Rapid Prototyping of ALL-IN-1 Applications
Tools & Techniques
Session OA-110

This discussion describes various homegrown tools and techniques used to facilitate the rapid prototyping & development of fully-functional ALL-IN-1 data applications. The steps required to establish your own such system will be discussed, as well as general tips on naming conventions, application storage areas, etc.

Alan D. Hull

Principal Software Specialist

Digital Equipment - Great Lakes District EIS

December 10, 1990, 4:00 pm

Digital Equipment Corporation

1 Introduction

One aspect that many people fail to realise about ALL-IN-1 is the powerful data storage handling capabilities it offers. By using forms (menu, arg, entry, and index) and scripts, complete applications can be written that are flexible, fully integrated with any other ALL-IN-1 applications, and offer a user interface already known to ALL-IN-1 users, so no retraining is necessary.

Over the past five years I have written many such data applications, and as a result of that effort I developed a generic data application which I use as the base-level code for all new applications. I have also gathered together various tools and established techniques to manipulate this generic application to enable me to rapidly create, in a matter of a few hours, new data applications. This discussion will detail those steps and techniques.

2 Standards Followed

The only real standard that needs to be followed religiously is that *ALL* file names, form names, etc., must all begin with the same unique application mnemonic. For example, if I were writing a "Request for Change Notice" system, I could use *RCN_* as the prefix to all forms, files, etc. This use of one name for all related files in a given application is critical to the success of the techniques described here. It also allows for very easy file lookups at a later date, since all related files begin with the same unique application mnemonic.

To illustrate the point, here is the directory of my generic application files, which all begin with *APPL* :

```
Directory [.APPLV23]APPL*.*

APPL$INDEX$FIND.FLG;3                 APPL$INDEX$MENU.FLG;5
APPL$INDEX.FLG;14   APPL.FDL;3        APPL.FLB;3          APPLBRIEF.BLP;2
APPLFULL.BLP;2      APPLHDRB.BLP;2    APPLHDRF.BLP;2      APPL_CREATE.SCP;4
APPL_DELETE.SCP;13  APPL_DOWN.SCP;4   APPL_ENTRY.FLG;8    APPL_MRG_RPT.DTR;2
APPL_MRG_RPT.SCP;2  APPL_NEXT.SCP;3   APPL_PRINT.FLG;2    APPL_PRINT.SCP;3
APPL_PRINT_BRIEF.SCP;2                APPL_PRINT_FULL.SCP;2
APPL_REPORTS.FLG;2  APPL_SEL.FLG;7    APPL_SYSMGR.FLG;12  APPL_UP.SCP;5

Total of 24 files.
```

As you can see, all the necessary files begin with *APPL*. Even if you don't make use of an application prototyping system like this, this is a good standard to follow, since it makes the movement of related files, as well as backing them up, very easy.

Table 1: Brief description of my generic application files

File	Description
APPL$INDEX$FIND.FLG	Index selection criteria form
APPL$INDEX$MENU.FLG	Index Gold-M menu of options
APPL$INDEX.FLG	Index form
APPLBRIEF.BLP	Multiple print brief format template
APPLFULL.BLP	Multiple print full format template
APPLHDRB.BLP	Multiple print brief format header template
APPLHDRF.BLP	Multiple print full format header template

Table 1 (Cont.): Brief description of my generic application files

File	Description
APPL_CREATE.SCP	Called by Create option on menu
APPL_DELETE.SCP	Called by Delete option on menu
APPL_DOWN.SCP	Down arrow on menu - rolls to next record in file
APPL_ENTRY.FLG	Data entry form for application
APPL_MRG_RPT.DTR	DATATRIEVE report template
APPL_MRG_RPT.SCP	Script to drive report generation
APPL_PRINT.FLG	Multiple print option menu form (Brief or Full)
APPL_PRINT.SCP	Print option from main application menu (same as Print Full)
APPL_PRINT_BRIEF.SCP	Index multiple-print brief script
APPL_PRINT_FULL.SCP	Index multiple print full script
APPL_REPORTS.FLG	Reports submenu
APPL_SEL.FLG	Select form
APPL_SYSMGR.FLG	Main application menu, full access
APPL_UP.SCP	Up arrow on menu - rolls to first record in file
CHANGE.COM	Global Search & Replace edit routine
FMS_DUMPLIB.COM	Extract .FLB contents out to .FLG description files
MORE$SCROLL$KEYS$INDEX.FLG	Index form key defs
REBUILD_FORMS.COM	Find all APPL* forms, Translate, rebuild formlib
RENAME_FILES.COM	Renames all APPL* files to use application prefix name

3 Tools Used

I use three DCL command procedures to accomplish 90% of the work involved in creating a new application prototype:

1. CHANGE.COM
2. RENAME_FILES.COM
3. REBUILD_FORMS.COM
4. FMS_DUMPLIB.COM

CHANGE.COM is the heart of my system. This simple command file allows me to make sweeping changes in groups of files (a true Global Search & Replace) all at once, speeding the process of cut-n-paste coding hundreds of times over. It also greatly reduces the possibility of introducing typing errors in your code. The command interface is

```
$ CHANGE :== @SYS$LOGIN:CHANGE  !set this up in your LOGIN.COM
$ CHANGE file-spec search-str new-str
```

so for example, if I wanted to change all occurances of *FLD1* in files *RCN*.** to *PART_NO*, I would issue the command

```
$ CHANGE  RCN*.*  FLD1  PART_NO
```

If the string to change contains spaces, you need to enclose each string in quotes, like

```
$ CHANGE  RCN*.*  "FLD1"  "PART NO"
```

