Fermi National Accelerator Laboratory

TM-1463
2390.000

A Multiple Node Software Development Environment*

P. Heinicke, T. Nicinski, P. Constanta-Fanourakis, D. Petravick, R. Pordes,
D. Ritchie, and V. White
Computing Department
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

June 1987

*Presented at the Fifth IEEE Conference on Real-Time Computer Applications in Nuclear, Particle, and Plasma Physics, San Francisco, California, May 12-14, 1987.

TM-1463

A Multiple Node Software Development Environment

Peter Heinicke, Tom Nicinski,
Penelope Constanta-Fanourakis, Donald Petravick,
Ruth Pordes, David Ritchie, Vicky White

Fermi National Accelerator Laboratory*
Computing Department / MS120
P. O. Box 500, Batavia, IL 60510

ABSTRACT

Experimenters on over 30 DECnet nodes at Fermilab use software developed, distributed, and maintained by the Data Acquisition Software Group. A general methodology and set of tools have been developed to distribute, use and manage the software on different sites. The methodology and tools are of interest to any group developing and using software on multiple nodes.

Introduction

The Fermi National Accelerator Laboratory (Fermilab) is a facility dedicated to basic research in the field of high energy physics. Research takes the form of "experiments", which are conducted by groups of physicists. The experiments are highly computerized; there are usually one or more minicomputers devoted to the tasks of data acquisition and analysis of the experimental data.

Most experiments have at least one VAX or MicroVAX computer, one or more PDP-11 computers, and possibly a few programmable microprocessors. Many different experiments either actively take data or prepare to do so simultaneously.

The Data Acquisition Software Group of the Fermilab Computing Department provides software and support for the experiments. Experimenters use the provided software to perform online data acquisition and analysis required for their experiment. In some cases, the software is used in a turnkey manner; more often, it is used as the basis for more elaborate and experiment-specific software. In the latter case, the experimenters obtain the basic package and then customize it to their particular needs through their own software development efforts.

Software is usually targetted for PDP-11 or VAX computers. Other targets include microprocessors, such as 68020's, etc. Target computers are physically located at approximately 30 different sites scattered over the 6800 acres of Fermilab. The VAX's and MicroVAX's at these sites are connected to one another via DECnet. These VAX's (or the Central Facility VAX Cluster) are used by the experimenters for software development in enhancing the supplied software as well as for online data acquisition and analysis. Software is transferred to these machines via DECnet from the Data Acquisition Software Group's Development VAX. It is also transferred via magnetic media to the computers not connected via DECnet; (PDP-11's not connected mainly due to memory limitations and microprocessors).

Additionally, the software sometimes needs to be transferred to the collaborating universities and research institutions which participate in Fermilab experiments. The experimenter may then continue software development or equipment testing activities while physically located at the home institution.

With so many sites and so much software in use at these sites, we quickly realized that some systemization of the task of organizing, maintaining, and distributing the software was mandatory. Keeping track of the software at the various sites is a formidable and necessary job. We must be able to offer assistance with the current version of the software at hand.

A requirement on the systemization was that it must support having different versions of the same software at different sites or even at the same site.

While it might be possible in principle to arrange to have the same version of the software at all sites, in practice this does not occur. An ongoing experiment does not necessarily want to avail itself of the latest enhanced version of a piece of software; bugs or side effects may be introduced which might complicate the primary task of monitoring the experiment. Even when an experiment decides that the new features outweigh any risks of complication, it is extremely important that the experiment be able to switch back to the previous version as quickly and reliably as possible.

In what follows, we describe the organization of our software into "Products", how these Products are created, maintained and versioned, and how this Product organization is used in the distribution of software to the target VAX computers, and from there to other target computers when necessary.

What is a Product?

A "Product" is an arbitrary group of logically connected directories and files (stored on a VAX/VMS system) and referred to by a Product name and optionally by qualifying names, such as the Version number, target operating system, or hardware interface. The Product name is a printable ASCII string describing the group in a mnemonic way. For each Product name, there is a single development version of the product and/or one or more distribution versions. It is not necessary that a Product be developed by the Computing Department to fit into this scheme. However, the Product (the directories and files which

* Operated by the Universities Research Association, Inc. under contract with the United States Department of Energy

comprise it) must be organized in a prescribed way. The constraints are relatively minor because we wanted the ability to include all kinds of software as products--not just those developed at Fermilab.

