPE Status	
4.3	FPE Function Control	
4.4	Invoking FPE	
4.5	Posting adapter coding	
5	FLM Routing Server	27
5.1	Using the Routing Tables for Form Submission	
5.2	FLM Routing Server for triggering Off-line Forms	
5.3	FLM Routing Server for Form Escalation	
5.4	FLM Routing Server for Reminder E-mails	
6	Output Forms	33
6.1	Output forms triggered by SAP application output	
6.2	Output forms triggered for form distribution list	
6.3	HR output forms	
6.4	FI output forms	
7	Form Structure	35
7.1	Data hierarchy	
7.2	Subform definition and binding	
7.3	Subform look and feel hints and tips	
8	Index of methods for form data handling	39
8.1	Get the complete address details from an address number	
8.2	Get the complete address details from a partner number	
8.3	Get e-mail address from partner number	
8.4	Get address from address number into single text field	
8.5	Get standard text into single text field	
8.6	Prepopulate field within a subform	
8.7	Add parent paths to form data xml table	
8.8	Get HR Personnel number from user id	
8.9	Get User ID from HR Personnel number	
8.10	Get E-mail address from user id	
8.11	Get E-mail address from HR Personnel number	
8.12	Navigate HR organisational structure	
8.13	Get previous form owner	
8.14	Get previous form actioner	

8.15	Get form name	
8.16	Get form current owner	
8.17	Get form current status	
9	Javascript Samples	44
9.1	Add a row [Javascript]	
9.2	Remove a row [Javascript]	
9.3	Remove a specific row [Javascript]	
9.4	FLM 'Submit' button [Javascript]	
9.5	FLM 'Check' button [Javascript]	
9.6	Total fields within a subform [Javascript]	
9.7	Total the same field for a repeating subform [Javascript]	
9.8	Count the number of rows in a subform	
9.9	Hide a field if there is no data within it	
9.10	Fill the fields of one drop-down list depending on the value selected of another drop-down list	
9.11	Lock Down Elements on a form	
9.12	Calculate Difference between Two Dates	
9.13	Validate a Date.....	
9.14	Set Focus on a Field	
9.15	Use Document Variables	

1 First Form

This section is aimed towards a beginner designing his first form. Although it is helpful for you to be familiar with SAP and Adobe Designer, it should be possible for anyone to follow the step-by-step instructions given here and create your first FLM form and get a feel for the processes involved.

The aims of this exercise are to:

- Create a new form data definition using the FLM form wizard
- Combine the data definition with the form template to create a working Interactive Form
- Enter the required UserExit code to prepopulate, create a dropdown menu, and validate fields on the form
- Launch the finished form in the FLM portal and test it.

Exercise Prerequisites:

- FLM installed on the system
- Customer code initialised
- Form Status, Action and Category codes created.
- Message 996 set up
- Message 'Form submitted for approval' set up e.g. as number 100
- Users 1 and 2 configured

1.1 Creating the Data Definition of a Form in SAPGUI

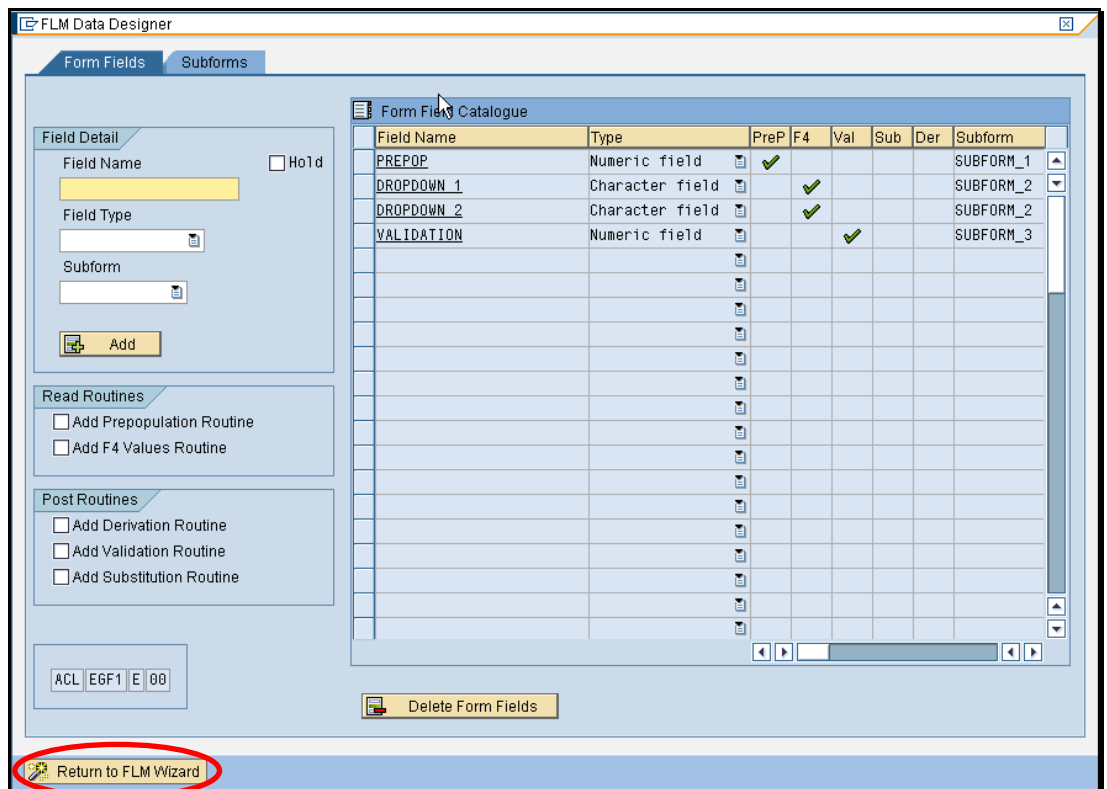
The form data definition is done in the SAP IMG.

Go to Cross-Application Components> General Application Functions> Forms Lifecycle Manager> Interactive Forms> Setup Forms> Form Wizard.


The Form Wizard will open:

1. Click Continue.
2. Enter a 4-character form character code (e.g. TEST),
Leave version number as 00, language and form description and Continue.
3. Launch the Form Data Designer.
4. Click on the 'Subforms' tab
 - a. Begin by creating a subform in the region shown. Call it 'subform_1' with a MaxOccurs of 1 with ROOT as the parent. Click 'add subform' and it will appear in the table on the right.
[MaxOccurs facility allows you to set the maximum number of times items on a subform can be repeated using the + and - buttons on the form.](#)
 - b. Repeat this process for two more subforms: subform_2 with MaxOccurs: 10 and Parent: ROOT and subform_3 with MaxOccurs: 1 and Parent: ROOT

5. Now go to the Form Fields Tab.
 - a. The first field, for the sake of this example, will be called 'prepop', of type 'numc' and parent 'subform_1'. Check the tickbox to 'add prepopulation routine'. Finally click 'Add' and it will appear in the field table.
 - b. Next, create two more fields, called 'Dropdown_1' and 'Dropdown_2', type: 'CHAR' and with parent: 'Subform_2'. Check the box to add an F4 values routine to both.
 - c. Create a fourth field, called 'validation', type = NUMC and parent = subform_3
 - d. Finally, Return to FLM Wizard.



Field Name	Type	PreP	F4	Val	Sub	Der	Subform
PREPOP	Numeric field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			SUBFORM_1
DROPDOWN_1	Character field	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			SUBFORM_2
DROPDOWN_2	Character field	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			SUBFORM_2
VALIDATION	Numeric field	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			SUBFORM_3