Figure 1: CHANGE.COM

```
$
$ ! C H A N G E . C O M
$ !
$ ! Change a string in the files that contain the string
$ ! A. D. Hull - DEC
$ !
$ FILE_SPEC = P1
$ SEARCH_STR = P2
$ NEW_STR = P3
$
$ if file_spec .nes. "" then goto get_search_str
$ read/prompt="File spec: " sys$command FILE_SPEC
$ IF F$LEN(FILE_SPEC) .EQ. 0 THEN GOTO ALL_DONE
$get_search_str:
$ if search_str .nes. "" then goto scan_files
$ read/prompt="String to search for: " sys$command SEARCH_STR
$ IF F$LEN(SEARCH_STR) .EQ. 0 THEN GOTO ALL_DONE
$
$scan_files:
$ WRITE SYS$OUTPUT "Searching . . ."
$ SEARCH/WIND=0/OUT=DOCHANGE.DAT 'FILE_SPEC' "''SEARCH_STR'"
$ WRITE SYS$OUTPUT "Search done."
$
$get_new_str:
$ if new_str .nes. "" then goto do_it
$ read/prompt="Change String To: " sys$command new_str
$ IF F$LEN(NEW_STR) .EQ. 0 THEN GOTO ALL_DONE
$
$do_it:
$ OPEN/WRITE TEMP_FILE DOCHANGE.COM
$ WRITE TEMP_FILE "$ OPEN/READ IN_FILE DOCHANGE.DAT"
$ WRITE TEMP_FILE "$ TOP:"
$ WRITE TEMP_FILE "$ READ/END=CLOSE_FILE IN_FILE FILE"
$ WRITE TEMP_FILE "$ WRITE SYS$OUTPUT "" """
$ WRITE TEMP_FILE "$ EDIT/EDT/NOCOMMAND 'FILE'"
$ WRITE TEMP_FILE "S/''SEARCH_STR'/''NEW_STR'/ wh"
$ WRITE TEMP_FILE "EX"
$ WRITE TEMP_FILE "$ GOTO TOP"
$ WRITE TEMP_FILE "$ CLOSE_FILE:"
$ WRITE TEMP_FILE "$ CLOSE IN_FILE"
$ WRITE TEMP_FILE "$ DELETE/NOLOG DOCHANGE.DAT;*"
$ WRITE TEMP_FILE "$ EXIT""
$ CLOSE TEMP_FILE
$
$ @DOCHANGE.COM
$ DELETE/NOLOG DOCHANGE.*;*
$
$ ALL_DONE:
$! IF VFY THEN SET VER
```

RENAME_FILES.COM prompts you for the old prefix name like *APPL*, then prompts you for the new application prefix name, like *RCN*, and then searches the default directory for all occurances of the old name. A list is built, and then all those files are renamed to the new prefix. Thus *APPL_UP.SCP* becomes *RCN_UP.SCP*, etc. This is always a one-time operation, unless you decide to change the prefix name of your application.

Figure 2: RENAME_FILES.COM

```
$! RENAME_FILES.COM  - renames all xxx*.* files to yyy*.*
$!
$! written By A. D. Hull - DEC
$!
$ old_appl = p1
$ if old_appl .nes. "" then goto cont
$ read/prompt="Enter name (prefix, ie APPL ) of old Application: " sys$command old_appl
$ if old_appl .eqs. "" then exit
$cont:
$ new_appl = p2
$ if new_appl .nes. "" then goto cont2
$ read/prompt="Enter name (prefix, ie XYZ ) of new Application: " sys$command new_appl
$ if new_appl .eqs. "" then exit
$cont2:
$ old_appl = f$edit(old_appl, "UPCASE")
$ new_appl = f$edit(new_appl, "UPCASE")
$ dir/out=files.lis/notrailer/column=1 'old_appl'*
$ open/read list files.lis
$! read past the header info
$ read/end=file_eof list filespec
$ read/end=file_eof list filespec
$ read/end=file_eof list filespec
$loop:
$ read/end=file_eof list filespec
$ new_filespec == "''filespec'" - "''old_appl'"
$sho sym new_filespec
$ new_filespec == "''new_appl'" + "''new_filespec'"
$sho sym new_filespec
$ rename/log 'filespec' 'new_filespec'
$ goto loop
$file_eof:
$ close list
$ delete/nolog files.lis;*
$exit
```

REBUILD_FORMS.COM searches for all form description files (.FLG) with the desired prefix, builds a list, and then uses FMS/TRANSLATE on each form description file to translate the text back into a binary form. You are prompted to enter the new application's prefix name, like *RCN*. All *RCN*.FLG* files are translated and placed into a formlibrary called *RCN.FLB*. The Index form key defs form is then also added to the new formlib.

Figure 3: REBUILD_FORMS.COM

```
$! REBUILD_FORMS.COM - gen list of .FLG files and then FMS/TRANSLATE them
$! written by A.D. Hull - DEC 4/8/88
$!
$ on error then goto error_exit
$ all_forms = ""
$ form_name = ""
$ new_appl = p1
$ if new_appl .nes. "" then goto cont
$ read/prompt="Enter name (prefix, ie APPL ) of new Application: " sys$command new_appl
$ if new_appl .eqs. "" then exit
$cont:
$ purge/log *.flg
$ dir/out=files.lis/notrailer/column=1 'new_appl'*.flg
$ open/read list files.lis
$! read past the header info
$ read/end=file_eof list full_filespec
$ read/end=file_eof list full_filespec
$ read/end=file_eof list full_filespec
$ first = 1
$loop:
$ read/end=file_eof list full_filespec
$ filespec = f$parse(full_filespec,,,"NAME") + f$parse(full_filespec,,,"TYPE")
$ form_name = filespec - ".FLG"
$ gosub build_form_list
$ write sys$output "Creating form ''form_name'"
$ fms/translate 'form_name'
$ goto loop
$build_form_list:
$ if first then goto first_name
$ all_forms = all_forms + "," + form_name
$ return
$first_name:
$ all_forms = form_name
$ first = 0
$return
$file_eof:
$ close list
$ delete/nolog files.lis;*
$sho sym all_forms
$create_frmlib:
$ fms/translate more$scroll$keys$index
$ fms/lib/cre/log  'new_appl'  'all_forms'
$ fms/lib/replace/log  'new_appl'  more$scroll$keys$index
$exit
$!
$error_exit:
$ write sys$output "Error encountered. Please correct and rerun this."
$ if f$trnlnm("list") .nes. "" then close list
$exit
```

If the named data on a form or a form display text is *CHANGEd* such that the final string length is greater than 80 characters (usually due to field names being too long), the translation for that form will fail. You will have to go back in and edit the .FLG file to fix the problem before you can rerun this procedure.