An example of a non-Fermi Product is KERMIT, a communications package. KERMIT_VMS is the Product name for the VMS version of KERMIT.

When the source code contained in the development version of a Product is updated, either for maintenance or enhancement reasons, a new "Version" of the Product is generated. This may occur even if the source code of the Product is unchanged. For example, if a Product is rebuilt using new "versions" of code on which it depends (such as an object library), but which is not a part of the Product itself, a new version of the Product is still generated. A Product version is used to inform the user, developer, and Product maintainer of not only which level of source code of the product it contains but also the entire state of the Product, its dependencies on other software Products, etc.

As a simple example of a Product with different Product versions, consider the COURIER product for VAXONLINE. Version V1.0 of the COURIER Product refers to the first released version of COURIER for VMS. It will normally have a product directory by the name of COURIER_V1_0. Later versions will have similar product directory names, e.g. COURIER_V1_2. If a UNIX version is developed, the support group would need to decide whether to keep the old name. If they decide to, they can rename the two products to be COURIER_VMS and COURIER_UNIX, or leave it as COURIER and COURIER_UNIX.

Products can be divided into two levels of complexity: "simple" and "compound." A simple Product consists of a collection of software which is expected to be used, upgraded to a new version, and distributed to target sites independent of the state of other software Products. The decision to organize a product as a "simple" one is basically that of the developer; it is a statement that this Product is somehow basic and not further made up of component Products.

This does not necessarily mean that the Product was not dependent upon other software external to the Product when it was "built" (compiled, linked, etc.). Nor does it necessarily mean that the Product requires no other software Product in order to function.

For example, many of our Products are written in FORTRAN. These are definitely dependent upon the FORTRAN compiler and the FORTRAN Run Time Library--both of which are external to the product and which (in the case of the Run Time Library, at least) are required in order for the Product to function.

A compound Product is a collection of different component" Products (either simple or compound), frequently used together. These Products do not necessarily have to be dependent upon each other although in many cases they are. They may be grouped together only for ease of distribution of many small Products which change infrequently. Alternatively, they may be grouped together because of dependencies on each other; hence, a change in a component Product would indicate that a new version of one or more of the other components is either necessary or desirable.

<u>Goals</u>

The distribution and installation of the software is only a peripheral (but time consuming) activity. To permit us to spend more time on software development, we have devised a formal specification for "Products" and specialized procedures, whose goals are:

<u>Provide a Uniform Product Specification.</u> The product specification is meant to provide system management tools and the user with a uniform interface to the software we are responsible for. The specification includes:

o *the directory tree structure of the files in a product*

o a list of required and optional files,

o the naming conventions for these files and directories,

o how logical names should be used.

<u>Keeping Track of Product Versions on a System</u>. Different sites use different versions of a product creating a need to maintain a database of which products and versions reside on a particular system. This functionality is provided by a system management tool we call SITE_PRODUCTS.

<u>Simplification of Product Distribution</u>. We need to automate the distribution of versions of products to remote sites (making use of DECnet) and the installation of the products on the target site. Such automated procedures are needed both for efficient use of our time and to minimize the risk of errors or omissions.

<u>Transportability to External Sites</u>. Although restrictions are placed on a products structure and interaction with users (how the product is distributed and how the system manager treats it), it is still necessary to permit the product to be easily installed and used on systems which do not follow our methodology.

<u>Permit Switching Between Product Versions</u>. In order to maintain and improve existing products, and have the new releases accepted by experimenters, there is a need to allow the use of the latest version of a product, but also to instantly and transparently "switch" to using a previous version residing on the same system.

The ability to switch between versions on the same system is also important for product developers and maintainers. A user may discover a bug at a previous release of the product - and the product maintainer is then able to check for the bug in that release just by switching to it. This capability is provided by PRODUCT_SETUP and the database of products and their versions (maintained by SITE_PRODUCTS).