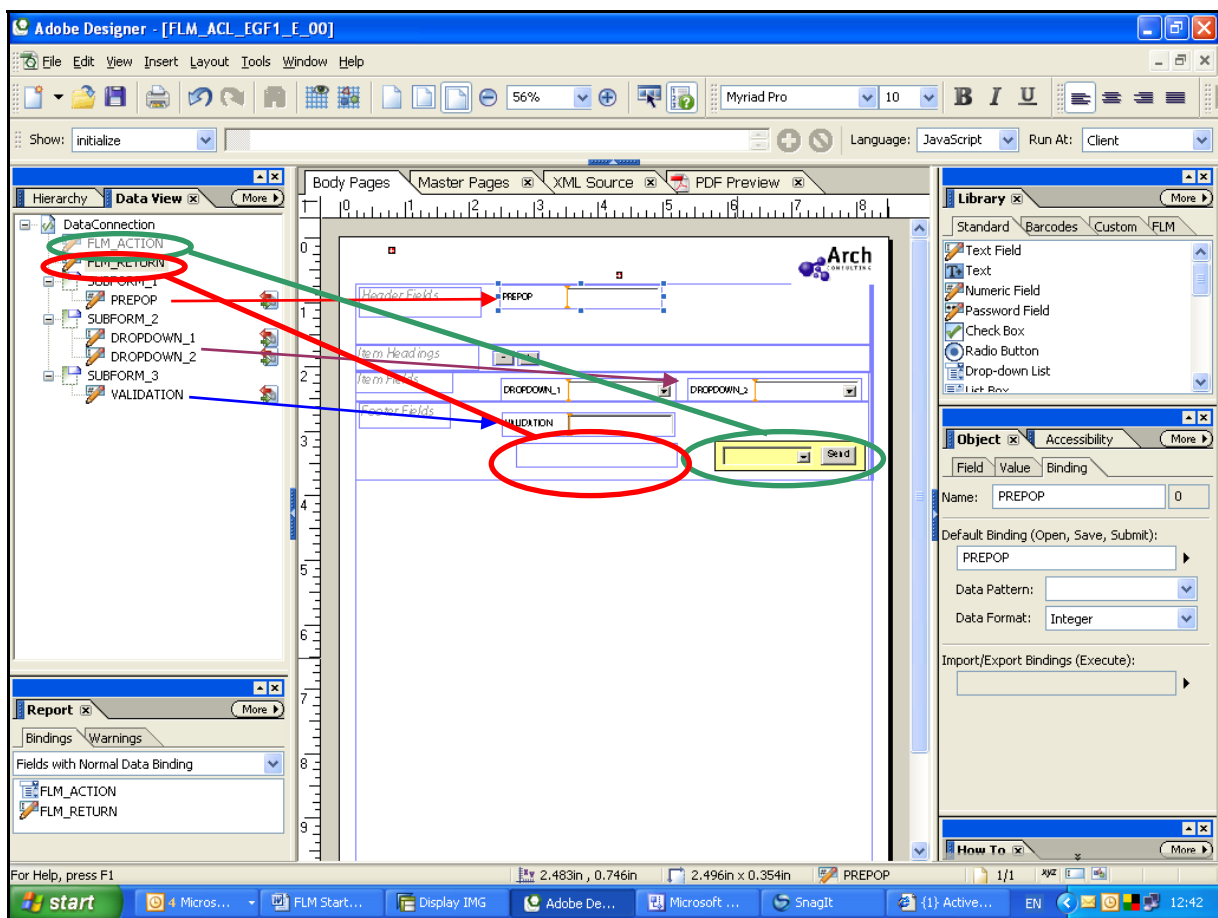
6. FLM automatically creates a directory on your computer in which to store the finished form file. Continue.
7. We will be using the standard FLM template for this exercise, so Continue.
8. Check Online transport and select a Category from the dropdown and Continue.
9. Continue to begin form code generation.
10. Click the  to complete the process or create a new customizing request. E.g. "John's First Form"

1.2 Designing the layout in Adobe Designer

Once the form's data definition has been created, open your form template in Adobe Designer. It will be in the C:/FLM/Form Templates folder and have the format: FLM_[3-digit customer code]_[4-digit form code]_E_[version number]. E.g. FLM_ZZZ_TEST_E_00

1. Go to File> New Data Connection
2. Select 'get data connection from XML schema' and Next.
3. Open the directory and select the FLM form. It will have a file name of FLM_FORM_DATA_[3-digit customer code]_[4-digit form code]_E_[version number]
4. Check 'embed XML schema' and Finish.

The Data View tab now contains the schema you defined in the FLM wizard. Subform_1 will correspond to the 'Header' field on the template, Subform_2 corresponds to 'Items' and Subform_3 corresponds to 'Footer'. 'FLM_Action' and 'FLM_Return' are required by the system so are generated automatically.




5. The next step is to drag-and-drop the fields in the Data View onto the corresponding form fields. Do not drag-and-drop the subforms.
 - a. Drag 'Prepop' onto the subform marked 'Header Fields'
 - b. Drag 'Dropdown_1' and 'Dropdown_2' onto the 'Item Fields' subform
 - c. Drag 'Validation' onto the 'Footer Fields' subform

When the fields are successfully bound, the  icon will appear next to that item in the Data View.

6. FLM_ACTION should be dragged and dropped onto the dropdown field next to the 'Send' button. FLM_RETURN corresponds to the blank box on the left of the ACTION field as shown.

7. You will need to create a drop-down chevron for both the dropdown fields on the form. To do this, select the required fields and click on the field type on the right-hand side. Select 'drop down list' from the options.
8. Click in the text boxes denoting the field names. Rename 'Prepop' to 'Prepopulated' and replace the underscores in the dropdown field names with spaces.

The names of the fields in the data view and in the FLM system do not change when you change the field descriptions on the form, so your field descriptions are not limited by the character and length restrictions in the data designer. It is advisable to keep the data designer names and field descriptions similar or have a system for field nomenclature when working with long forms.

9. In order to ensure that the form functions correctly, you will need to bind the subforms on the template to the subforms you created in the Form Wizard. To do this, return to the Hierarchy tab and under 'DataFlow', select the Header field. In the Object> Binding menu on the right, select the default binding to SUBFORM_1 by clicking on the  under the default binding and going to DataConnection> SUBFORM_1 as shown.
10. Repeat the process for the 'Items' and 'Footer' subforms.
11. Save the form and exit Adobe Designer.

1.3 Adding Business Logic

After working on the form in Adobe Designer we need to upload it back into FLM so we can add the business logic.

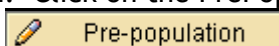
Return to SAP transaction SPRO IMG>Cross Application Components> General Application Functions> Forms Lifecycle Manager and go to 'Upload Form Template'

Select the .XDP file you have just been working with. Click the 'Execute' icon on the top-left of the page. Confirm the following prompts with Yes.

Now that the form is back in we do the following.

1. Go into Field UserExits under business logic from the IMG menu.
2. You will receive a prompt to enter the form type code you are working with. Enter 'TEST' and click 'Continue'
3. The screen shows a list of the fields that were assigned UserExits in the data designer, by category in the Form Wizard. To summarise:

'Dropdown_1' and 'Dropdown_2' were assigned an F4 values routine
 'Prepop' was assigned a prepopulation routine
 'Validation' was assigned a validation routine.

To edit the UserExit code Prepopulates the form. Click on the PrePopulation tab, select the required field and then click the  button.

Enter the following code into the space denoted by

<<<< *Start of Customer code* >>>> and <<<< *End of Customer Code* >>>>
as before:

ex_value = '12345'.

This code will prepopulate the field with the value '12,345'. As with the F4 code, any ABAP code can be entered here to prepopulate the field.

Click Save  and then confirm.

Activate the code using the  tool as before and then exit .

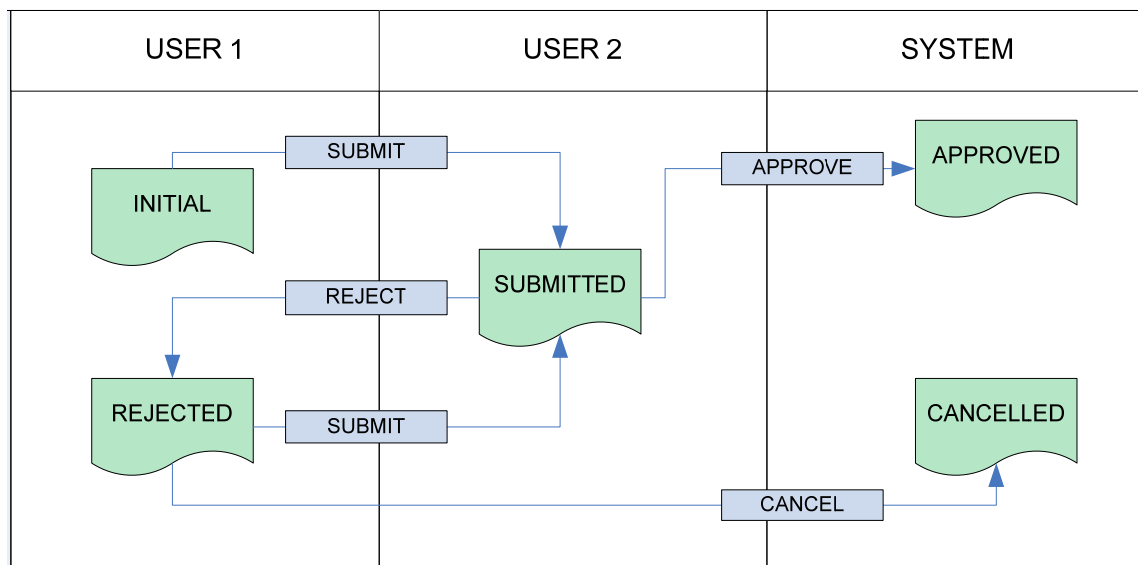
- Follow the same procedure for the Dropdown boxes and for Validation. The UserExits contain code you can uncomment. The Form will still run even if the UserExits are left blank.