FMS_DUMPLIB.COM is the opposite of REBUILD_FORMS.COM. FMS_DUMPLIB will take any form library, extract each form, and create a full description file (.FLG) of the form and leave it in the current default directory. This is useful if you forgot to make a global change, or want to go back and rename one of the FLD1—FLD4 names again. You would dump all the forms, CHANGE them as needed, and then rerun REBUILD_FORMS.

Figure 4: FMS_DUMPLIB.COM

```
$! FMS_DUMPLIB.COM - WJR                - Digital Equipment Corporation
$!                                      - Chicago District
$! mod. by A.D. Hull - DEC - 14-Apr-87  - prompt for Formlib, gen the fms/dir
$!                                        listing, then proceed
$!
$! This command procedure creates full form descriptions and named data
$!  only descriptions
$!
$! Inputs:
$!  P1 = name of the input form library
$!  P2 = keep the full description (Y or N) [default is Y]
$!
$ IF F$LENGTH(P2) .EQ. 0 THEN $P2 = "Y"
$ CNT = 1
$ ON ERROR THEN GOTO 40
$ 5:
$ IF F$LENGTH(P1) .GT. 0 THEN FILE_1 = P1
$ CNT = CNT + 1
$ IF CNT .GE. 4 THEN $EXIT
$ IF F$LENGTH(P1) .EQ. 0 THEN GOTO 10
$ FLB = F$PARSE(P1,,,"NAME")
$ FMS/DIR/OUT='FLB'.LIS 'P1
$ FILE_1 := 'FLB'.LIS
$ GOTO 20
$ 10:
$ READ/PROMPT="Enter Form Library Name --> " sys$command P1
$ GOTO 5
$ 20:
$ OPEN /READ F1 'FILE_1'
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$!
$ READ /END=50 F1 LIB1
$ LIB = F$EXTRACT(8,(F$LOCATE(";",LIB1)-8),LIB1)
$ EXT = F$PARSE(LIB,,,"NAME")
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ READ /END=50 F1 LIB1
$ 30:
$ READ /END=40 F1 FORM
$ set mess/nof/noi/nos/not
$ FMS/DESCRIP/FULL 'LIB'/FORM='FORM'/OUT='FORM'.FLG /LOG
$ set mess/f/i/s/t
$ TMP_FORM_FLX := 'FORM'.FLG
$ write sys$output "Full Form description is in file  ''TMP_FORM_FLX'"
$ GOTO 30
$ 40:
$ CLOSE F1
$ set mess/f/i/s/t
$ DELETE/NOLOG 'FILE_1';*
$ EXIT
$ 50:
$ CLOSE F1
$! I just love homemade error messages ...
$ WRITE SYS$OUTPUT "%RAZ-F-NOTENGH, Not enough data available"
$ EXIT
```

4 Creating Your Initial Generic Application

Your hardest task will be the initial setting up of your own generic application to use as your base system from now on. The best advice in doing this is take an application that works the way you want, strip it down to the minimum level of code needed, and make all field names, form names, etc, completely generic, following the convention in the previous section. My initial starting point for my code was the DIR PER (Personal Phone Directory) subsystem in ALL-IN-1. I had to clean up the code somewhat, and then added all the standard stuff that we always used, like the up and down arrows, standardized style of multiple printing, standard style report procedures, etc.

Once you have your forms and scripts set up, if you need to make an addition to something, or you have some new code to add, just add it into the generic application as needed, and you'll pick it up on all future applications as you generate them.

Refer to the Scripts Used section at the end of this document for examples of my generic code.

5 Design Standards Followed

Main Application Menu:

The main menu for the application is always called APPL_SYSMGR, implying that access to this form gives the user full access to the data (add/delete/change). Once I have an application completely coded, I go back and copy the APPL_SYSMGR menu form to APPL_ USER, which is a restricted menu, allowing only Read-only and Print options.

Each menu follows the ALL-IN-1 standard of having a current context window in the upper right of the screen. I always start with 4 fields in the window (3 for the key fields and 1 other data item), and adjust the application as necessary. Whatever is in the context window is considered the "current record", and any subsequent menu options will work against that record. The menu options are "SEL, C, E, D, P, R, I, CO, RI".

Select form:

The Select form, APPL_SEL, is initially coded with the 3 key fields. I write most applications to use a 3-part segmented primary key. When a user Creates a record, the Select form is displayed first. The key field values are entered, and then I check to see iof that key already exists. If it does, I call the entry form in CHANGE mode, else I call it in ADD mode. This is more user-friendly than having ALL-IN-1 dump you back to the menu with "Record already on file".

Up & Down Arrow Keys:

Pressing the Up Arrow (APPL_UP.SCP) always retrieves the first record in the datafile, and sets the display in the context window. If the datafile exists, but is empty, a message is displayed to the user. Pressing the Down Arrow (APPL_DOWN.SCP) will roll to the next record in key order in the context window. If you are on the last record, it rolls over to the first record.

Index and Index support forms:

The Index (APPL$INDEX) is the most powerful feature of ALL-IN-1 data applications. You can do most operations from the Index, and to multiple records at once. The first form displayed is the Find form (APPL$INDEX$FIND), where the user selects the field criteria to use in the Index lookup. Once the display has come up showing the selected records, the user can press Gold-M for an options menu (APPL$INDEX$MENU). After the Index form has been coded and tested, a copy of the form having only Read and print capabilities is added into the application, and is called from the APPL_USER menu.

Data Entry form:

The entry form (APPL_ENTRY) is initially generated with just 4 fields. These are named FLD1, FLD2, FLD3 and FLD4. Fields 1-3 make up the record key as generated. This key can be adjusted later to just 1 or two fields if necessary. All the named data to handle field validation and recognition is already in place for those 4 fields, as well some handy code examples in dummy named data entries. Once the key fields are set up as desired, you go back later and finish adding the remaining fields and possibly other entry forms if this requires using a form-set to map all your data.

6 Building Your Prototype - Step by Step

6.1 Copying Generic Files

I store all my generic application in a subdirectory off my login level, in [hull.applv23]. When I start a new application, the first step is to create a new subdirectory to work out of. I just create it 1 more level down, but it could be anywhere. Lets say our new application prefix is *RCN*. For this discussion, I'll use [hull.applv23.rcn] as my working directory. Copy all the APPL*.* files into the working directory. Also copy MORE$SCROLL$KEYS$INDEX.FLG into the working directory. I usually leave the .COM files in their original location, and reference them from that location.

