The MECCA Source Code Capture Utility

Alex M. Waller
Fermilab
Accelerator Division Controls Department
MS 347
P.O. Box 500
Batavia, IL. 60510 USA

HISTORICAL BACKGROUND

In the early 1980s the Accelerator Division Control System underwent a major upgrade[1]. Until that time almost all the software for the control system was written by the controls group staff. This was largely because the means by which programs were entered into the control system were not readily accessible to most people outside the controls department. Programs previously were entered through magnetic tape and punched cards. Much care needed to be taken in entering an application in this fashion, as even editing of source was done with commands on punched cards and source on magnetic tape. This mode of operation was to be eliminated as the controls development environment became interactive. The move was from a system with a batch monitor operating system to one with a multitasking operating system that allowed multiple users access through CRT terminals. It was anticipated that with such an environment non-controls programmers would be writing applications for the newer Accelerator Division control system.

Until that point in time in the early 1980s, there was little need to keep track of application source code for the various programs that ran the accelerator. The source was all vaulted in rows of tape racks and trays of punched cards. Now source would be edited interactively and exist dispersed on rotating magnetic disk storage. It became desirable to archive all the vital code for the accelerator in some centralized fashion rather than having code reside within the private accounts and disk allocations of the various users.

There was yet another strong argument to keep all the source code centrally located. Since the control system was undergoing conversion and expansion both in hardware and software, it was necessary to be able recompile and relink all applications under certain circumstances. This could only be successfully accomplished if the source code was all centrally located.

As a result of the concern for source code management with the arrival of the newer control system, an in-house code capture system called APL was written[2]. It was based on the observation that most applications were not written by teams of programmers but rather by one or possibly two collaborating individuals. To this day this is largely true of all applications that run the accelerator. The utility simply copied all source code from a user's directory to an APL directory. Compiling and linking were automated but the necessary commands still were needed to be composed by the person writing the application.

The source code capture utility was very successful but had some limitations that became more annoying with time. Initially it was written in the VAX VMS script language DCL[3]. Since the commands were interpreted, APL would run rather slowly. Users could not write their own library procedures and hence a wealth of useful procedures could not be shared by all. The source code capture utility only applied to console applications. All task building, linking and overlaying instructions had to be defined by the user. For a large application this last point took quite a bit of expertise.

In the late 1980s an APLII was created to address some of the growing problems. The actual code capture utility was written in FORTRAN rather than DCL to speed up execution time. This was also the first try at integrating a commercial product in helping manage the source code. The VMS product CMS (Code Management System) was used[4]. Individual modules within the CMS database were compared and source code differences were logged in the CMS database. This allowed the reconstruction of any previous version of code. Unfortunately the price paid for this flexibility was execution speed. APLII never went far beyond beta testing because source code updating was too slow for users[5].

THE NEXT GENERATION

History and experience were the teachers for a new generation of source code capture systems. It was noted that archiving all sources associated with an application was adequate throughout the lifetime of the earlier source code capture systems. Include files from various system resources needed to be allowed while still other include files (other than those of the programmer) needed to be disallowed to contain applications within a scope of available, reconstructable code. Also, cataloging the owner of an application and maintaining a description of it proved invaluable to operations.

As more users programmed over a period of time, useful features became obvious. The next generation source code capture system had to address the current one's limitations. User libraries needed to be allowed and captured. These libraries had to be available for all other users also. Programming tasks other than just console applications should also have their source code captured. The user had to be freed from having to specify the program compilation, include file dependencies and module linking order. This process had to be fully automated so that the task building process was very simple as far as the user was concerned.

What was also desired was more functionality and flexibility for the source code capture utility. The utility should allow program development to go on within a user's default directory but without always archiving the modified source. Also the source capture utility itself should be designed in such a way as to be able to extend its functionality without having to recompile, link and install a new image thus disrupting user work every time. Modification or fixes "on the fly" to the source code capture utility was a desirable maintenance feature.

THE FEATURES OF MECCA

All the above mentioned desires were implemented into the new source code capture utility called MECCA (Management Environment for Controls Console Applications)[6]. MECCA did not stop with archiving only console applications, although this was its initial goal. User libraries and service applications also are captured. A service application makes use of accelerator console libraries but typically does not do any console I/O. These applications are typically servers or monitor applications.

MECCA incrementally builds applications by using the VMS commercial product MMS (Module Management System)[7]. This approach rebuilds an application project much more quickly. An entire application can be rebuilt if necessary by specifying a parameter to MECCA. Also one can retreat back to a previous version if required.

The application project directory can be listed by another MECCA parameter. Extra diagnostics can be turned on and inserted into the resulting log file of the application build. These diagnostics have proved invaluable in tracking down MECCA and user problems. Users can "announce" changes in their application through a mail list that is built from other users who "subscribe" to any number of such MECCA announcements for console applications and service applications.

Users can build VMS help files for the console or service application as well as libraries. Any user can get help on any application as needed. Author and history information are available on each application. If the maintainer of an application should change, modifications can be made to the author information that is kept in MECCA.