1.4 Posting Data

Needs filling in.

1.5 Form Routing

Here we will setup a simple routing for the form:



This routing can be represented by a table of actions:

User	Form Status	Action	New User	New Status
Initial user (User 1)	Initial	Submit	Approver (User 2)	Submitted

Approver (User2)	Submitted	Approve	System	Approved
Approver (User 2)	Submitted	Reject	Initial user (User 1)	Rejected
Initial user (User 1)	Rejected	Submit	Approver (User 2)	Submitted
Initial user (User 1)	Rejected	Cancel	System	Cancelled

To configure this workflow into FLM, first go into Interactive Forms > Setup Form Routing > Form Status Determination from the IMG menu

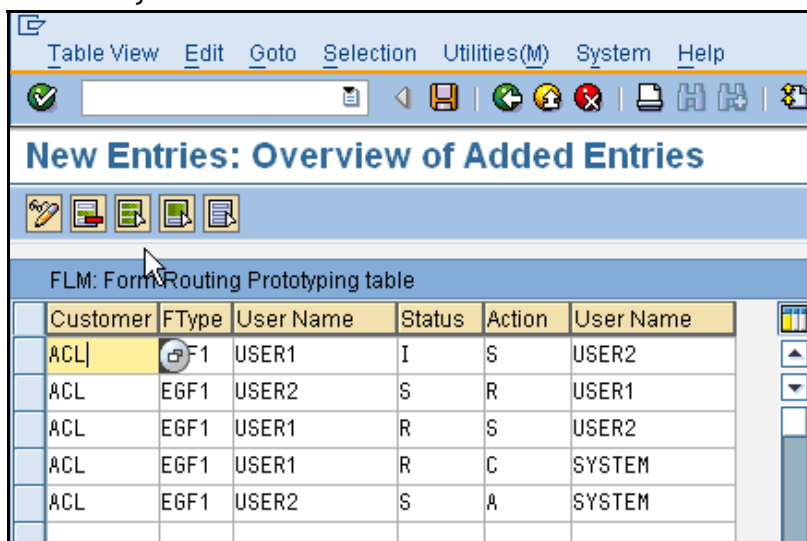
1. Select the name of the form you are working with from the list and double-click on 'Workflow Status' at the top-left of the screen.
2. From the second, third and fifth columns of the routing table above we can derive the following other actions:

Submitted → Approve = End of life
 Submitted → Reject = Rejected
 Rejected → Submit = Submitted
 Rejected → Cancel = Cancelled

Go to 'New Entries' and enter the above status processes into the table as shown below. When you have finished click the 'Save' and Save the data into the correct Customizing Request for that form.

This table can be used for more advanced routing configurations, e.g. email reminders and version variations, so it has a number of options that we won't use for the sake of this example.

3. Next, go into 'Form Owner Determination' from the IMG menu.
4. 'Go into 'New Entries' to configure the form user routing by entering the information as follows substituting in the current Customer Code, Ftype, User, etc... for your form. Click the Save icon and attach it to the correct request.



Customer	FType	User Name	Status	Action	User Name
ACL	E6F1	USER1	I	S	USER2
ACL	E6F1	USER2	S	R	USER1
ACL	E6F1	USER1	R	S	USER2
ACL	E6F1	USER1	R	C	SYSTEM
ACL	E6F1	USER2	S	A	SYSTEM

SYSTEM' is not a configured user, however it is used here to transfer ownership of the form from the user who submitted it and hence remove it from their



inbox. The form will be stored in the system without an 'owner', ready for any subsequent actions, such as data posting, to be applied to it.

1.6 Launching the form in the portal

Launch the FLM portal in a browser using the URL:

<http://yourserver:50100/webdynpro/dispatcher/flm.com/flmgui/flmportal>

The port for your server may be different. (e.g. 50300).

5. Login as the first user in the routing (USER1).
6. Go to Tasks and then New Form
7. Click on the form category 'TEST' to which the form was assigned, scroll down through the list using the  button to the form you created and click on it. It will launch in the browser.
8. When the form launches, the 'Prepopulated' field should contain the value '12,345', as determined by the UserExit code you entered
9. Select 'Submit' and click 'Send' to send the form to the next user in the routing.
10. The form will be submitted, and a confirmation message will appear on the screen
11. If you have the login details of the next user in the routing, you can check that the form was routed successfully to them, as specified in the routing table. To do this, first , then enter the login details of the next user
12. Go to Tasks> Inbox and the form you have just submitted should appear in the listing, with status 'Submitted', created by USER1 with today's date on it. Click on the form to open it.
13. You can decide whether to approve or reject the form. Rejecting it will return it to User 1; approving it will change its status to 'Approved', which can then be input as a valid status for posting to SAP by the Forms Posting Engine.

Congratulations! You have just successfully created and tested your first FLM form. For a more complete example see the FLM Forms Tutorial.

2 Form User-exits

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.

2.1 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_row(3) TYPE n,
           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.

```

2.2 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.

2.2.1 Offline form triggered by FLM Output

The email user-exit described above is always used to determine the e-mail recipient for an email form.

2.2.2 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.

2.2.3 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.

2.3 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.

2.4 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.

2.5 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`.

2.6 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.

2.7 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.

3 Field User-exits

Forms Lifecycle Manager> Interactive Forms> Business Logic> Field User-Exits

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

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 Possible entries

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

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

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

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.
```

4 Form Posting Engine (FPE)

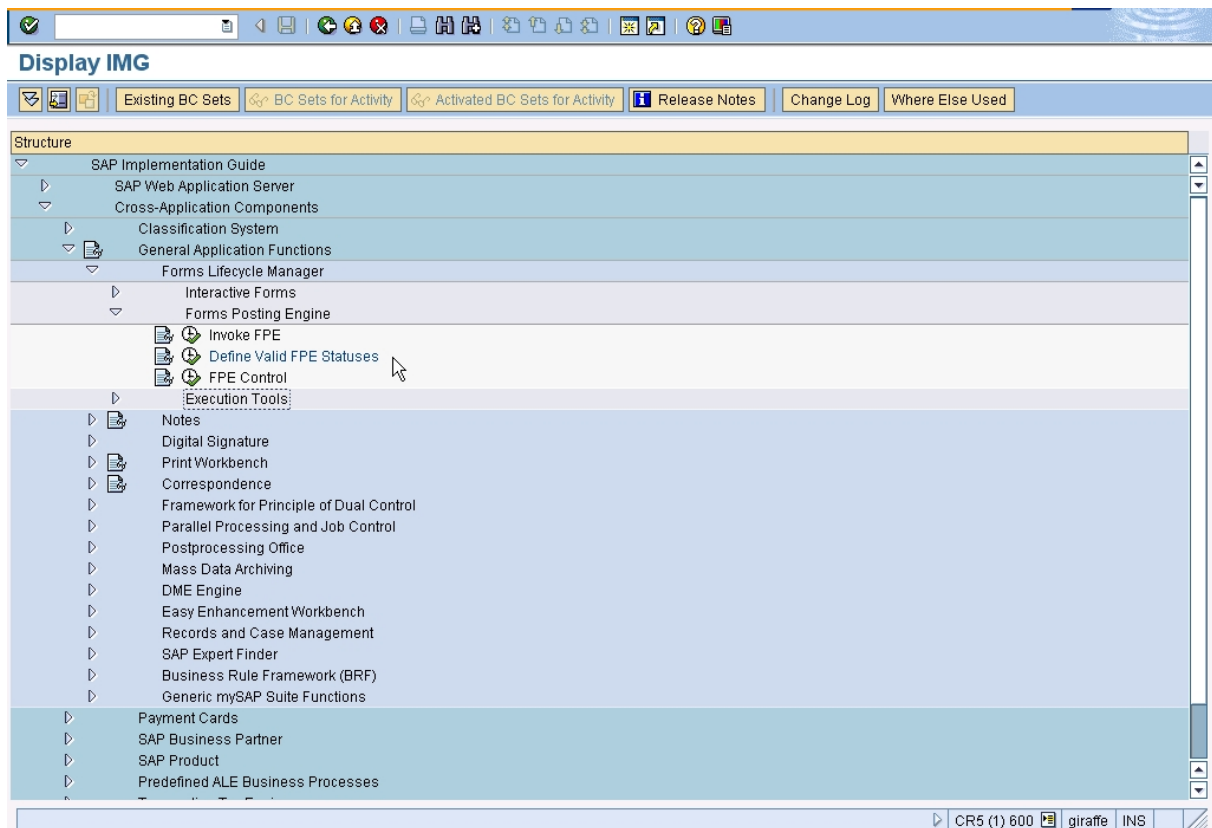
The Forms Posting Engine [FPE] is the component of FLM that controls the posting of data captured on an interactive form into an SAP back end. FPE provides the framework to control when a form is applicable for posting. FPE also provides the facility to handle errors in posting and to repost the data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