6.2 Renaming Files

Our next step is to rename all the APPL*.* files we just copied to RCN*.*. Since the DCL verb RENAME doesn't accept multiple files in 1 operation, we use the RENAME_ FILES.COM command file.

```
$ @[-]RENAME_FILES
Enter name (prefix, ie APPL ) of old Application: APPL
Enter name (prefix, ie XYZ ) of new Application: RCN
```

The procedure prompts you for the old prefix (enter *APPL*), and then for the new prefix to use (enter *RCN*). All the files will be renamed to RCN*.*.

6.3 Defining Initial Field Names

Before you go any further, you should have your data form roughed out on paper, and the exact field names you want to use for the initial 3 key fields and 1 data field. Remember, these 4 fields are the current item window fields, so these must be defined at a minimum. Try to keep the field names under 8 characters or so, and use underscores for readability. for instance, PART NUMBER is best named as PART_NO. The 3 key fields are generally the first three fields in order on the entry form, but the 4th field can be placed anywhere within the record once the form is fully developed.

6.4 Renaming Field Names

This is the phase where you application really begins to take shape. There are 6 names that need changing in my generic application: *FLD1, FLD2, FLD3, FLD4, $THEKEY* and *APPL*. When you rename the fields, make sure you don't use any FMS reserved keywords, like TYPE, DATA, FIELD, etc. These reserved words show up in the .FLG files, and if you accidentally change a field to "TYPE", you are in for lots of trouble. After each CHANGE action is finished, you must $ PURGE the directory to only keep the most recent version of the files. So. for example, the series of CHANGEs made might look like this:

```
$ CHANGE RCN*.*  FLD1   RCN_NO
$ PURGE/LOG
$ CHANGE RCN*.*  FLD2   DESCRIP
$ PURGE/LOG
$ CHANGE RCN*.*  FLD3   PART_NO
$ PURGE/LOG
$ CHANGE RCN*.*  FLD4   ENGR
$ PURGE/LOG
$ CHANGE RCN*.*  $THEKEY $RCN_KEY
$ PURGE/LOG
$ CHANGE RCN*.*  APPL   RCN
```

To see if I've forgotten anything during this phase, I type out the RCN_UP.SCP file (which references almost all the variables) after each pass.

This is what the file looks like before the changes:

```
! APPL_UP.SCP
! Allows the user to go to the top entry of his current data.

 GET $THEKEY = appl_entry.%NEXT[""]
 .if $thekey eqs "" then .goto file_empty

 for first appl_entry with .%key == $thekey do -
  GET $APPL_FLD1 = .FLD1 -
  \\GET $APPL_FLD2 = .FLD2 -
  \\GET $APPL_FLD3 = .FLD3 -
  \\GET $APPL_FLD4 = .FLD4

 DISPLAY Top of Log

 .exit

.label file_empty

 get $thekey = $appl_fld1 = $appl_fld2 = $appl_fld3 = $appl_fld4 = ""
 display File is empty
 .exit
```

and here's the same file after the global edits have been applied:

```
! RCN_UP.SCP
! Allows the user to go to the top entry of his current data.

 GET $RCN_KEY = RCN_entry.%NEXT[""]
 .if $RCN_KEY eqs "" then .goto file_empty

 for first RCN_entry with .%key == $RCN_KEY do -
  GET $RCN_RCN_NO = .RCN_NO -
  \\GET $RCN_DESCRIP = .DESCRIP -
  \\GET $RCN_PART_NO = .PART_NO -
  \\GET $RCN_ENGR = .ENGR

 DISPLAY Top of Log

 .exit

.label file_empty

 get $RCN_KEY = $RCN_RCN_NO = $RCN_DESCRIP = $RCN_PART_NO = $RCN_ENGR = ""
 display File is empty
 .exit
```

6.5 Rebuilding the Form Library

Once you've made all the global edits you can, you now need to rebuild the formlib and continue any editing of forms in the FMS editor itself. Assuming you kept your field names reasonably short, all your forms should convert back to binary format without any trouble.

If you have an occurance where the named data in one of the forms is too long after you made a CHANGE, and the FMS/TRANSLATE failed, you'll have to manually go in and find the error.

This is the kind of error you'll see:

```
%FMS-E-INVNMDLEN, The length of data is invalid.
%FMS-E-ENDNOFRM, Translation was completed with errors. No binary form was output.
```

Use the command

```
$ FMS/TRANSLATE/NOOUTPUT/LIST=TEMP.LIS form
$ TYPE TEMP.LIS
```

to see where the error occurs. Then use the editor to cut that line in half, and add in another "index" into the named data. In other words, if index=5 (out of 30) and the line is too long to translate, you'll have to chop index line 5 in half, and put the new half below 5. This means that all your subsequent index numbers on the form are off by one, so you'll have to manually edit the numbers to properly resequence them. Make sure you have all the necessary 'INDEX=' statements as needed, and that you have all the necessary single quote marks as requiredû in the FMS syntax.

Here's an example of a line being too long, and after it's been "adjusted":

```
!          FMS Form Description Application Aid
!                    Version V2.3

FORM NAME='APPL_PRINT'

[listing cut here for brevity]

NAMED_DATA INDEX=1 NAME='.TYPE'
    DATA='MENU /CHOICE=CHOICE/CLEAR/OVERLAY/HARD=''APPL PRINT OPTIONS''/ONCE/XXXXXXXXXXXXXXXXXX' ;  <<<<< line too long
NAMED_DATA INDEX=2 NAME='OA$_MO_BRIEF'
    DATA='DO APPL_PRINT_BRIEF' ;
NAMED_DATA INDEX=3 NAME='OA$_MO_FULL_DESCRIPTION'
    DATA='DO APPL_PRINT_FULL' ;
NAMED_DATA INDEX=4 NAME='CHOICE'
    DATA='/HARD=''B,F''' ;

END_OF_FORM NAME='APPL_PRINT' ;

[Now, INDEX 1 has been cut into two pieces, and all Index numbers adjusted,
turning 4 index lines into 5]

!          FMS Form Description Application Aid
!                    Version V2.3

FORM NAME='APPL_PRINT'

[listing cut here for brevity]

NAMED_DATA INDEX=1 NAME='.TYPE'
    DATA='MENU /CHOICE=CHOICE/CLEAR/OVERLAY' ;
NAMED_DATA INDEX=2 NAME='.TYPE'                                    <<<<<< new
    DATA='/HARD=''APPL PRINT OPTIONS''/ONCE/XXXXXXXXXXXXXXXXXX' ;  <<<<<< new
NAMED_DATA INDEX=3 NAME='OA$_MO_BRIEF'                             <<<<<< all rest resequenced
    DATA='DO APPL_PRINT_BRIEF' ;
NAMED_DATA INDEX=4 NAME='OA$_MO_FULL_DESCRIPTION'
    DATA='DO APPL_PRINT_FULL' ;
NAMED_DATA INDEX=5 NAME='CHOICE'
    DATA='/HARD=''B,F''' ;

END_OF_FORM NAME='APPL_PRINT' ;
```