HOW MECCA WORKS

Having dispensed with the historical background that led to the development of MECCA and expounding on the features that were attained with it, a description of the workings of this program is in order. A conceptual diagram of MECCA appears in Figure 1. A more detailed and philosophical account appears in the final section of this paper.

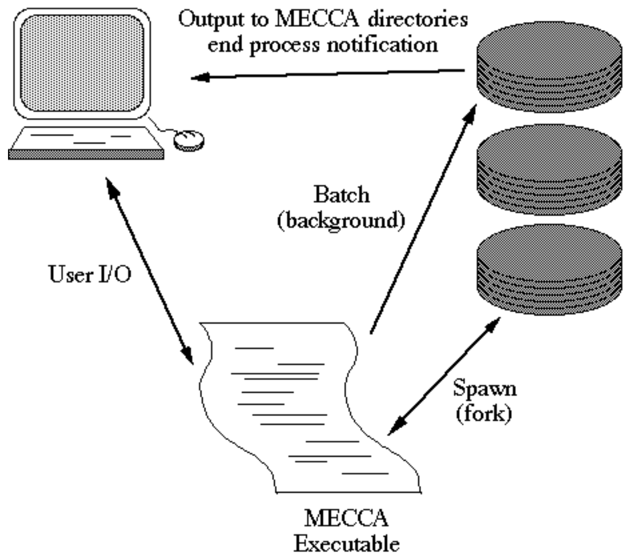

Figure 1
Conceptual Flow of MECCA Control

Any source code modification process begins when a user "checks out" the MECCA source of an application or library or service by performing a MECCA COPY command. All pertinent files for the application are copied into the user's default directory. Since MECCA is running under a VMS environment, the command would appear as a VMS command. Thus the syntax would be: MECCA/COPY nnnnnnnn; where nnnnnnnn is the program, library or service name. All other MECCA commands take a similar format. Remember from the historical note that individual applications within the Accelerator Division controls department are mostly developed by one or two people. A more elaborate check-out mechanism beyond the simple source copy is not necessary.

The next step is to build the application (after the appropriate editing has been performed). The form of this command is generally: MECCA nnnnnnnn; again where nnnnnnnn is the application, library, or service name. The user generally needs not specify more VMS-like parameters unless (s)he wishes to do something special. For example, one may wish to force a recompile of all the source with the /COMPILE_ALL parameter. One may wish to have extra diagnostics placed within his/her log file with the /DIAGNOSTICS parameter.

A simple file date compare is performed on the current MECCA modules and the modules located within the user's default directory. New and modified files have their include files scan to check for any include files that are not allowed and to build up a list of dependencies for each new or modified module. The dependencies that are found are used to automatically generate an MMS (make) file for the application.

While MECCA is moving files into the MECCA directories and the source is being compiled and the image linked, a simple lock mechanism prevents others from trying to perform any MECCA operations on this application. A simple lock is created with a file. This mechanism is sufficient to prevent conflicts during this critical section within the MECCA operation on an application.

At this point MECCA copies all current working source from a user's default directory to an appropriate sub-directory located within the MECCA root directory. If this is a development phase (specified with a /DEVELOPMENT parameter) then no MECCA source will be copied to the appropriate directories within MECCA. The build operation is then started as a batch (background) process so the user may go on to do other things while the application, library, or service is being built. Within the batch process, it is determined if this is a console application or service or a library. If this is an application a linking process is involved. If this is a library the necessary library module is created. A log file, possibly with diagnostics, is produced by the batch process and is placed in the user directory where the initial MECCA operation began.

If no errors have occurred, the old source code maintained within the MECCA directories is moved to a directory with the text OLDn appended to it and the successful build is left in the current MECCA directory for the application. The n in the OLD text is incremented for each revision number. If there is an error, the old source maintained by MECCA is kept within the current MECCA directory for the application, and the version with errors is kept in a directory named TEMP for the user to peruse. Figure 2 details the MECCA directory structure. At the end of the MECCA batch process the user is informed of the state of the build operation.

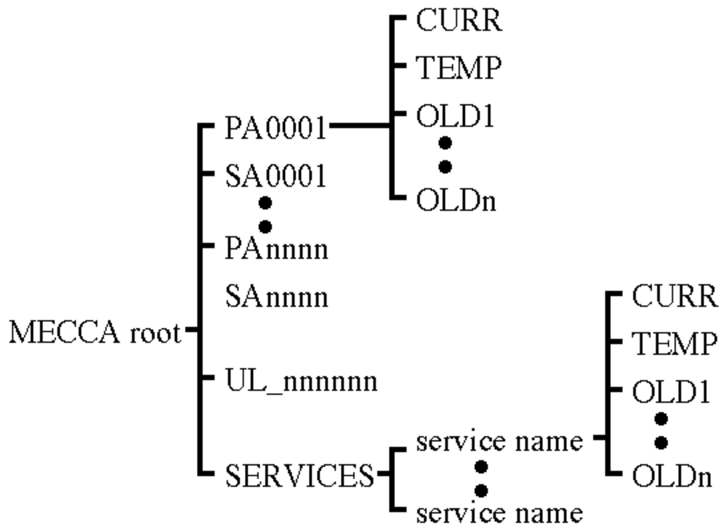

Figure 2
MECCA Directory Structure

THE MECCA PHILOSOPHY

User interface designed to be simple

Since the possibility of using character cell terminals existed, the MECCA interface was kept to be textual and command driven. Also, on a careful analysis of what was required, MECCA was better suited to be command driven. There is very little involved that would require a more sophisticated graphical user interface for selecting input options, especially if the default options were chosen wisely. As mentioned earlier, MECCA defaults are such that what the user would "normally" want to do requires no extra MECCA parameters to be specified.