4.1 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:

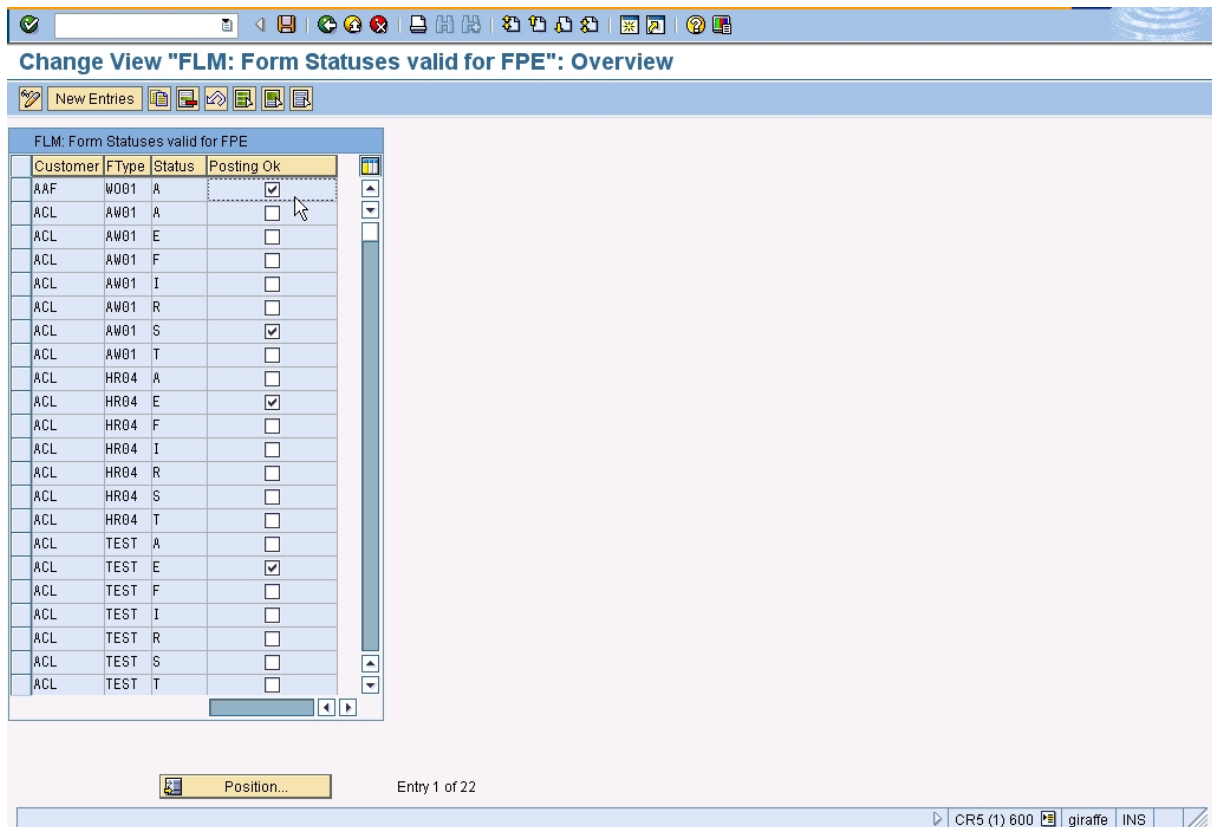


The first table defines what Forms status is applicable for FPE posting. In other words at what stage in the Forms life do we want to persist some data from it into an SAP back end.

The other table controls what function module will be invoked to extract data from the Forms and post it into SAP.

4.2 Valid FPE status.

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



Customer	FType	Status	Posting Ok
AAF	W001	A	<input checked="" type="checkbox"/>
ACL	AW01	A	<input type="checkbox"/>
ACL	AW01	E	<input type="checkbox"/>
ACL	AW01	F	<input type="checkbox"/>
ACL	AW01	I	<input type="checkbox"/>
ACL	AW01	R	<input type="checkbox"/>
ACL	AW01	S	<input checked="" type="checkbox"/>
ACL	AW01	T	<input type="checkbox"/>
ACL	HR04	A	<input type="checkbox"/>
ACL	HR04	E	<input checked="" type="checkbox"/>
ACL	HR04	F	<input type="checkbox"/>
ACL	HR04	I	<input type="checkbox"/>
ACL	HR04	R	<input type="checkbox"/>
ACL	HR04	S	<input type="checkbox"/>
ACL	HR04	T	<input type="checkbox"/>
ACL	TEST	A	<input type="checkbox"/>
ACL	TEST	E	<input checked="" type="checkbox"/>
ACL	TEST	F	<input type="checkbox"/>
ACL	TEST	I	<input type="checkbox"/>
ACL	TEST	R	<input type="checkbox"/>
ACL	TEST	S	<input type="checkbox"/>
ACL	TEST	T	<input type="checkbox"/>

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

4.3 FPE function control

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

[illegible]

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
- YYYYY is the 4 character form ID the posting module relates to
- VVV is the minimum SAP version required eg. 700

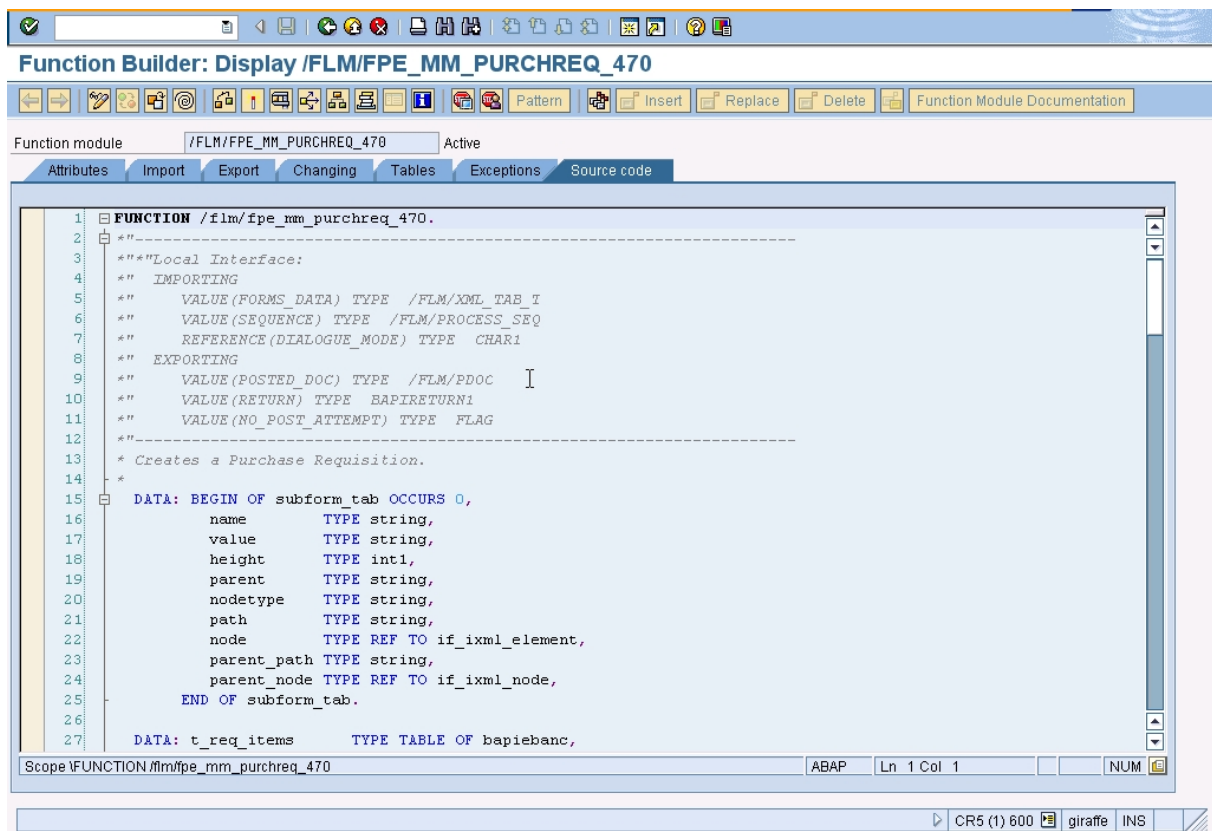
The FPE framework checks the SAP version is at the minimum level required to run the posting function so the last 3 characters of the function module must be the SAP version of the machine that will process the form.