6.6 Defining ALL-IN-1 File Search Order

Now we have all the scripts, templates and forms changed to our base level of the application prototype. The remaining work in now done through the ALL-IN-1 Ford Development system. But before that can happen, we need to define a new file search order so that ALL-IN-1 will be able to find all our scripts, etc., before we move them all into a production area.

To accomplish this you just need to create a User-Defined Process (UDP) under the WP UD menu which sets the search order, and opens the formlib for us. Assuming that all the files are in [hull.applv23.rcn] with formlib RCN.FLB, the UDP would be coded as:

```
! RCN.UDP
.fx get oa$file_search_order = oa$file_search_order ",[hull.applv23.rcn]"
.fx oa$flo_open [hull.applv23.rcn]rcn
rcn_sysmgr{CR}
```

Each time you log back into ALL-IN-1 with a new session, to work on the application, you just press the DO key, or Gold U, and select *RCN* as the UDP to execute. You should only do this once per ALL-IN-1 login, otherwise you will have a broken file search order.

6.7 Final Revisions & Cleanup

The new application code which has been generated should work as a prototype system without any problems. Depending on how you coded your generic code regarding field lengths, etc, the "form" of the screens and the data fields may not be correct, but everything should work on a functional level. In my case, since I use a 3-part key, I have to create an FDL file

to create the data file before I can actually work with any data. This FDL file is referenced in the entry form.

At this stage you just need to go back in and finish up the forms, namely creating and/or adjusting field lengths, menu text, etc. If you find you need to make a sweeping change to the forms again, exit ALL-IN-1, go back to the working directory, and use the *FMS_DUMPLIB* routine to extract current copies of your forms. Purge your directory, and Delete any .FRM files that were left. Now make your changes using *CHANGE*, and then go through the same steps previously outlined to rebuild the forms, etc.

Once all your forms are locked in, and no more script changes are needed, etc., you can go in and finish up the print templates and the read-only versions of the menu and index forms. You should work very closely with the users of the application to tailor it to their exact needs and appearance.

Once you have a good generic application made up, you ought to be able to have a solid, working prototype application up and running in about 4 hours or less.

7 Scripts and Templates Used

Here is a listing of the major generic application scripts:

Figure 5: APPL_CREATE.SCP

```
! appl_create.scp
! A.D. Hull - DEC -   16-JUN-1988
!

.label get_record

 form appl_sel arg/reset/clear/highlight/fields=FLD1,FLD2,FLD3
 .if oa$form_dispose == 0 then .goto finis
 get #FLD1_tmp = FLD1
 get #FLD2_tmp = FLD2
 get #FLD3_tmp = FLD3
 get #THEkey = FLD1:N FLD2:N FLD3:N
 .if appl_ENTRY.%key[#THEkey] == '' then get #CREATE_MODE = 'ADD' -
  else get  #CREATE_MODE = 'CHANGE'

 close prior
 .if #CREATE_MODE eqs 'ADD' then .goto add
 .if #CREATE_MODE eqs 'CHANGE' then .goto change

.label add

 FORM appl_ENTRY /mode=ADD/SAVE_START="#FLD1_tmp,#FLD2_tmp,#FLD3_tmp"

 IFEXIT -
 \GET $THEKEY = $APPL_FLD1:n $APPL_FLD2:n $APPL_FLD3:n -
 \GET $APPL_FLD4 = appl_ENTRY.FLD4[$THEKEY]

 .goto finis

.label change

 FORM appl_ENTRY/mode=CHANGE/SAVE_START="#FLD1_tmp,#FLD2_tmp,#FLD3_tmp"

 IFEXIT -
 \GET $THEKEY = $APPL_FLD1:n $APPL_FLD2:n $APPL_FLD3:n -
 \GET $APPL_FLD4 = appl_ENTRY.FLD4[$THEKEY]

 .goto finis

.label finis
 .exit
```

Figure 6: APPL_DELETE.SCP

```
! APPL_delete.scp
! written 2-4-88 A.D. Hull - DEC

 .IF $THEKEY EQS "" THEN .GOTO NO_CURRENT_ITEM

  YESNO_PROMPT\OA$FLD_STAY
 .IF OA$PROMPT_TEXT EQS 'Y' THEN -
  FORM appl_entry/MODE=DELETE/ONE_ENTRY/SAVE_START="$APPL_FLD1,$APPL_FLD2,$APPL_FLD3" -
 \IFEXIT\GET $APPL_FLD1=$APPL_FLD2=$APPL_FLD3=$APPL_FLD4=$THEKEY="" -
 \display APPL Item has been deleted
 .exit

.label no_current_item

  DISPLAY APPL Item must be SELected to delete it\FORCE
 .exit
```

Figure 7: APPL_DOWN.SCP

```
! APPL_DOWN.SCP
! Allows user to go down through APPL Item file listings by
! using down arrow key.

 GET $THEKEY = appl_entry.%NEXT[$THEKEY]
 for first appl_entry with .%key == $thekey do -
  GET $APPL_FLD1 = .FLD1 -
  \\GET $APPL_FLD2 = .FLD2 -
  \\GET $APPL_FLD3 = .FLD3 -
  \\GET $APPL_FLD4 = .FLD4

 .if $thekey = appl_entry.%NEXT[""] then -
  display Top of Log
 .exit
```