<u>Permit the Composition of a Product to be Known Precisely</u>. We make extensive use of DEC's CMS (Code Management System) and MMS (Module Management System) to control the source code version of a product and to automate the construction of that product from its sources and any other libraries etc. it may be dependent on. (CMS and MMS are similar to the SCCS and MAKE utilities on UNIX). In situations where a product may be dependent on libraries in other products - the specific version of the

library-related products used must be both controllable and forever known. The time-stamps of the individual files as used by MMS are not sufficient to control such inter-dependencies.

The procedures, (which we call BUILD), permit the dependencies of one product on another, either as a part of a compound product, or just as a required but separate piece of software, which must be present in order to build the product, to be expressed in a formal way. From this formal specification the order of creation of the component parts can be determined and the business of creating a very large software product can be automated in a foolproof way.

The Resulting Tools

All the management tools we have developed are written as DCL command procedures. Any language could have been chosen, and the system could have been implemented in a more operating system independent way. DCL command procedures were chosen because of speed of implementation, and because we underestimated the full extent of the project we were undertaking. Future implementations of this functionality will probably use a different tool than DCL, since DCL is so slow, and unmodular. We are considering reimplementing the system in FORTRAN or an inexpensive 4GL.

The remainder of this paper will discuss the concepts and management tools introduced above which together allow us to achieve the goals outlined in the previous section. These include: Specification of a product, use of the BUILD procedures, the SITE_PRODUCTS, DISTRIBUTE and PRODUCT_SETUP procedures.

Specification of a Product

The product specification provides system management tools and the user with a uniform interface to the software. We have written a 50-page specification of a product including the mandatory and recommended requirements thereon. The product specification addresses three areas:

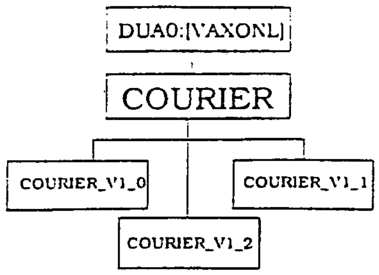
o Directory tree structure and the files in a product.

o Logical names to be defined (associated with the product).

o Required and optional command procedures and how they are used. (definition of parameters).

Directory Tree Structures

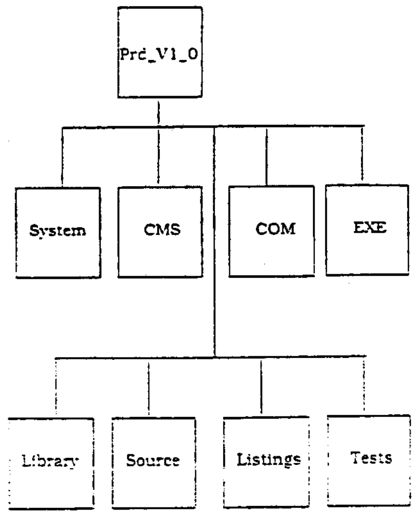
Products reside under rooted directories. Actually, two rooted directories are associated with a particular product. The "Version" Root is the rooted directory for a particular version of a product. This is the rooted directory that a user will see when using a product. Version Rooted directories reside under an "Umbrella" Rooted directory. The Umbrella Rooted directory contains all the versions of a product. However, more than one product and its versions can reside under the same Umbrella Root. For example:



Courier Product Directory

The leaves are products, while [VAXONL.COURIER] is the Umbrella Root. The Version Roots for the products are [COURIER_V1_0], [COURIER_V1_1], and [COURIER_V1_2]. A directory named [VERSION] is necessary for the current implementation of the product tools, and contains information about which product is installed. Future implementations will probably centralize this information with the rest of the database.

For each product version, there is a set of required and optional directories:



Sample Product Version Directory

The [PRD_V1_0] directory is the Version's Rooted directory, while [COM] and [SYSTEM] are required directories, and [MAINT] is an optional directory. Beyond these directories, the developer can use any tree structure (under the product's version rooted directory).