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.

4.4 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:



```

1 FUNCTION /flm/fpe_mm_purchreq_470.
2
3 *-- *-- Local Interface:
4 *-- IMPORTING
5 *--   VALUE(FORMS_DATA) TYPE /FLM/XML_TAB_I
6 *--   VALUE(SEQUENCE) TYPE /FLM/PROCESS_SEQ
7 *--   REFERENCE(DIALOGUE_MODE) TYPE CHAR1
8 *-- EXPORTING
9 *--   VALUE(POSTED_DOC) TYPE /FLM/PDOC
10 *--   VALUE(RETURN) TYPE BAPIRETURN1
11 *--   VALUE(NO_POST_ATTEMPT) TYPE FLAG
12 *-- *--
13 *-- *-- Creates a Purchase Requisition.
14 *-- *--
15 DATA: BEGIN OF subform_tab OCCURS 0,
16         name      TYPE string,
17         value      TYPE string,
18         height     TYPE int1,
19         parent     TYPE string,
20         nodetype   TYPE string,
21         path       TYPE string,
22         node       TYPE REF TO if_ixml_element,
23         parent_path TYPE string,
24         parent_node TYPE REF TO if_ixml_node,
25       END OF subform_tab.
26
27 DATA: t_req_items TYPE TABLE OF bapiebanc,

```

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.

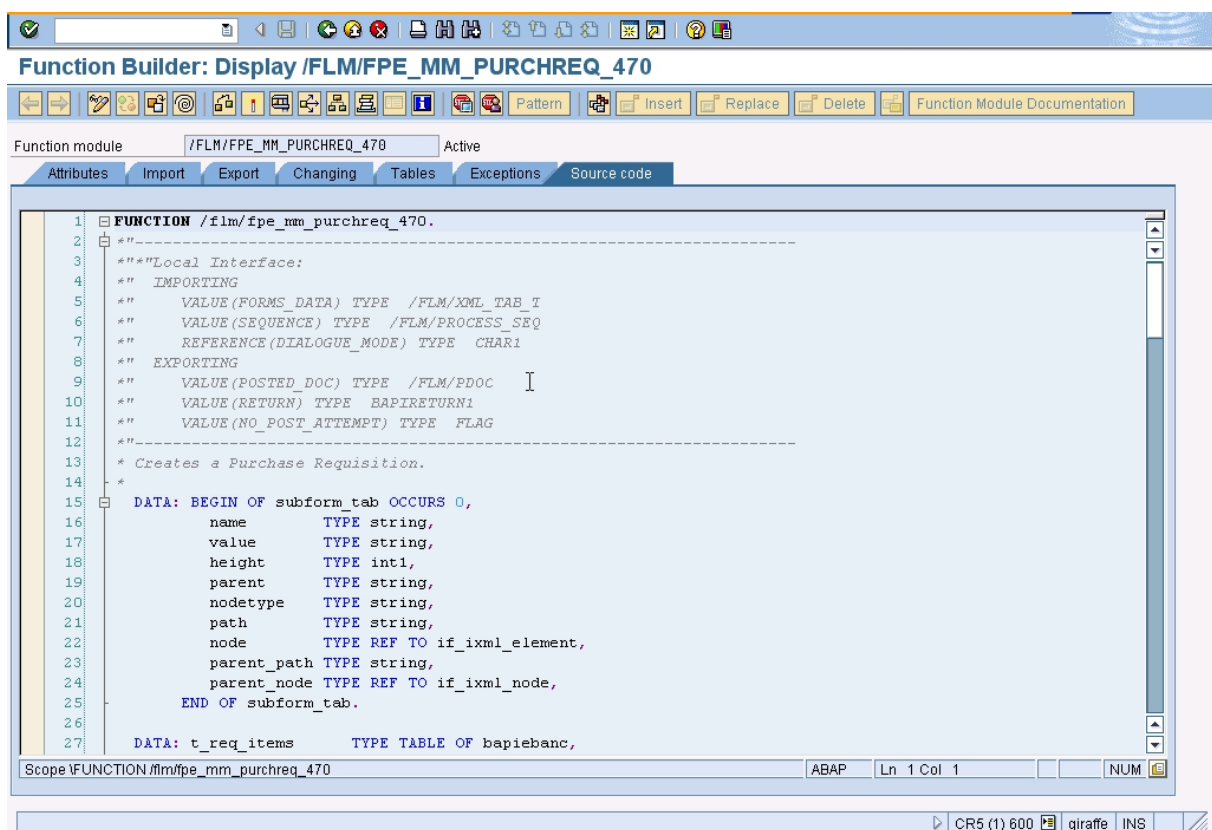
- No_post_attempt - indicates to the FPE framework that no attempt to post was made - so neither success or failure.