Figure 8: APPL_PRINT.SCP

```
! APPL_print.scp  - create a merged output of the current APPL record,
! and send it to your favorite local printer.
! written by A.D. Hull - DEC  1-Feb-1988


 MERGE appl_entry.blp, appl_entry.wpl

 get #print_file = 'appl_entry.WPL'
 get #print_dsab = 'WPSPLUS'
 get #print_object = 'APPL Item'
 get #print_delete = 'Y'

 DO WPPRINT

 DELETE_FILE appl_entry.WPL
 .exit
```

Figure 9: APPL_PRINT_BRIEF.SCP

```
! APPL_PRINT_BRIEF.SCP
! This script prints a brief description of the records contained in
! the current index.
.LABEL START
!DISPLAY "Creating directory listing of selected records . . ."
  GET OA$DISPLAY=OA$_DIR_CREDIRLST\force
 GET #SELFILE="APPLLIST.SEL"
 .IF OA$DIR:"*.*".%WHOLE[#SELFILE] == "" THEN .GOTO NO_SEL
 MERGE APPLBRIEF.BLP,APPLBRIEF.LIS,APPLLIST.SEL,APPLHDRB.BLP
 GET #PRINT_FILE="APPLBRIEF.LIS"
 GET #PRINT_DSAB="ASCII"\DO WPPRINT
 .EXIT

.LABEL NO_SEL
!DISPLAY "You must do an Index to create the selection of records to print"
  GET OA$DISPLAY=OA$_DIR_NOINDAVA
```

Figure 10: APPL_PRINT_FULL.SCP

```
! APPL_PRINT_FULL.SCP
! This script prints a FULL description of the records contained in
! the current index.

!DISPLAY "Creating directory listing of selected records . . ."
  GET OA$DISPLAY=OA$_DIR_CREDIRLST\FORCE
 GET #SELFILE="APPLLIST.SEL"
 .IF OA$DIR:"*.*".%WHOLE[#SELFILE] == "" THEN .GOTO NO_SEL
 MERGE APPLFULL.BLP,APPLFULL.LIS,APPLLIST.SEL,APPLHDRF.BLP
 GET #PRINT_FILE="APPLFULL.LIS"
 GET #PRINT_DSAB="ASCII"\DO WPPRINT
 .EXIT

.LABEL NO_SEL
!DISPLAY "You must do an Index to create the selection of records to print"
 GET OA$DISPLAY=OA$_DIR_NOINDAVA
```

Figure 10: APPL_UP.SCP

```
! APPL_UP.SCP
! Allows the user to go to the top entry of his current data.

 GET $THEKEY = appl_entry.%NEXT[""]
 .if $thekey eqs "" then .goto file_empty

 for first appl_entry with .%key == $thekey do -
  GET $APPL_FLD1 = .FLD1 -
  \\GET $APPL_FLD2 = .FLD2 -
  \\GET $APPL_FLD3 = .FLD3 -
  \\GET $APPL_FLD4 = .FLD4

 DISPLAY Top of Log

 .exit

.label file_empty

 get $thekey = $appl_fld1 = $appl_fld2 = $appl_fld3 = $appl_fld4 = ""
 display File is empty
 .exit
```

Figure 12: APPLBRIEF.BLP

```
<APPL_ENTRY.%OPEN[OA$MERGE_KEY]><->
<&TAB 4><.FLD1:10><&TAB 15><.FLD2:12><&TAB 31><.FLD3:3><->
<&TAB 38><.FLD4:30><->
<APPL_ENTRY.%CLOSE[OA$MERGE_KEY]>
```

Figure 13: APPLFULL.BLP

```
<APPL_ENTRY.%OPEN[OA$MERGE_KEY]><->
<&TAB 2>Log Sheet:<->
<&TAB 14><&UNDERLINE><.FLD1><&CLEAR>
<&TAB 2>FLD2   :<&TAB 14><&UNDERLINE><.FLD2><&CLEAR>
<&TAB 2>FLD3   :<->
<&TAB 14><&UNDERLINE><.FLD3><&CLEAR><->
<&TAB 45>FLD4:<->
<&TAB 57><&UNDERLINE><.FLD4><&CLEAR><->
<APPL_ENTRY.%CLOSE[OA$MERGE_KEY]>
<&PAGE>
```

Figure 14: APPLHDRB.BLP

```
                                  APPL Index
Date: <OA$DATE> <&TAB 65>Page:<OA$PAGE_NUMBER>
--------------------------------------------------------------------------------
    FLD1       FLD2         FLD3      FLD4
--------------------------------------------------------------------------------
```

Figure 15: APPLHDRF.BLP

```
                           APPL Full Listing
Date: <OA$DATE> <&TAB 65>Page:<OA$PAGE_NUMBER>
--------------------------------------------------------------------------------
```

8 Named Data for Main Menu, Index and Entry Form

Here is a listing of the major generic application forms' named data:

Figure 16: Form APPL_SYSMGR Named Data

```
;;.TYPE;;

MENU/USER=USER/DATE=DATE/MAIL=MAIL/CHOICE=CHOICE/CLEAR/GET=DAY,OA$DAY;
FLD1,$APPL_FLD1;FLD2,$APPL_FLD2;
FLD3,$APPL_FLD3;FLD4,$APPL_FLD4
/HARD=' APPL Item Menu'/TITLE=TITLE

;;OA$_MO_SELECT;;

FORM APPL_SEL

;;OA$_MO_CREATE;;

DO APPL_CREATE

;;OA$_MO_EDIT;;

.IF $THEKEY EQS "" THEN
 DISPLAY APPL Item must be SELected to edit it\\FORCE ELSE
 FORM APPL_ENTRY/MODE=CHANGE
/SAVE_START="$APPL_FLD1,$APPL_FLD2,$APPL_FLD3"
\IFEXIT
\GET $APPL_FLD4=APPL_ENTRY.FLD4[$THEKEY]

;;OA$_MO_DELETE;;

DO APPL_DELETE

;;OA$_MO_PRINT;;

.IF $THEKEY EQS "" THEN
 DISPLAY APPL Item must be SELected to print it\\FORCE ELSE
 DO APPL_PRINT

;;OA$_MO_READ;;

.IF $THEKEY EQS "" THEN
 DISPLAY APPL Item must be SELected to read it\\FORCE ELSE
 FORM APPL_ENTRY/MODE=INQUIRE
/SAVE_START="$APPL_FLD1,$APPL_FLD2,$APPL_FLD3"

;;OA$_MO_INDEX;;

OA$SCL_EXIT\BIND_BREAK *APPL\FORM APPL$INDEX

;;OA$_MO_RECALL_INDEX;;

FOR FIRST *APPL DO GET #APPL_KEY = .%KEY\
.IF #APPL_KEY EQS "" THEN
  OA$MSG_PURGE\\GET OA$DISPLAY = OA$_INXNOINDEX ELSE FORM APPL$INDEX

;;FLD1;;

/GET_SAVE=$APPL_FLD1 /HARD="Key"

;;FLD2;;

/GET_SAVE=$APPL_FLD2 /HARD="FLD2"

;;FLD3;;

/GET_SAVE=$APPL_FLD3/HARD="FLD3"

;;APPL_FLD4;;
```

