A Universal Approach an an Alternative to Conventional Programming

Bill McAfee and Craig Winters
Futura Systems
Austin

Some two years ago we set about to find a shortcut to programming, a way to simplify and speedup the actual coding, to eliminate all or nearly all of the housekeeping, and to improve the reliability and maintain-ability of our work. We wanted to be able to deal with any problem in terms of the logical operations to solve it, rather than with a sequence of detailed programming statements.

We identified approximately 100 routines to handle input, validation, conversion, formatting, and other functions not provided in the System Library. We designed an English-like language and compiler to invoke these operations as well as those in the SL and to pass them parameters; and we designed a driver to execute all operations in a reliable, consistent manner.

Our primary objectives were:

- to define data types by the significance of their contents (date, phone, zip code, quantities, monetary amounts, etc.) and to perform data entry, validation, conversion and formatting automatically, regardless of storage type.
- to provide a very high-level English-like language that would be both easy to learn and self-documenting.
- to be able to use any number and type of files simultaneously including multiple databases, datasets, KSAM, printer, etc.
- to automatically store and load tables to supply values needed at run-time.
- to simplify declarations and eliminate the dull, boring redundant part of programming, where most errors are made
- to provide text specification syntax, including literal text, program variables and control characters, for use as program messages, report output, headings, etc.
- to work equally well for interactive and batch applications.

We wrote the system in SPL. It has been in daily operation for just over two years, during which time there have been two major rewrites and many additions and enhancements designed to further simplify its use and improve performance. Presently we are just putting the finishing touches on the final version which will incorporate all the things we have learned from these past two years of use and will reflect at every step what we feel will be the best design and coding available.

Since this is a new and unique approach to programming, there is no generic for it. We call it The Futura System, and it consists of a language, compiler, driver and an extensive procedure library. We have attempted to give it the ability to do anything, and when we have discovered something it would not do, we have added it. And while our primary intent was to use it for applications programming, we have found that it is equally strong and valuable as a powerful, versatile utility that is able to supplement and round-out the various system utilities quite handily.

Programming using FUTURA consists of Initialization Commands and Mainline Commands. The compiler reads and validates these, checks their parameters, provides default values where desirable, builds the Mainline binary command module, and formats and prints a program listing in one of several styles. The binary command module resides in the data stack and drives and controls the entire program execution.

InitCommands include:

STACK — which sets the total space the program will require. It has a default value of 3500 bytes, which will handle most utility needs as well as quite a lot of applications.

ALLOCATE — which dimensions the various buffers, should the defaults not be quite right.

BASE — used to open an IMAGE database.

SETS — for identifying the DataSets to be accessed.

FILE — for opening MPE and KSAM files.

TABLE — declares and loads a table, taking care of data conversion, statistics and storage automatically.

PRINTER — opens a printer file according to your specifications, including headlines, page numbers and location, forms-message, and all other parameters used with the line printer.

LOAD — which initializes any area in the data stack with any string or binary value.

INTEGERS — used to load a string of binary single-word integers at any location.

IDENTIFY — an InitCmmd that may be used or implied by the syntax, it sets up a table of identifiers for use throughout your program.

All InitCmmds that may be required must precede the Mainline.

Mainline Commands are names of logical operations such as ADD, MOVE, UPDATE(datatype), BINARY, etc. They may have up to five parameters, some of which are required and some optional. There are MainCmmds to do everything, and frequently there are several, giving the programmer meaningful options on how to accomplish a step. For interactive applications there are a dozen-or-so UPDATE (datatype) commands, such as UPDALFA, UPDNMBR, UPDZIP, UPDSSNO, etc., which not only accepts, validates, formats and displays, and stores the data, but also gives the programmer complete control and recognizes up to 8 special characters that permit backing up one or more fields, begin record over, check for mail, etc.

The fact that commands are the names of logical operations rather than language requirements means that when you have logically solved the problem you have also largely written the program.

Many MainCmmds return one or more values to the program such as the Condition Code, Length, DBStatus, Returned Value, etc. as may be needed.

Text strings to be used as prompts for interactive operations are passed automatically to the program, as the compiler counts them and stores them together with any control characters needed to handle the screen and make an eye-appealing presentation. Text needed for any other purpose is also passed, counted, stored, and recalled with little or no effort on the part of the programmer.

The MainCmmds themselves, the Identifiers, and the way the text strings are handled all provide a great deal of self-documentation right where it is needed in a program, and other documentation and comments may be added at any point. There is an index-building facility that produces an index for the documentation consisting of the program name and all of the comments in each program.

Many commands provide for testing and branching. They are processed uniformly by a subroutine, and branching may be either to a label or to another instruction. Subroutines may be nested up to 20 deep; they may call themselves, and they may reside anywhere in the Mainline. There are both Init and Mainline $INCLUDE commands, allowing routines to be stored separately where they may be used by several programs by including only the reference table.

The binary module together with any tables and initial values may be automatically saved and used again without recompiling by simply adding "$" to the STACK command ($STACK). This binary file may be purged any time changes are made, and it will be recompiled and saved at next compile if the "$" is in place.

In the handout pages we have included examples showing the program file as it is keyed using EDITOR, the normal program listing provided by the compiler which formats this Editor file and prints the permanent documentation, and a look as the terminal screen as each of these programs would appear when run, and a sample of the printer output where applicable.

These are some of the programs that were used to produce the Proceedings and the Exhibit and Conference Guide. While we asked that the papers for the Proceedings be keyed in cap and lower case using the EDITOR, with standard 72-byte records, the facts are that everyone used his/her own method — with record lengths from 60 to 160 bytes and some embedded control characters that would completely snarl our typesetting computer if they were not removed.

These are mostly small, simple programs that will illustrate the truly universal nature of the Futura System as a powerful and versatile utility. I have also brought the documentation for the Automatic TimeSharing Accounting and Billing System (ATSABS) which will show how it can be used for a large, complex system.

This will also show the automatic indexing and system documentation features that are available. We would be glad to have you all look this over and discuss it either at our booth or at other times and places by arrangement. This system totally automates our TimeSharing accounting and billing. It required approximately 5,000 lines of FUTURA code, and we estimate it would have required more than 30,000 of SPL.