04-May-2025 07:15
John Braddock

DECFORMS summary

Intro

A summary of DECFORMS, mainly for app developers.

Key Concepts

The following assist with understanding how DECFORMS works:

1. A form is coded in a file called an IFDL file (Independent Form Description Language).

2. A form must be invoked from a program beginning with a forms$enable. The program exchanges data with the form with send, receive, and transceive responses. The program should do a forms$disable before the program exits.

3. Form Data in DECFORMS persists in memory between calls from the main program.

4. DECFORMS maintains an Activation List which is a list of all fields on the panel that are accessible. The field that currently has focus (via a cursor) is the Current Activation Item.

5. DECFORMS automatically copies data from parameter records to Form records on a forms$send or a forms$transceive based on the names matching as soon as the form is reentered. This is called the distribution phase

6. DECFORMS automatically copies data from Form records to parameter records on a forms$receive or a forms$transceive based on the names matching just before control returns to the program. This is called the collection phase.

7. Data is provided directly to a procedural escape program. There is no need to use Form Records.

8. The form Manager responds to events (such as actions at the terminal) with default responses. These responses can be overridden.

9. DEFORMS provides default key bindings for commonly used keys. These are called built in functions. For example “Enter” causes Form Manager to move the cursor to the next field in th activation list. This is called the NEXT ITEM built in function.

10. The phases of the Form Magager are:
Accept Phase is where the operator interacts with the form. When the operator presses a function key the Form Manager performs a function response. This will be the default respose unless overridden by a user-defined funtion.

11. A WAIT activation item is available to cause the Form Manager to wait for a function key instead of initiating the next item response when the field fills.

12.

Form Manager
This controls communication between the display device, the form, and the program(s).

Example:

```
Layout VT_LAYOUT
    Device
      Terminal
            Type %VT100
    End Device
    Size 24 lines by 80 columns
End Layout
```

Layout
One for each display device. Decforms automatically selects the correct one for the device being used.

Viewport
A rectangular area of the screen used to display items. Coordinates are relative to the upper left of the screen. A screen may have many viewpports. Multiple viewports can be displayed simultaneously and they may overlap.

Panel
A container for the items to be displayed. A panel must be associated with a viewport to be visible. Several panels may be associated with a viewport but only one can be displayed at a time. A panel can occupy different viewports at different times. A panel cannot e displayed in more than one panel sumultaneously.

Items
These may be literals, fields or icons. They can be grouped together, for example a repeat field to list products or customers. A field will have a picture string to describe how it is displayed.

Field
described an item on a panel.
Example:

```
Field MYFIELD
   Line 12
   Column 20
   Output “”
     When (MY_DATE = BLANK_DATE)
   Output Picture 999R
   Justification Right
   Protected
      When (MY_DATE = BLANK_DATE)
End Field
```

Functions
optional association between non-alphanumeric keys and tasks to be performed.
Example:

```
Function CALCULATE_TOTALS
   Is %DO
      (%PF1)
End Function
```

Responses
These are triggered by various events. Such as:
Function Responses triggered by a Function (associated with a keyboard key press
Internal Responses triggered by being Included from another responses
External Response triggered by an external event, eg:
Enable Response triggered by a forms$enable request
Accept Phase responses. Declares at the field level Eg:
Entry Responses
Exit Responses
Validation Response

Statements in a response are called response steps. Examples:

Activate	an item (field, panel)
Call	a program
Deactivate	an item (field or panel)
Display	a panel
Include	another internal response
If	test something
Let	assign a value to a form data item
Message	appears on the message line
Position	sets the current activation item
Remove	a viewport
Return	terminates the accept phase. But response completes
Return Immediate	do not do any more validation

Function Responses
are optional routines to be performed when triggered by a function. For example a response might call a sub program in COBOL. This is called an escape routine. (or a procedural escape).
Examples 1.

```
Function Response CALCULATE_TOTALS
   Include CALC_TOTS
End Response

Function Response MOVE_UP
   If (NOT UPPERMOST ITEM) Then
      Position to Up Item
   Else
      Message “whatever”
   End If
End Response

Function Response Next Help
```

```
    Activate PANEL HLP_PANEL
    Position to HLP_PANEL
End Response
```

Internal Responses
Are invoked from another response with an Include statement

Activation List

Procedural escapes

Event Log
This is useful for tracing activity in the form.

Internal Responses with Procedural Escape
These call a sub program. Parameters are by reference and may be either fields or records in the Form Data. Eg:

```
Internal Response CALC_TOTS
    Message “Computing Totals”
    Call “PGM001_CALC_TOTS” Using
                By Reference FORM_DAT1
                By Reference FORM_DAT2
                By Reference FORM_DAT3
End Response
Message “Computing Totals Complete. Please check.”
```

Procedural escapes
These may be in Internal Responses or other places such as in the Exist Response for a particular field. As a matter of style it is better to always have them in internal responses and include those resposes where needed.

Form Records
These define how form data items are organised. This is for the purpose of requests.

A request may refer to one form record or more generally to several records by way of a Form Record List
As a matter of style always include Form Record Lists in a request instead of individual records.

Fields in a Form Record always correspond to a Form Data Item. The correspondence is implicit based on field names or can be overridden using the TRANSFER clause.