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

4.5 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:



```

1 FUNCTION /flm/fpe_mm_purchreq_470.
2
3 *--"Local Interface:
4 *--
5 *-- IMPORTING
6 *--   VALUE(FORMS_DATA) TYPE /FLM/XML_TAB_I
7 *--   VALUE(SEQUENCE) TYPE /FLM/PROCESS_SEQ
8 *--   REFERENCE(DIALOGUE_MODE) TYPE CHAR1
9 *-- EXPORTING
10 *--   VALUE(POSTED_DOC) TYPE /FLM/PDOC
11 *--   VALUE(RETURN) TYPE BAPIRETURN1
12 *--   VALUE(NO_POST_ATTEMPT) TYPE FLAG
13 *--
14 *--
15 *-- * Creates a Purchase Requisition.
16 *--
17 *--
18 *--
19 *--
20 *--
21 *--
22 *--
23 *--
24 *--
25 *--
26 *--
27 *--
28 *--
29 *--
30 *--
31 *--
32 *--
33 *--
34 *--
35 *--
36 *--
37 *--
38 *--
39 *--
40 *--
41 *--
42 *--
43 *--
44 *--
45 *--
46 *--
47 *--
48 *--
49 *--
50 *--
51 *--
52 *--
53 *--
54 *--
55 *--
56 *--
57 *--
58 *--
59 *--
60 *--
61 *--
62 *--
63 *--
64 *--
65 *--
66 *--
67 *--
68 *--
69 *--
70 *--
71 *--
72 *--
73 *--
74 *--
75 *--
76 *--
77 *--
78 *--
79 *--
80 *--
81 *--
82 *--
83 *--
84 *--
85 *--
86 *--
87 *--
88 *--
89 *--
90 *--
91 *--
92 *--
93 *--
94 *--
95 *--
96 *--
97 *--
98 *--
99 *--
100 *--
101 *--
102 *--
103 *--
104 *--
105 *--
106 *--
107 *--
108 *--
109 *--
110 *--
111 *--
112 *--
113 *--
114 *--
115 *--
116 *--
117 *--
118 *--
119 *--
120 *--
121 *--
122 *--
123 *--
124 *--
125 *--
126 *--
127 *--
128 *--
129 *--
130 *--
131 *--
132 *--
133 *--
134 *--
135 *--
136 *--
137 *--
138 *--
139 *--
140 *--
141 *--
142 *--
143 *--
144 *--
145 *--
146 *--
147 *--
148 *--
149 *--
150 *--
151 *--
152 *--
153 *--
154 *--
155 *--
156 *--
157 *--
158 *--
159 *--
160 *--
161 *--
162 *--
163 *--
164 *--
165 *--
166 *--
167 *--
168 *--
169 *--
170 *--
171 *--
172 *--
173 *--
174 *--
175 *--
176 *--
177 *--
178 *--
179 *--
180 *--
181 *--
182 *--
183 *--
184 *--
185 *--
186 *--
187 *--
188 *--
189 *--
190 *--
191 *--
192 *--
193 *--
194 *--
195 *--
196 *--
197 *--
198 *--
199 *--
200 *--
201 *--
202 *--
203 *--
204 *--
205 *--
206 *--
207 *--
208 *--
209 *--
210 *--
211 *--
212 *--
213 *--
214 *--
215 *--
216 *--
217 *--
218 *--
219 *--
220 *--
221 *--
222 *--
223 *--
224 *--
225 *--
226 *--
227 *--
228 *--
229 *--
230 *--
231 *--
232 *--
233 *--
234 *--
235 *--
236 *--
237 *--
238 *--
239 *--
240 *--
241 *--
242 *--
243 *--
244 *--
245 *--
246 *--
247 *--
248 *--
249 *--
250 *--
251 *--
252 *--
253 *--
254 *--
255 *--
256 *--
257 *--
258 *--
259 *--
260 *--
261 *--
262 *--
263 *--
264 *--
265 *--
266 *--
267 *--
268 *--
269 *--
270 *--
271 *--
272 *--
273 *--
274 *--
275 *--
276 *--
277 *--
278 *--
279 *--
280 *--
281 *--
282 *--
283 *--
284 *--
285 *--
286 *--
287 *--
288 *--
289 *--
290 *--
291 *--
292 *--
293 *--
294 *--
295 *--
296 *--
297 *--
298 *--
299 *--
300 *--
301 *--
302 *--
303 *--
304 *--
305 *--
306 *--
307 *--
308 *--
309 *--
310 *--
311 *--
312 *--
313 *--
314 *--
315 *--
316 *--
317 *--
318 *--
319 *--
320 *--
321 *--
322 *--
323 *--
324 *--
325 *--
326 *--
327 *--
328 *--
329 *--
330 *--
331 *--
332 *--
333 *--
334 *--
335 *--
336 *--
337 *--
338 *--
339 *--
340 *--
341 *--
342 *--
343 *--
344 *--
345 *--
346 *--
347 *--
348 *--
349 *--
350 *--
351 *--
352 *--
353 *--
354 *--
355 *--
356 *--
357 *--
358 *--
359 *--
360 *--
361 *--
362 *--
363 *--
364 *--
365 *--
366 *--
367 *--
368 *--
369 *--
370 *--
371 *--
372 *--
373 *--
374 *--
375 *--
376 *--
377 *--
378 *--
379 *--
380 *--
381 *--
382 *--
383 *--
384 *--
385 *--
386 *--
387 *--
388 *--
389 *--
390 *--
391 *--
392 *--
393 *--
394 *--
395 *--
396 *--
397 *--
398 *--
399 *--
400 *--
401 *--
402 *--
403 *--
404 *--
405 *--
406 *--
407 *--
408 *--
409 *--
410 *--
411 *--
412 *--
413 *--
414 *--
415 *--
416 *--
417 *--
418 *--
419 *--
420 *--
421 *--
422 *--
423 *--
424 *--
425 *--
426 *--
427 *--
428 *--
429 *--
430 *--
431 *--
432 *--
433 *--
434 *--
435 *--
436 *--
437 *--
438 *--
439 *--
440 *--
441 *--
442 *--
443 *--
444 *--
445 *--
446 *--
447 *--
448 *--
449 *--
450 *--
451 *--
452 *--
453 *--
454 *--
455 *--
456 *--
457 *--
458 *--
459 *--
460 *--
461 *--
462 *--
463 *--
464 *--
465 *--
466 *--
467 *--
468 *--
469 *--
470 *--
471 *--
472 *--
473 *--
474 *--
475 *--
476 *--
477 *--
478 *--
479 *--
480 *--
481 *--
482 *--
483 *--
484 *--
485 *--
486 *--
487 *--
488 *--
489 *--
490 *--
491 *--
492 *--
493 *--
494 *--
495 *--
496 *--
497 *--
498 *--
499 *--
500 *--
501 *--
502 *--
503 *--
504 *--
505 *--
506 *--
507 *--
508 *--
509 *--
510 *--
511 *--
512 *--
513 *--
514 *--
515 *--
516 *--
517 *--
518 *--
519 *--
520 *--
521 *--
522 *--
523 *--
524 *--
525 *--
526 *--
527 *--
528 *--
529 *--
530 *--
531 *--
532 *--
533 *--
534 *--
535 *--
536 *--
537 *--
538 *--
539 *--
540 *--
541 *--
542 *--
543 *--
544 *--
545 *--
546 *--
547 *--
548 *--
549 *--
550 *--
551 *--
552 *--
553 *--
554 *--
555 *--
556 *--
557 *--
558 *--
559 *--
560 *--
561 *--
562 *--
563 *--
564 *--
565 *--
566 *--
567 *--
568 *--
569 *--
570 *--
571 *--
572 *--
573 *--
574 *--
575 *--
576 *--
577 *--
578 *--
579 *--
580 *--
581 *--
582 *--
583 *--
584 *--
585 *--
586 *--
587 *--
588 *--
589 *--
590 *--
591 *--
592 *--
593 *--
594 *--
595 *--
596 *--
597 *--
598 *--
599 *--
600 *--
601 *--
602 *--
603 *--
604 *--
605 *--
606 *--
607 *--
608 *--
609 *--
610 *--
611 *--
612 *--
613 *--
614 *--
615 *--
616 *--
617 *--
618 *--
619 *--
620 *--
621 *--
622 *--
623 *--
624 *--
625 *--
626 *--
627 *--
628 *--
629 *--
630 *--
631 *--
632 *--
633 *--
634 *--
635 *--
636 *--
637 *--
638 *--
639 *--
640 *--
641 *--
642 *--
643 *--
644 *--
645 *--
646 *--
647 *--
648 *--
649 *--
650 *--
651 *--
652 *--
653 *--
654 *--
655 *--
656 *--
657 *--
658 *--
659 *--
660 *--
661 *--
662 *--
663 *--
664 *--
665 *--
666 *--
667 *--
668 *--
669 *--
670 *--
671 *--
672 *--
673 *--
674 *--
675 *--
676 *--
677 *--
678 *--
679 *--
680 *--
681 *--
682 *--
683 *--
684 *--
685 *--
686 *--
687 *--
688 *--
689 *--
690 *--
691 *--
692 *--
693 *--
694 *--
695 *--
696 *--
697 *--
698 *--
699 *--
700 *--
701 *--
702 *--
703 *--
704 *--
705 *--
706 *--
707 *--
708 *--
709 *--
710 *--
711 *--
712 *--
713 *--
714 *--
715 *--
716 *--
717 *--
718 *--
719 *--
720 *--
721 *--
722 *--
723 *--
724 *--
725 *--
726 *--
727 *--
728 *--
729 *--
730 *--
731 *--
732 *--
733 *--
734 *--
735 *--
736 *--
737 *--
738 *--
739 *--
740 *--
741 *--
742 *--
743 *--
744 *--
745 *--
746 *--
747 *--
748 *--
749 *--
750 *--
751 *--
752 *--
753 *--
754 *--
755 *--
756 *--
757 *--
758 *--
759 *--
760 *--
761 *--
762 *--
763 *--
764 *--
765 *--
766 *--
767 *--
768 *--
769 *--
770 *--
771 *--
772 *--
773 *--
774 *--
775 *--
776 *--
777 *--
778 *--
779 *--
780 *--
781 *--
782 *--
783 *--
784 *--
785 *--
786 *--
787 *--
788 *--
789 *--
790 *--
791 *--
792 *--
793 *--
794 *--
795 *--
796 *--
797 *--
798 *--
799 *--
800 *--
801 *--
802 *--
803 *--
804 *--
805 *--
806 *--
807 *--
808 *--
809 *--
810 *--
811 *--
812 *--
813 *--
814 *--
815 *--
816 *--
817 *--
818 *--
819 *--
820 *--
821 *--
822 *--
823 *--
824 *--
825 *--
826 *--
827 *--
828 *--
829 *--
830 *--
831 *--
832 *--
833 *--
834 *--
835 *--
836 *--
837 *--
838 *--
839 *--
840 *--
841 *--
842 *--
843 *--
844 *--
845 *--
846 *--
847 *--
848 *--
849 *--
850 *--
851 *--
852 *--
853 *--
854 *--
855 *--
856 *--
857 *--
858 *--
859 *--
860 *--
861 *--
862 *--
863 *--
864 *--
865 *--
866 *--
867 *--
868 *--
869 *--
870 *--
871 *--
872 *--
873 *--
874 *--
875 *--
876 *--
877 *--
878 *--
879 *--
880 *--
881 *--
882 *--
883 *--
884 *--
885 *--
886 *--
887 *--
888 *--
889 *--
890 *--
891 *--
892 *--
893 *--
894 *--
895 *--
896 *--
897 *--
898 *--
899 *--
900 *--
901 *--
902 *--
903 *--
904 *--
905 *--
906 *--
907 *--
908 *--
909 *--
910 *--
911 *--
912 *--
913 *--
914 *--
915 *--
916 *--
917 *--
918 *--
919 *--
920 *--
921 *--
922 *--
923 *--
924 *--
925 *--
926 *--
927 *--
928 *--
929 *--
930 *--
931 *--
932 *--
933 *--
934 *--
935 *--
936 *--
937 *--
938 *--
939 *--
940 *--
941 *--
942 *--
943 *--
944 *--
945 *--
946 *--
947 *--
948 *--
949 *--
950 *--
951 *--
952 *--
953 *--
954 *--
955 *--
956 *--
957 *--
958 *--
959 *--
960 *--
961 *--
962 *--
963 *--
964 *--
965 *--
966 *--
967 *--
968 *--
969 *--
970 *--
971 *--
972 *--
973 *--
974 *--
975 *--
976 *--
977 *--
978 *--
979 *--
980 *--
981 *--
982 *--
983 *--
984 *--
985 *--
986 *--
987 *--
988 *--
989 *--
990 *--
991 *--
992 *--
993 *--
994 *--
995 *--
996 *--
997 *--
998 *--
999 *--
1000 *--

```

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.

