Computer Assisted VIEW, IMAGE & SPL

Norman A. Hills
N.A.Hills Computing Services Limited
336 Piccadilly Street
London ON Canada N6A 1S7

Introduction

VIEW provides an effective user to terminal interface, IMAGE is an efficient Data Base for organizing data storage, and SPL is a versatile language with which to build a busness system. DATA ELEMENTS must be defined in each of VIEW, IMAGE and SPL conforming to the various rules which are different for each sub-system, as illustrated in Table I.

Table I

DATA ELEMENT COMPARISONS

Sub-System	VIEW	IMAGE	SPL
Data Element	FIELD NAME	ITEM PART	IDENTIFIER
Alpha first Char and A-Z, 0-9 or:	_	+-*/?'#%&@	'
Numeric Types	DIG NUMn IMPn	K1 I1 I2	LOGICAL INTEGER DOUBLE REAL LONG
Character	CHAR	Xn	BYTE (char) LOGICAL (words)

While the programmer must live with these differences, he must also maintain a conformity between the sub-systems so that the Data Elements do not become damaged or distorted during any transfer between the sub-systems. This need for intra-system conformity complicates the task for system maintenance particularly when there is a need to revise any of the characteritics of a data element.

The presence of the data element in THREE distinct systems requires that each data element be defined THREE TIMES. For anyone concerned with labour efficiency or cost effectiveness there is an obvious redundancy of effort associated with THREE definitions for the same data element.

There are many packages available that address in various ways this issue of programmer productivity. Our approach has

been to make more effective use of the existing resources that are available in the standard HP3000 utilities.

VIEW

Most systems start with the development of VIEW screens as samples for the user to review his visible interface to the system.

RUN ENTRY.PUB.SYS

requires the user to identify the name of the FORMS file and a BATCH file. The potential user can then experience the Field Edits and the other features of VIEW that can be included in the design of the screens.

As a later step, when the database has also been created, the user can employ DBENTRY from the CSL libraries to interact between the VIEW screen and the IMAGE database and it can be very useful to have this type of practice prior to the development of any application programs.

The information about each of our Data Elements that has been stored in our VIEW forms file is of course very pertinent to the balance of the program development. It would be very helpful to be able to have the computer convert some of this information rather than have the programmer create manually the corresponding data elements for the other sub-systems.

Existing VIEW Intrinsics in Table II can be used to retrieve pertinent information relative to each Field Name.

Table II

VIEW INTRINSICS

INTRINSIC	WORD OF BUFFER	INFORMATION RETURNED
VGETFILEINFO	5	Number of Forms in File
VGETFORMINFO	3-10	Form Name
	12	Number of FIELDS in Form
VGETFIELDINFO	11-18	Field Name
	20	Field Number
	21	Field Length
	25-26	Data Type of Field

To obtain complete information from the Forms File, the number of forms from VGETFILEINFO establishes the number of iterations required for VGETFORMINFO, and the number of fields from each VGETFORMINFO establishes the number of iterations required for VGETFIELDINFO.

For this retreived VIEW information to be applicable to IMAGE and SPL we have replaced the screen design identifier in the FIELD MENU during forms design with a FIELD NAME that follows our particular conventions established for IMAGE naming of SETS and ITEMS.

- Each Field Name becomes Itemname_Setname in the VIEW formsfile. In View we are limited to the underscore as the only permissible joiner, and this has the unfortunate feature of becoming invisible in the forms file listings.

- Although the JOIN utility of QUERY uses the convention of Setname.Itemname, we find it much more convenient during program preparation and program maintenance to be able to have the Item as the primary key of any sort, so we are using the Itemname first.

- Each ITEM and SET name will be a maximum of 5 characters.

This information from the Formsfile can be re-arranged to develop a significant start for the IMAGE database DBSCHEMA as:

- A sorted list of ITEM names.

- A sorted list of SET names.

- A list of ITEM names that are associated with each SET name.

- A highly probable identification of the ITEM Type which we would usually translate as follows:

Table III

DATA TYPE TRANSLATION

VIEW TYPE	FIELD LENGTH	IMAGE TYPE	SPL TYPE
DIG	< 6	K1	LOGICAL
		or I1	INTEGER
	> 5	I2	DOUBLE
NUMn	< 6	I1	INTEGER
	> 5	I2	DOUBLE
IMPn	< 6	I1	INTEGER
	> 5	I2	DOUBLE
CHAR	n	Xn	LOGICAL

- An extraction of the PROCESSING SPECIFICATION associated with each Field is not available through an existing intrinsic. Our only solutions so far to this desire, is to copy the formsfile listing to a disc file, scan it for the occurrence of the Field Name, and then extract the subsequent text of PROCESSING SPECIFICATION.

IMAGE DBSCHEMA

The skeleton of Itemnames and Setnames followed by Items of the set can be enhanced by the programmer to include comments and any additional items or sets that have not originated as a View Form Field.

For those who would like to write the DBSCHEMA as their first step, is is unfortunate that there are no intrinsics that will help to develop VIEW from the DBSCHEMA.

RUN DBSCHEMA.PUB.SYS is used in the conventional way to create the ROOT FILE for the Database.

RUN DBDERIVE.PUB.SYS is used to create the BASENAMEnn dataset files for the database, and at the same time create the files of declarations and code that can become part of the subsequent SPL programs via appropriate $INCLUDE statements.

This process is quite fully explained in the June '87 issue of Interact article titled DERIVATIONAL PROGRAM CODE. To

avoid repetition, we will concentrate here on the aspects that have been incorporated subsequent to this reference publication.