Figure 16 Cont'd. on next page

Figure 16 (Cont.): Form APPL_SYSMGR Named Data

```
/GET_SAVE=$APPL_FLD4 /HARD="FLD4"

;;CHOICE;;

/HARD='OPTIONS: SEL,C,E,D,P,R,I,CO,RI'

;;.DOWN;;

DO APPL_DOWN

;;.UP;;

DO APPL_UP

;;.GOLD T;;

DO APPL_UP

;;CO;;

.IF $THEKEY EQS "" THEN
 DISPLAY APPL Item must be SELected to copy it\\FORCE ELSE
 FORM APPL_ENTRY/MODE=COPY
/SAVE_START="$APPL_FLD1,$APPL_FLD2,$APPL_FLD3"
\IFEXIT
\GET $THEKEY=$APPL_FLD1 $APPL_FLD2 $APPL_FLD3
\GET $APPL_FLD4=APPL_ENTRY.FLD4[$THEKEY]
```

Figure 16: Form APPL$INDEX Named Data

```
;;.TYPE;;

INDEX/CHOICE=CHOICE/CLEAR
/PRE_FUNCTION='OA$SCL_INIT ,,,*APPL
\IFNOTSTATUS\XOP "~~FIND~~"'

;;.MORE;;

MORE$SCROLL$KEYS$INDEX

;;.GOLD M;;

FORM APPL$INDEX$MENU\CLOSE_PRIOR

;;LINE;;

/VIEW=,,,*APPL
/ALIAS=%LINE

;;POINTER;;

/POINTER

;;SELECTED;;

/SELECTED

;;~~FIND~~;;
```

Figure 16 Cont'd. on next page

Figure 16 (Cont.): Form APPL$INDEX Named Data

```
FORM APPL$INDEX$FIND\
IFEXIT\
CLOSE_PRIOR\
OA$SCL_EXIT\
BIND_BREAK *APPL\
XOP "~~BINDW~~"\
OA$SCL_INIT,,,*APPL

;;~~BINDW~~;;

BINDW *APPL TO APPL_ENTRY
 WITH .FLD1 = #APPL_FLD1
 AND  .FLD2 = #APPL_FLD2
 AND .FLD3 = #APPL_FLD3

;;CHOICE;;

/PRE='GET SELECTIONS=OA$SCROLL_SELECTED\GET UNREAD=OA$MAIL_COUNT'

;;OA$_MO_INDEX;;

XOP "~~FIND~~"

;;.GOLD F;;

GET $APPL_FLD1 = APPL_ENTRY.FLD1[OA$SCROLL_KEY]\
GET $APPL_FLD2 = APPL_ENTRY.FLD2[OA$SCROLL_KEY]\
GET $APPL_FLD3 = APPL_ENTRY.FLD3[OA$SCROLL_KEY]\
GET $APPL_KEY = $APPL_FLD1 $APPL_FLD2 $APPL_FLD3\
GET $APPL_FLD4 = APPL_ENTRY.FLD4[$APPL_KEY]\
OA$FLD_EXIT

;;OA$_MO_CREATE;;

FORM APPL_ENTRY/ONE_ENTRY/MODE=ADD\
IFEXIT\
CLOSE_PRIOR\
OA$SCL_REFRESH

;;OA$_MO_EDIT;;

GET #APPL_KEY=*APPL.%KEY[OA$SCROLL_ADDRESS]\
GET #APPL_TEMP_FLD1 = *APPL.FLD1[#APPL_KEY]\
GET #APPL_TEMP_FLD2 = *APPL.FLD2[#APPL_KEY]\
GET #APPL_TEMP_FLD3 = *APPL.FLD3[#APPL_KEY]\
FORM APPL_ENTRY/MODE=CHANGE
/SAVE="#APPL_TEMP_FLD1,#APPL_TEMP_FLD2,#APPL_TEMP_FLD3"\
IFEXIT\
CLOSE_PRIOR\
OA$SCL_UPDATE\
OA$SCL_REFRESH

;;OA$_MO_DELETE;;

GET #APPL_KEY=*APPL.%KEY[OA$SCROLL_ADDRESS]\
GET #APPL_TEMP_FLD1 = *APPL.FLD1[#APPL_KEY]\
GET #APPL_TEMP_FLD2 = *APPL.FLD2[#APPL_KEY]\
GET #APPL_TEMP_FLD3 = *APPL.FLD3[#APPL_KEY]\
FORM APPL_ENTRY/MODE=DELETE
/SAVE="#APPL_TEMP_FLD1,#APPL_TEMP_FLD2,#APPL_TEMP_FLD3"\
IFEXIT\
CLOSE_PRIOR\
OA$SCL_REFRESH

;;OA$_MO_PRINT;;
```