4.6 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.
  ENDLLOOP.

  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.

5 FLM Routing Server

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.

5.1 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.

5.1.1 Table /FLM/WE_STAT: Form Status Derivation

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'. See section 1.2*
- E-mail notification settings
 - *There is an e-mail notification flag, plus standard texts for the e-mail subject and e-mail body.*

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.

5.1.2 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.

5.1.3 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.)*

5.1.4 Variable substitution in e-mail standard texts

Variable substitution is available in all three e-mail scenarios (*offline* e-mail, *notification* e-mail and *reminder* e-mail).

Three types of variable can be substituted: FLM fields, system fields and form fields.

FLM fields

FLM fields can be included in each standard text using the format:

'&<field name>&'

In FLM version 2.4, the following fields are available:

- &FID& Unique Form ID
- &FORMTYPE& Description of form type
- &URL& URL link to the form

System fields

System fields can be included in each standard text using the menu path: Include-

>Symbols->System->ABAP System Symbols

In FLM version 2.4, the following fields are supported:

- &SYST-DATUM& System date
- &SYST-UZEIT& System time
- &SYST-UNAME& System user

Form fields

In the scenarios when form data exists, any non-repeating data can be included in the standard texts; the data field must not exist within a repeating subform. So root/header/footer fields can all be included, whereas item fields cannot.

Again the format required is:

'&<field name>&'

Note that if any form field was given the same name as an FLM field then the substitution would work for the FLM field, not the form field.

5.2 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.

5.2.1 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

Customer	FType	Language	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	L6CA	EN	00		ag@arch.co.uk	/FLM/OFFLINE_EMAIL_TITLE	/FLM/OFFLINE_EMAIL_BODY	/FLM/OFFLINE_EMAIL_ATT_NAME

5.3 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 to

Form Type 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.

5.3.1 Table /FLM/WF_ESCA: Form Escalation settings

Change View "FLM: Form Escalation": Overview					
     					
FLM: Form Escalation					
Customer	FType	Status	Esc Days	Action	
ACL	ABIE	P	10	Reject	
ACL	ADBE	P	1	Reject	
ACL	KL10	P	10	Reject	

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.

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 1.1 and 1.2.

5.4 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.

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.

5.4.1 Table /FLM/WF_REMI: Form Reminder settings

Change View "FLM: E-mail Reminders": Overview						
 New Entries      						
FLM: E-mail Reminders						
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	

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 1.1.4. 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.

6 Output Forms

6.1 Output forms triggered by SAP application output

An interface with name /FLM/xx needs to be defined where xx is the SAP application code (EF = purchasing, V1 = sales order etc.)

The import parameters are always:

Parameter	Assignment	Type name	Optional flag	Pass value
/1BCDWB/DOCPARAMS	TYPE	SFPDOCPARAMS	1	1
NAST	TYPE	NAST	0	1

The export parameters are always:

Parameter	Assignment	Type name	Pass value
/1BCDWB/FORMOUTPUT	TYPE	FPFORMOUTPUT	1

Within the 'Global data' part of the interface we add the structures required to be mapped to form fields.

Within the 'Code Initialization' part of the interface, we add the code to call the function module to fill the structures defined in the global data. We export the 'NAST' table entry and import the data in the structures required to map to the form. For example:

```
CALL FUNCTION '/FLM/FLMO_EF'
  EXPORTING
    nast           = nast
  IMPORTING
    ef_po_print    = ef_po_print
  EXCEPTIONS
    data_error     = 1
    OTHERS         = 2.
```

Note that recipient e-mail addresses are derived from the partner in the condition record (passed on to table NAST)

6.2 Output forms triggered for form distribution list

For output forms to a large distribution list, use the email form scenario with no interactive fields. Use the e-mail user-exit to determine the distribution list.

6.3 HR output forms

PDF output forms are already integrated with HR output. Use transaction HRFORMS to branch to the SAP Form Builder which has Adobe Designer embedded. There is no integration with FLM.

6.4 FI output forms

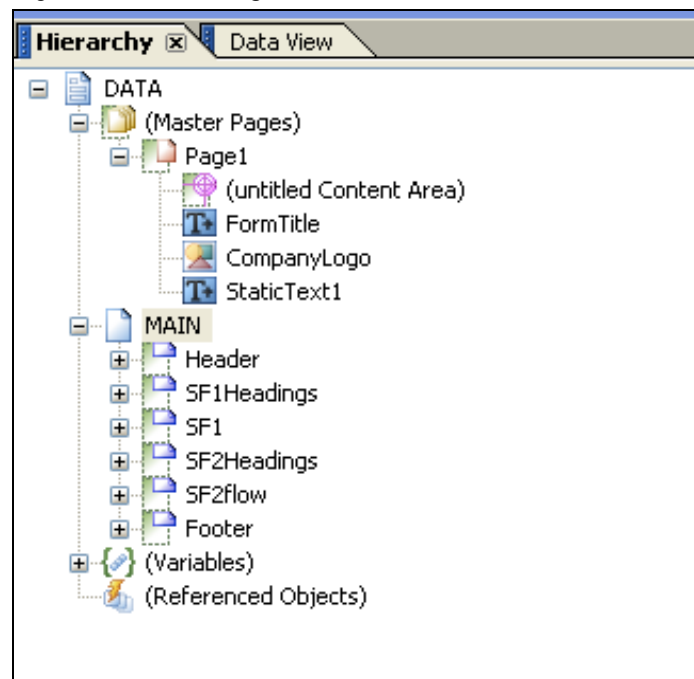
PDF output forms are already integrated with FI correspondance. Link the custom program to the form in table T001F through view V_T001F2. Then correspondance program RFKORI80 will use this table and generate PDF output forms. There is no integration with FLM.

7 Form Structure

7.1 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.

7.2 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.

7.2.1 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:

<code>\$record.Header.form_status</code>
--

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.

7.2.2 Binding for repeating subforms.

It is essential to bind subforms that can repeat.

The binding of the subform takes the following form:

<code>\$record.EmployeeUtil.ResUtilisation[*]</code>
--

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

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 are 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:

<code>\$record.ITEM[*].EBELP</code>

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.

7.2.3 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:

<code>plantdata[*]</code>

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.

7.3 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.

8 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.

8.1 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.

8.2 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.

8.3 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.

8.4 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.

8.5 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 TDOBNAM	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.

8.6 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.

8.7 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.

8.8 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.

8.9 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.

8.10 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.

8.11 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.

8.12 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 described 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.

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

8.13 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.

8.14 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.

8.15 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.

8.16 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.

8.17 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.

9 Javascript Samples

9.1 Add a row [Javascript]

Here 'ITEM' is the subform.
This code will typically sit behind the click event of a pushbutton.
Also we set the presence of a deletion button when there are multiple rows present.

```
_ITEM.addInstance(1)
subtracta.presence = "visible";
```

9.2 Remove a row [Javascript]

Here 'ITEM' is the subform.
Again this code will sit behind the click event of a pushbutton for deletion.
With this code we ensure that there is always at least one instance of the subform; we hide the deletion button when the row count is '1'.

```
var nRowCount_e = _ITEM.count;
if (nRowCount_e == 1)
{
    _ITEM.removeInstance(0);
    this.presence = "invisible";
}
else
{
    _ITEM.removeInstance(nRowCount_e - 1);
}
```

9.3 Remove a specific row [Javascript]

Here 'ITEM' is the subform. We would use this code if we put a pushbutton within the repeating subform.

```
_ITEM.removeInstance(this.parent.index);
```

9.4 FLM 'Submit' button [Javascript]

Here we trigger either the SAP call or the offline e-mail based on the value of the FTRANSPORT variable.

```
soUtils.lockDownForm(xfa.resolveNode("xfa.form"),true);

if (FTRANSPORT.value == "1")
{
    ContainerFoundation_JS.SendMessageToContainer(event.target, "submit",
    "", "", "", "");
}
else
{
    event.target.submitForm({cURL: "mailto:" + RET_EMAIL.value,
    bEmpty: true, // Post all fields (true), or do Not
    post all fields (false)
    cSubmitAs: RET_FILE_TYPE.value});
}
```