In the above reference we descripe the contents and purpose of four files created by DBDERIVE:

basenameDC

basenameGL

basenamePC

basenameZX

We now have DBDERIVE create a fifth file named basenameSP which is a BTREE file containing the correct spelling for all the known expression elements that may be referenced in the source code, as follows:

- BasenameGL contents of identifiers
- INTRLIST file of valid SPL Intrinsic names
- All of the SPL Reserved words.

This BTREE file basenameSP acts as a DICTIONARY of valid words that may be included in the SPL Program Code.

EDITOR entry of SOURCE CODE

The HP EDIT3000 can be customized to execute up to 3 user interfaces by using the initiation command:

RUN EDITOR.PUB.SYS;PARM=16

These user interfaces can be located in an SL at either the SYSTEM ACCOUNT or GROUP level, and can be invoked at either:

- INITIALIZATION to set up files or processes
- COMMAND phase so that whenever any test is entered in response to a / prompt, the users' procedure will be executed and may execute user defined special commands.
- ADD phase will be executed whenever any text is entered in response to a line number prompt.

We make use of this feature of EDIT3000 by having INIT Activate a Process which will receive via a message file during the ADD phase, a copy of each line as it is entered.

This background process takes each word of the entered line and searches the BTREE spelling Dictionary file basenameSP to determine if the word is valid. If a line contains any word that is not in the BTREE dictionary, then the offending word is surrounded by the escape sequence for blinking inverse video and the whole line is returned directly to the $STDLIST screen.

By having the spelling check performed by a background process, it does not slow down the interactive response of the terminal to each terminating line feed. Only the offending lines are returned to the $STDLIST screen and although the offending lines may not appear until two or three lines after they were entered, this is still far more efficient and productive than wating until the first compile to be made aware of transpositions and other spelling errors.

If the user is presented with a blinking word that is correct, then this is an effective reminder that the user should have the word apprpriately added to the program DECLARATIONS, at either the LOCAL or GLOBAL level, and have the word added to the BTREE file basenameSP so that the background process in future will return only lines that contain genuine omissions.

SPL COMPILES

Now that we have been using these techniques for about two years, we have settled down to a few conventions of convenience.

PROGDEV user is the program developer, with AL,PH,MR capability, home is the program testing group PROGTEST and the source code for development versions is in the group TSTLIBRY. The programs for testing are compiled as appropriate:

1/ Following any database revision, by a JOB STREAM which includes all the required steps such as:

   CODIDENT to create the currently required contents for all of the $INCLUDE files that are part of the source code listings Since all the modules will require compiling, this stream will also PURGE USLname and BUILD USLname.

   SPL textname.TSTLIBRY, USLname, $NULL to compile all of the source code files into the USL file.

   PREP and Save the compiled program.

2/ UDCname textname
SPL !textname.TSTLIBRY, USLname, $NULL
PREP USLname, $newpass; MAXDATA=nnnn; CAP=IA. BA. PH, MR
PURGE Progname
SAVE $OLDPASS, Progname

UDCname textname as a UDC with only one PARM which can be used to compile any source text from TSTLIBRY to USLname for PREP and SAVE of each modification during testing. It is important to recognize that any program change that intriduces global variables not previously used, will require the Job Stream to perform a complete recompile with the expanded $INCLUDE of Global variables.

LBRARIAN user is the ACCOUNT LIBRARIAN, with AL,PH,MR capability, home is the group LIBRARY and the source code for production versions is in the group LIBRARY.

In addition to this, we have a group OLDLIBRY which will contain a copy of the most recently replaced production source code, and a group named OLDPROG which will contain the most recently replaced PROG code. The steps to be performed when a new version has completed its test and is ready for production include:

3/ REN textname
RENAME !textname.LIBRARY, !textname.OLDLIBRY
FCOPY FROM = !textname.TSTLIBRY; TO = !textname.LIBRARY

REN is a UDC for quickly renaming source codes files that are about to be replaced.

4/ An expanded version of the job stream in TSTLIBRY is maintained as a SYSTEM job so that it can include the commands to DEALLOCATE before and ALLOCATE after the complete SPL and PREP of the production version of the program code.

As each new module is created, it is added to both of the job streams so that the updating of any production version can be accomplished with a minimum of instruction to the computer.

DATA TRANSFERS

Every application of VIEW and IMAGE requires program code to effect a data transfer between the Form Field and itemname'setname. Character strings are a reltively simple one to one transfer, but each of the numeric fields require a translation between the character format of the form field contents and the binary format of the database itemname. We

have simplified this transfer through the use of VGET'TYPE AND VPUT'TYPE intrinsics of our own creation. These intrinsics operate on three INTEGER ARRAYS.

- The first array contains the field numbers for each window that is to be transferred, and the process is termination by a '0' as the field number.

- The second array contains the same number of integer elements, and each integer identifies the type of conversion that is to be performed during the transfer.

- The third integer array again contains the same number of elements, and the element is the word address of the location on the stack for the database item.

The creation and keying into the program of these arrays is tedious and easily subject to error. Since all of the variables were identified during the run of these utilities, we can readily have the three integer arrays and their assigned values computer created and ready for inclusion in the programs by a $INCLUDE statement.

SUMMARY

Our approach has been the application of creative laziness to a more effective utilization of existing resources that are available in the standard HP3000 utilites. While the product of our efforts may be of some interest to other users of IMAGE with SPL or VIEW, we feel that telling the story of how our shortcuts have evolved, could be an inspiration to others who should be looking for opportunities to implement savings within their own particular environment.