-

FLM Developer Guide

Author: Chris Scott
File name: FLM 267 - Developer Guide.doc
Version: 1
Distribution:

Version History

Version	Author(s)	Reason for update
1	Chris Scott	Initial release of the document

Table of Contents

1 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.

1.1 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.
```

1.2 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.

1.3 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.

1.4 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.

1.5 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.

1.6 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 intial 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.

1.7 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.

2 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.

2.1 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'.

2.2 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.
```

2.3 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.

2.4 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.

2.5 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.
```

3 Offline Forms

3.1 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.

3.2 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.

3.3 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.

4 Posting Adapters

4.1 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 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 occurance 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.

5 Output Forms

5.1 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)

5.2 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.

5.3 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.

5.4 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.

6 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.

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6 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.

6.7 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.

6.8 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.

6.9 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.

6.10 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.

6.11 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.

6.12 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.

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

6.13 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.

6.14 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.

6.15 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.

6.16 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.

6.17 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

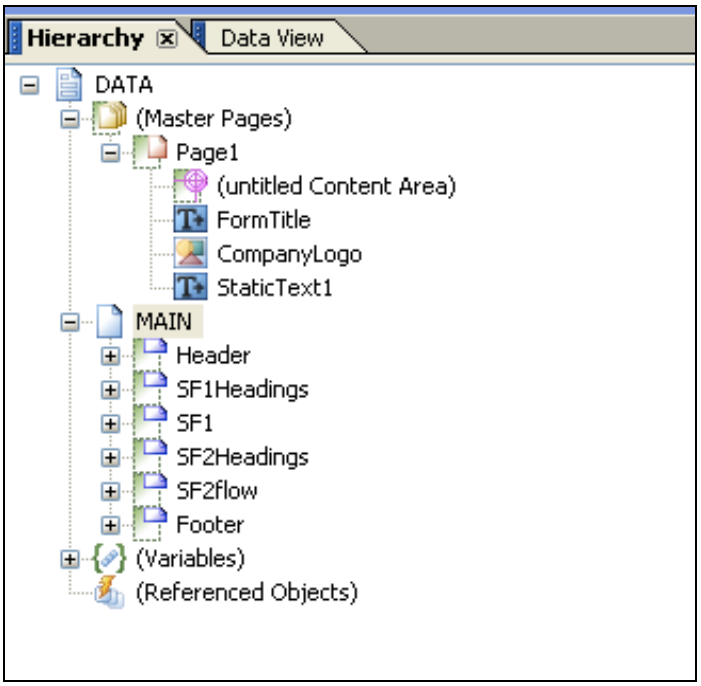
This method returns the current status of a form.

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 occuring 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:

```
$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.

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:

```
$record.EmployeeUtil.ResUtilisation[*]
```

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 pratice 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:

```
$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.

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:

```
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.

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 Javascript Samples

8.1 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";
```

8.2 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);
}
```

8.3 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);
```

8.4 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});
}
```

8.5 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",
"", "", "", "");
}
```

8.6 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);
```

8.7 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_Detail
s";

//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;
```

8.8 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_Detail
s";

//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;
```

8.9 Hide a field if there is no data within it

This will be placed in the initiliaze 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 defualt this script will always assign the
//field the last value on the RHS
this.rawValue = this.rawValue;
```

8.10 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 wiil 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 acheive 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.

8.11 Lock Down Elements on a form

This will be placed in the initiliaze 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 defualt this script will always assign the
//field the last value on the RHS
this.rawValue = this.rawValue;
```

8.12 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);

}
```

8.13 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.");
}
```

8.14 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);
```

8.15 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;
```