Logical Names

To keep products site-independent, logical names are used to point to different files. All logical names should be defined in terms of one logical name which points to lower level in the directory tree:

'product'$ROOT

which is the rooted logical name pointing to the PRODUCT's Version Root. By changing 'product'$ROOT's definition (with PRODUCT_SETUP), a user can easily "switch" between different versions of a Product. In the example in the first figure, the rooted logical name for COURIER is

```
$ SHOW LOGICAL COURIER$ROOT
   "COURIER$ROOT" =
   "disk:[VAXONL.COURIER.COURIER_V1_0.]"
```

Required Files

The product specification requires that each product provide two command files, of defined logical names, to be implicitly invoked at system bootstrap time and when a user wants to use the product. All products must provide these files in a particular directory for the product version. The specification also recommends a Help file to be provided with each product; this is automatically included in the general product Help library when the product is entered into the SITE_PRODUCTS database.

[COM]SETUP.COM is used to define logical names and symbols on a per process basis. That is, the user invokes SETUP.COM (normally at login time) if there is a need to use the product.

[SYSTEM]PRSTARTUP.COM is used during system boot time (product startup) to define shareable logical names in the logical name table generated for the product, and to perform any other operations which affect the product system wide (such as INSTALLing files, loading device drivers, starting a queue, etc.) and other privileged initialization functions.

Developing the Products (BUILD)

The BUILD procedure is used to construct a product based upon its dependencies on other products. BUILD takes into account that a product may:

o Depend on other products.

o Depend on specific versions of other products.

o Incorporate other products totally within it.

The construction of a product consists of compiling and linking the software comprising the product.

A product developer uses a product Maintenance Language (PML) file to describe how a product is dependent upon other products. Only the immediate dependencies need to be described, since BUILD recursively uses the dependent product's PML files to generate a final list (a product Maintenance Output (PMO) file) which sequentially describes the order in which products should be built (to satisfy all dependencies).

For example, the product KERMIT_VMS is to be "BUILT":

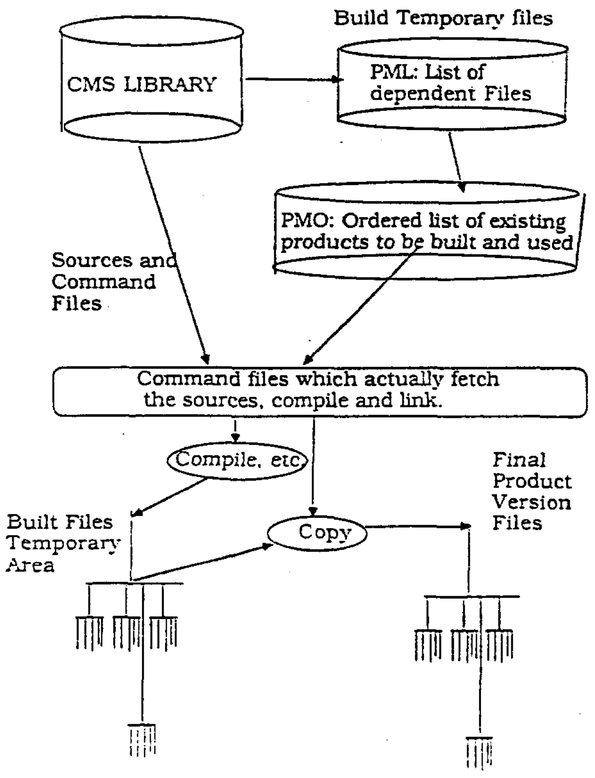
o KERMIT_VMS is dependent upon an another product called GET_PORT

o KERMIT_PDP is dependent upon KERMIT_RT, KERMIT_RSX, and KERMIT_RSTS. BUILD would determine that the products would need to be built in the following order:

```
GET_PORT
KERMIT_VMS
KERMIT_RT
KERMIT_RSX
KERMIT_RSTS
KERMIT_PDP
```

BUILD then will construct the products in the appropriate order to generate the final product. To save time, BUILD will not construct a product if the required version already exists.

The actual details of construction of each of the component pieces are left up to the component piece of software. We normally use DEC's CMS and MMS wherever possible. This is especially useful in conjunction with our methodology of one development version of a product and multiple distribution versions. By having a single CMS library in the development version of each product and creating classes for each source release level we avoid the need to keep the sources with or for each version of the product. We can always recreate any version at any time. This saves disk space and also provides a centralized record of who changed the software and when.