9.5 FLM 'Check' button [Javascript]

Here we set the action to 'C'.

```
var sRetVars = CCODE.value + "-" +
               FTYPE.value + "-" +
               FLANG.value + "-" +
               FVER.value + "-" +
               FID.value + "-" +
               FID_VAR.value + "+" +
               REC_EMAIL.value;

var sAction = FLM_ACTION.rawValue;

sRetVars = "C" + "+" + sRetVars;
Button2.access = "open";

FLM_RETURN.rawValue = sRetVars;

if (FTRANSPORT.value == "1")
{
    ContainerFoundation_JS.SendMessageToContainer(event.target, "check",
    "", "", "", "");
}
```

9.6 Total fields within a subform [Javascript]

Since the fields are in the same subform we don't need to define the full path.

```
this.rawValue = (fld_1.rawValue + fld_2.rawValue + fld_3.rawValue);
```

9.7 Total the same field for a repeating subform [Javascript]

Use the following script:

```
//Define our count variable
var iTotalHrs = 0;

//Define your XPATH to the the repeating subform, and store in a Variable
var vNode =
"xfa.form.WorkSheet.Fred_Form.Work_Order.Work_Order_Details.Employee_Details";

//Count how many items we have
//NB This uses the resolveNodes method which will return an object which we
can use to get the length
//This will tell use how many objects we have, NB it is zero based.
var objRootItem = xfa.resolveNodes(vNode + "[*]");

//Check we have at least one subform created.
if (objRootItem.length > 0)
{
    //Now loop through all subforms
    for (var i=0; i<objRootItem.length; i++)
    {
        //Set the current Position
        var sPos = "[" + i + "]"

        //Now Query the child node and gets it value
    }
}
```

```

        iTotalHrs = iTotalHrs + xfa.resolveNode(vNode + sPos +
".TotalHrs").rawValue;

    }

}

//Now set the value of this object to the calculated value
this.rawValue = iTotalHrs;

```

9.8 Count the number of rows in a subform

```

//Define your XPATH to the the repeating subform, and store in a Variable
var vNode =
"xfa.form.WorkSheet.Fred_Form.Work_Order.Work_Order_Details.Employee_Details";

//Count how many items we have
//NB This uses the resolveNodes method which will return an object which we
can use to get the length
//This will tell use how many objects we have, NB it is zero based.
var objRootItem = xfa.resolveNodes(vNode + "[*]");

//Store how many subforms we have, NB this is zero based
var TotalNo = objRootItem.length;

```

9.9 Hide a field if there is no data within it

This will be placed in the initilize event if thi action occurs when the form is loaded.

```

if (this.rawValue == null ||
    this.rawValue == "")
{
    this.presence = "hidden";
}
else
{
    this.presence = "visible";
}

```

If you want the field to be shown or hidden while working on the form then you can use the calculate event.

NB The last line in the Calculate event is very critical otherwise this field will have the value of hidden or visible

```

if (this.rawValue == null ||
    this.rawValue == "")
{
    this.presence = "hidden";
}
else
{
    this.presence = "visible";
}

```

```

//This is needed as by default this script will always assign the
//field the last value on the RHS
this.rawValue = this.rawValue;

```

9.10 Fill the fields of one drop-down list depending on the value selected of another drop-down list.

```
//The Following code shows this example using the Calculate event of a drop
down
//NB That this code uses the default FLM Format for Paired (ID|VALUE) and
Linked
//(ID|VALUE;LINKED_ID) Arrays. Below is an example
//Parent Array will have an ID|Value and elements in the array are
seperated by ;
//001|Category1;002|Category2;003|Category3
//This will produce a drop down with three elements the ID being 001 002
003
//and the Value being Category 1 Category 2 Category 3
//The Child Array with the link to the parent will look like this
//0001|Cat1 Data1;001#0002|Cat1 Data2;001#0003|Cat2 Data1;002#0004|Cat3
Data1;003#0005|Cat3 Data2;003
//Which will produce Child IDs od 0001 0002 0003 0004 0005
//which will have Values of Cat1 Data1 Cat1 Data2 Cat2 Data1 Cat3 Data1
Cat3 Data2
//And will be linked to the Corresponding Parent Item by the Linked ID 001
for the First 2 Items
//Linked ID 002 for the third item and Linked ID 003 for the Last two
//So if a user select the parent object as 002 Category 2
//then the child Drop Down would only show the item 0003 Cat2 Data1

//The below shows the code shows how to do this

//This code needs the Script Object soDropDownComplex
//And we use the method populateLinkedByName

//So we call the method populateLinkedByName
//
//This will first cache the value currently selected
//
//it will then rebuild the drop down using the values
//passed in the second param which hold the cached drop down values
//this is in the format of value;value
//
//Next it will check the cached value and if that is part of the new drop
down
//will set that value back
//
//The First Param is the drop down XPATH, which will be populated
//The Second Param is the hidden field storing the Drop Down values in a
seperated list
//The Last Param will the linked (Parent ID) to populate this list by, so
only items with
//that Linked ID will be added

//First we need to get the LinkedID, so we query the parent Object and its
its item
//in this example we know we display the value and not the ID
var sLinkedName = xfa.form.DATA.MAIN.Header.FR2.rawValue;

//So the next thing we need to do is get the ID for that Item so we use the
method
```

```
//getSimpleElementByID and request the 0 element which is the ID (Element 1
would be the name, and can be used to check
//if a Value exists in the array)
var sLinkedID =
soDropDownComplex.getSimpleElementByName(xfa.form.DATA.MAIN.Header.FR1.somE
xpression,sLinkedName,0);

//Now we call the method populateLinkedByName as described above
var sDropDownValue =
soDropDownComplex.populateLinkedByName(this.somExpression,
FS1_1.somExpression, sLinkedID );

//As we using the calculate event we need to get the value returned
//from the method and set this as the current value, if the current item it
will be ""
//otherwise it would carry over the previous stored value
this.rawValue = sDropDownValue;
```

There are several ways to achieve cascading drop-down lists.

For example, a web-service could be called by a form action in order to fill one drop-down list given the selection in another form field - this would work for on-line forms.

See the attached example template FLM_ACL_CDD1_E_00.xdp for another example using data arrays.

9.11 Lock Down Elements on a form

This will be placed in the initialize event if this action occurs when the form is loaded.

```
if (this.rawValue == null ||
    this.rawValue == "")
{
    this.presence = "hidden";
}
else
{
    this.presence = "visible";
}
```

If you want the field to be shown or hidden while working on the form then you can use the calculate event.
NB The last line in the Calculate event is very critical otherwise this field will have the value of hidden or visible

```
if (this.rawValue == null ||
    this.rawValue == "")
{
    this.presence = "hidden";
}
else
{
    this.presence = "visible";
}
```

//This is needed as by default this script will always assign the
//field the last value on the RHS
this.rawValue = this.rawValue;

9.12 Calculate Difference between Two Dates

```
//This relies on the function dateDiff which should be
//available in the soDate Script Object

//First we need to get our two dates
//NB That getting rawValue will mean the date is in the correct
//format which is YYYY-MM-DD
var dDate1 = xfa.form.DATA.MAIN.Header.FR1.rawValue;
var dDate2 = xfa.form.DATA.MAIN.Header.FR2.rawValue;

//Check to make sure we have two values
if (dDate1 != null && dDate2 != null)
{
    //Now Pass the Dates, the first param is the Start Date
    //The second is the End Date
    var iDiff = soDate.dateDiff(dDate1,dDate2);

    //Now display the Difference
    xfa.host.messageBox("Difference in Dates is = " + iDiff);
}
```

9.13 Validate a Date

```
//This relies on the function dateValid which should be
//available in the soDate Script Object

//Get the rawValue this is needed as the
//internal format for dates is YYYY-MM-DD
var dDate = xfa.form.DATA.MAIN.Header.FR1.rawValue;

//Now call the dateValid Method
if (soDate.dateValid(dDate) == false)
{
    xfa.host.messageBox("date in not correct.");
}
```

9.14 Set Focus on a Field

```
//This code is place in the exit event
//and will set focus back to the same field

//NB the property somExpression will hold the complete XPATH for the field
xfa.host.setFocus(this.somExpression);
```

9.15 Use Document Variables

```
//NB That the Background Colour on the object must be set to Solid for this
to work
//Also based upon the layout of the object you can get different effects

//This is also useful in the Enter and Exit events to show the current
field
//Set to Yellow On Enter
//this.fillColor = COLOUR_YELLOW.value;
//Set to White On Exit
//this.fillColor = COLOUR_WHITE.value;
```

```
//One Last thing to note is that Document Variables will always revert to  
the stored  
//value set at design time when the form is opened or re-opened.  
  
//Now set it colour to red  
xfa.form.DATA.MAIN.Header.FR1.fillColor = COLOUR_RED.value;  
  
xfa.form.DATA.MAIN.Header.FR2.fillColor = COLOUR_RED.value;
```