Using the Copy clause to include the same CDO record in the form as is used in the program can guarantee default transfers of data,

Example (if you must insist on defining records explicitly)

```
   Form Record MY_RECORD
            MY_KEY    character(12)
            MY_DATA character (50)
            MY_BAL    Longword Integer
            MY_DTS     Datetime (8)
End Record
```

Record List
These are used for transferring multiple records in (for example) a forms$transceive

Example:

```
Record List
     RECORD_1
     RECORD_2
End
```

Then

```
CALL “forms$transceive”
USING
```

Requests

These are the interface to the forms manager.

Request	purpose	COBOL Example
forms$enable	Initial setup. An association between a form and a display device is established, an instance of the form data is created. This is called a session and has a unique session-id.	Call forms$enable
forms$send	Program sends data to form	Call forms$send
forms$receive	Program receives data from form	Call forms$receive
forms$transceive	Data is exchanged between program and form	Call forms$trsansceive
forms$disable	Form is disabled from further activity	Call forms$disable
forms$cancel		

Enable Request

Enable Request in COBOL

Eg:

```
01  SESSION-ID                      PIC X(16) GLOBAL.
01  DEVICE-NAME                        PIC X(9)  VALUE "SYS$INPUT".
01  FORM-FILE                      PIC X(10) VALUE
                                "EZITRAK011".
01  FORMS-STATUS                       PIC S9(9) COMP GLOBAL.
01  SINGLE_REC_COUNT                       PIC S9(5) COMP VALUE 1 GLOBAL.
01  NO-TIMEOUT                     PIC S9(9) COMP VALUE 0 GLOBAL.
01  ORIGINAL_REQ                       PIC S9(9) COMP VALUE 0 GLOBAL.
01  NO-OPTIONS                     PIC S9(9) COMP VALUE 0 GLOBAL.
01  NO-SHAD                        PIC S9(9) COMP VALUE 0 GLOBAL.

01  SEND_REC_NAME                      PIC X(14).
01  RECE-REC-NAME                      PIC X(14).

  CALL "FORMS$ENABLE"
  USING
    OMITTED
    BY DESCRIPTOR  DEVICE_NAME
    BY DESCRIPTOR  SESSION_ID
    BY DESCRIPTOR  FORM-FILE,
  GIVING          FORMS-STATUS.

  IF FORMS-STATUS IS FAILURE
     PERFORM 8300-FORMS-ERROR
     GO TO   9900-EXIT
  END-IF.
```

Transceive Request

COBOL example:

```
   01  HEADER_REC_NAME                  PIC X(14) VALUE
                                        "EZITRAK011_REC".

   MOVE HEADER_REC_NAME TO SEND_REC_NAME.
   MOVE HEADER_REC_NAME TO RECE_REC_NAME.

   CALL "FORMS$TRANSCEIVE"
   USING
     BY DESCRIPTOR  SESSION_ID
     BY DESCRIPTOR  SEND_REC_NAME
     BY REFERENCE   SINGLE_REC_COUNT
     BY DESCRIPTOR  RECE_REC_NAME
     BY REFERENCE   SINGLE_REC_COUNT
     BY DESCRIPTOR  INP_CTL_STRING
     BY REFERENCE   INP_CTL_COUNT
     BY DESCRIPTOR  OUT_CTL_STRING
     BY REFERENCE   OUT_CTL_COUNT
     BY VALUE          NO_TIMEOUT
                       ORIGINAL_REQ
                      NO_OPTIONS
     BY DESCRIPTOR  EZITRAK011_REC
     BY VALUE          NO_SHAD
     BY DESCRIPTOR  EZITRAK011_REC
     BY VALUE          NO_SHAD
   GIVING              FORMS-STATUS.

   IF FORMS-STATUS IS FAILURE
      PERFORM 8300-FORMS-ERROR
      GO TO   9900-EXIT
   END-IF.
```

And in the form:

```
Transceive Response EZITRAK011_REC EZITRAK011_REC
         Activate
            Panel EZITRAK01_HEADER
         Message
            ERROR_MESSAGE
 End Response
```

Application control

There are basically two ways to control processing in an application using DECFORMS:

1. The main program maintains control, accesses the database, does calculations, and only invokes DECFORMS via a forms$ request when necessary to display and accept data.

2. The DECFORMS maintains control, is only invoked once using the forms$enable request, and accesses the database and does calculations by calling sub programs in procedural escapes.

Note that the activation list is emptied after each request, so if using the first option, the main program needs to keep track of which keys were pressed and which fields were active.

In the second option, DECFORMS maintains the activation list until the forms$disable request.

Building the application

First construct the IFDL file using a suitable text editor such as LSE

$EDIT/LSE myform.ifdl

Then compile the form to creat a .form file

$FORMS/TRANSLATE/LIS myform.ifdl

Create the object file for the linker:

$FORMS EXTRACT OBJECT myform.form

Compile the main program and all sub programs used in procedural escapes

$COB/LIS mymain.cob

$COB/LIS pgm001_calc_tots.cob

Creeate the image file

$LINK mymain, myform, pgm001_calc_tots

Set some logicals:

$DEFINE forms$trace T
$DEFINE forms$trace_file “mymain.trace”
$DEFINE forms$default_device sys$input

Run the app:

$Run mymain.exe

Any errors can be found in the trace file called mymain.trace.