While the user is within the MECCA monitor process very little user feed-back (prompting) is required. For entering a new library or application one needs to enter a line of text giving a description of the library or application. If one is retreating to an older version MECCA needs a version number to retreat to. If previous source code for an application was captured from a default directory different from the current default directory or modified by a user other than the current user this information is displayed and one is prompted to continue or abort. Finally, if all goes well, the user is asked if (s)he wishes to submit the application for the batch build operation.

Monitor process co-ordinates flow of control

Command input parameters are processed by a separate module of the MECCA monitor. This allows an easier port to other platforms where a non-VMS style of command parameters would be more acceptable. As mentioned in the previous section, the monitor will solicit any necessary user input. For new libraries and applications the monitor creates the necessary directories from the MECCA root directory; for console applications the directory name is derived from a sequentially assigned number; for libraries the directory name is prefixed with the string "UL_"; for service applications the service name is used directly.

The monitor process is responsible for spawning (forking) processes. Some of the processes that get spawned are: Help, which executes the VMS DCL HELP command; Directory, which executes the VMS DCL DIRECTORY command; Copy, which executes the COPY command; and HISTORY, which executes a VMS TPU[8] section which evokes an editor to browse the history file of a given application. When a process is spawned control returns to the MECCA monitor process. A spawned process is generally one where a single VMS command line can be executed, a brief VMS DCL script can be executed, or a brief piece of auxiliary code such as C or TPU can be executed.

The monitor process is also responsible for submitting batch jobs (background tasks). The background jobs that get submitted are: Build, which is the primary functionality of MECCA; Help, which builds the necessary VMS help from the extracted comments within a source program; Retreat, which retreats to a previous version of an application in MECCA. A Batch process is almost always the end result of a MECCA operation. Batch processes are tasks that are lengthy, complicated, or time consuming.

Auxiliary operations are kept separate from monitor process

Since many MECCA tasks are relegated to spawned or batch jobs, MECCA is highly modular. Only new functionality requires adding information to the monitor process so that the monitor can co-ordinate the execution of a new task. MECCA behavior can be altered or extended without compiling, linking, or installing an executable image with every instance of required change. This modular approach allows a tight granularity on maintenance. Generally only a small section of code or script has to be modified or debugged and fixed since all functionality that needs to be changed (or fixed) is entirely self-contained within a single module or script. Also, new functionality that is added can be developed and tested independently from the current running version of MECCA.

The many MECCA modules are written in a variety of ways. What is used largely depends on the situation. The MMS (make) generator is written in a powerful string processing language called TPU on our VMS system. This was most appropriate since much string processing is needed. The included scanner is written in C. Many of the other procedures are DCL scripts which take advantage of the command facilities of the VAX VMS operating system.

CONCLUSIONS

MECCA has adhered to the simplicity of earlier source code capture systems but yet was able to achieve the flexibility necessary for the growing needs of the programming community of console applications and services. It has proved fast and efficient for the given environment of the Accelerator Division control system. The decision to keep MECCA code itself modular has greatly aided in debugging and developing MECCA rapidly thus keeping maintenance controlled and to a minimum.

ACKNOWLEDGMENT

I would like to thank Jim Smedinghoff for his valuable suggestions on the general design direction MECCA should follow. Jim also provided the TPU procedure that generates the make (MMS) file for any given application, library or service.

REFERENCES

[1] D.Bogert, The Fermilab Accelerator Control System, Nuclear Instruments and Methods in Physics Research, Volume A247 (1986), pp. 8-24.

[2] A. Thomas, D.Baddorf, K. Cahill, D. Rohde, J. Smedinghoff, L. Winterowd, User's Guide to ACNET Console Systems, Fermilab Internal Report, Software Documentation Memo 62.3 (1985), Chapter 6.

[3] OpenVMS DCL Dictionary, Digital Equipment Corporation, 1995.

[4] DECset, DEC Code Management System Reference Manual, Digital Equipment Corporation, 1995.

[5] Private communications with Robert Joshel and Brian Hendricks.

[6] A. Waller, MECCA; the Management Environment for Controls Console Applications, Fermilab Internal Report, Software Documentation Memo 208 (1994).

[7] DEC3GL Implementation Toolkit for VMS, MMS Description Generator User's Guide, Digital Equipment Corporation, 1992.

[8] DECTPU, DEC Text Processing Utility Reference Manual, Digital Equipment Corporation, 1993.