Figure 16 Cont'd. on next page

Figure 16 (Cont.): Form APPL$INDEX Named Data

```
MERGE SELECTIONLIST1,APPLLIST.SEL\
FORM APPL_PRINT\
PURGE_FILE APPLLIST.SEL

;;OA$_MO_X_PRINT;;

XOP "~~CHECK_SELECTIONS~~"\IFSTATUS\
MERGE SELECTIONLIST,APPLLIST.SEL\
FORM APPL_PRINT\
PURGE_FILE APPLLIST.SEL

;;OA$_MO_READ;;

GET #APPL_KEY=*APPL.%KEY[OA$SCROLL_ADDRESS]\
GET #APPL_TEMP_FLD1 = *APPL.FLD1[#APPL_KEY]\
GET #APPL_TEMP_FLD2 = *APPL.FLD2[#APPL_KEY]\
GET #APPL_TEMP_FLD3 = *APPL.FLD3[#APPL_KEY]\
FORM APPL_ENTRY/MODE=INQUIRE
/SAVE="#APPL_TEMP_FLD1,#APPL_TEMP_FLD2,#APPL_TEMP_FLD3"\
IFEXIT\
CLOSE_PRIOR\
OA$SCL_REFRESH\

;;OA$_MO_X_READ;;

XOP "~~CHECK_SELECTIONS~~"\IFSTATUS\
GET #RFA=''\
.FX
 NEXT_SELECTED *APPL,#RFA\\
 IFSTATUS\\
 GET #APPL_KEY = *APPL.%KEY[#RFA]\\
 GET #APPL_TEMP_FLD1 = *APPL.FLD1[#APPL_KEY]\\
 GET #APPL_TEMP_FLD2 = *APPL.FLD2[#APPL_KEY]\\
 GET #APPL_TEMP_FLD3 = *APPL.FLD3[#APPL_KEY]\\
 FORM APPL_ENTRY/MODE=INQUIRE
/SAVE="#APPL_TEMP_FLD1,#APPL_TEMP_FLD2,#APPL_TEMP_FLD3"\\
 CLOSE_PRIOR\\
 REPEAT

;;OA$_MO_X_DELETE;;

XOP "~~CHECK_SELECTIONS~~"\IFSTATUS\
YESNO_PROMPT\
OA$FLD_STAY\
GET #RFA=''\
.IF   OA$PROMPT_TEXT = OA$Y
 THEN
     XOP "~~DELETE_LOOP~~"\
OA$SCL_REFRESH

;;~~DELETE_LOOP~~;;

.FX
 NEXT_SELECTED "*APPL",#RFA\\
 IFSTATUS\\
 GET #APPL_KEY=*APPL.%KEY[#RFA]\\
 WRITE DELETE *APPL %KEY=#RFA\\
 GET #RFA=''\\
 REPEAT

;;OA$_MO_X_EDIT;;
```

Figure 16 Cont'd. on next page

Figure 16 (Cont.): Form APPL$INDEX Named Data

```
XOP "~~CHECK_SELECTIONS~~"\IFSTATUS\
GET #RFA=’’\
XOP "~~EDIT_LOOP~~"\
OA$SCL_REFRESH

;;~~EDIT_LOOP~~;;

.FX
 NEXT_SELECTED *APPL,#RFA\\
 IFSTATUS\\
 GET #APPL_KEY = *APPL.%KEY[#RFA]\\
 GET #APPL_TEMP_FLD1 = *APPL.FLD1[#APPL_KEY]\\
 GET #APPL_TEMP_FLD2 = *APPL.FLD2[#APPL_KEY]\\
 GET #APPL_TEMP_FLD3 = *APPL.FLD3[#APPL_KEY]\\
 FORM APPL_ENTRY/MODE=CHANGE
 /SAVE="#APPL_TEMP_FLD1,#APPL_TEMP_FLD2,#APPL_TEMP_FLD3"\\
 CLOSE_PRIOR\\
 OA$SCL_UPDATE #RFA, #RFA\\
 REPEAT
```

Figure 18: Form APPL_ENTRY Named Data

```
;;.TYPE;;

ENTRY /MODE=UPDATE/HARD='Entry Form'
/KEY=FLD1,FLD2,FLD3
/POST='IFEXIT\GET $APPL_FLD1 = FLD1\GET $APPL_FLD2 = FLD2
\GET $APPL_FLD3 = FLD3'

;;.FILE;;

OA$SITE_DATA_SHARE:APPL.DAT,
OA$SITE_DATA_SHARE:APPL.FDL

;;FLD1;;

/RSE_RECOG=appl_entry.FLD1 WITH (.FLD1 = FLD1 AND
 .FLD2 = FLD2 AND
 .FLD3 = FLD3)
/SHOW='.FLD1 "  " .FLD2 "  " .FLD3 "  " .FLD4'

;;FLD2;;

/RSE_RECOG=appl_entry.FLD2 WITH (.FLD1 = FLD1 AND
 .FLD2 = FLD2 AND
 .FLD3 = FLD3)
/SHOW='.FLD1 "  " .FLD2 "  " .FLD3 "  " .FLD4'

;;FLD3;;

/RSE_RECOG=appl_entry.FLD3 WITH (.FLD1 = FLD1 AND
 .FLD2 = FLD2 AND
 .FLD3 = FLD3)
/SHOW='.FLD1 "  " .FLD2 "  " .FLD3 "  " .FLD4'

;;FLD4;;

/RECOG=appl_entry.FLD4

;;ANY_TABLE;;

/VALID=OA$TABLE:"A,C,H,V"
/RECOG=OA$TABLE:"A = Active,C = Closed,H = On Hold,V = Void"

;;ANY_DATE_FIELD;;

/VALID=CAL$CHECK_DATE:"ANY_DATE_FIELD"/OPTIONAL
/POST='DATE_CONVERT ANY_DATE_FIELD, ANY_DATE_FIELD, 1'

;;.DATE & TIME;;

GET OA$FUNCTION = "GET " OA$FIELD_NAME " = OA$DATE"

;;.COMMA;;

GET OA$FUNCTION = 'GET ' OA$FIELD_NAME '= #FIELD_BUFFER'

;;.HYPHEN;;

GET OA$FUNCTION = 'GET #FIELD_BUFFER = ' OA$FIELD_NAME
\GET OA$FUNCTION = 'GET ' OA$FIELD_NAME ' = ""'

;;.GOLD HYPHEN;;

GET OA$FUNCTION = 'GET #FIELD_BUFFER = ' OA$FIELD_NAME
```