Build Temporary files
CMS LIBRARY
PML: List of dependent Files
PMO: Ordered list of existing products to be built and used
Sources and Command Files
Command files which actually fetch the sources, compile and link.
Compile, etc
Final Product Version Files
Built Files Temporary Area
Copy

Steps in "Build"ing a product

System Management of Products (SITE_PRODUCTS)

SITE_PRODUCTS was developed to keep track of which versions of which products reside on a system. It not only maintains a database of products and their versions, but it schedules the starting up of products at system boot time (or any other time) and the shutting down of products. SITE_PRODUCTS avoids the need for the system manager to change the system specific startup command procedure (SYSTARTUP) every time a product or a version of a product is added, modified, or removed.

Products are made "known" to SITE_PRODUCTS. The "Known Product List" file, maintains this information.

For each known product, SITE_PRODUCTS maintains a "Product Version List" file which resides under the product Umbrella directory. The product developer is able to add, modify, and remove product versions without requiring privileges (only access to the particular product's area is required).

The SITE_PRODUCTS procedures point to the Known Product List using a logical name. Users can use SITE_PRODUCTS to maintain their own Known Product List, and Product Version Lists. This can be extended for use on a VAX Cluster system, where a common Known Product List is used to startup (shutdown) all Products common to all nodes in the Cluster. Then, by redefining the logical name, a node-specific Known Product List can be used to manipulate software products licensed (or useable) only for that particular machine.

SITE_PRODUCTS allows the addition, modification, and removal of products and Versions. These operations only modify the Known Product List and Product Version Lists, not the actual files of the products. When a product version is declared to be the default version on a system, its Help file is included in a general product help library (if one exists) and a Bulletin is posted on the system.

For each product, the Known Product List maintains the product's name, the specification of the Umbrella Root, and other miscellaneous information. Associated with each product version in the product Version List is a directory path from the Umbrella Root to the rooted directory for the product version.

When a product is started up by SITE_PRODUCTS, a shareable (system wide) logical name table is created to contain logical names defined by the product. Then the product specific startup command procedure is invoked. This procedure usually defines logical names, device drivers, starts up queues, installs privileged images, etc.

Using the Products (PRODUCT_SETUP)

The final stage of any product is its use. PRODUCT_SETUP is used to "setup" a product for use by a user. It also allows a user to choose which version of a product to setup. Setting up a product involves the definition of logical names and symbols required for using the product.

A symbol by the name of SETUP is used on all systems to invoke PRODUCT_SETUP. Users of a software Product such as our example KERMIT_VMS simply type

SETUP KERMIT_VMS

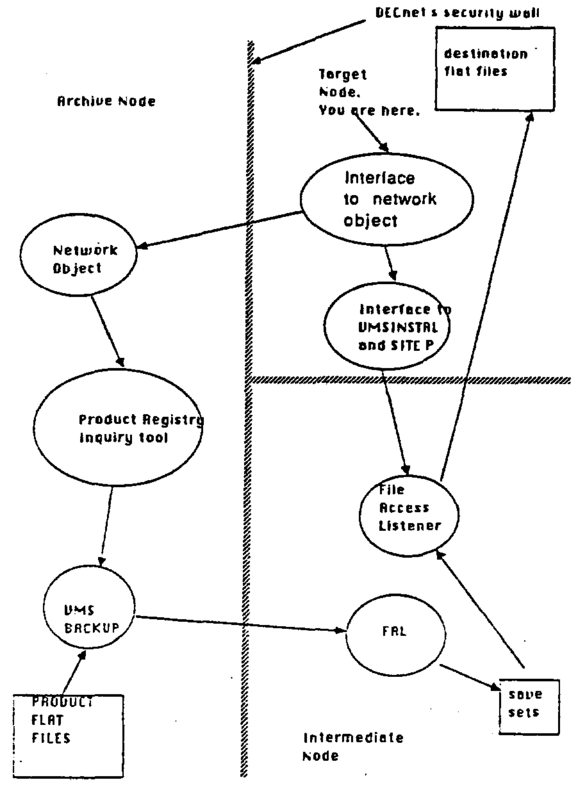
to use the default version of the product and all its component sub-Products.

The ability to switch transparently between product versions is provided by the logical name tables created for the product. When switching between product versions, PRODUCT_SETUP creates a new logical name table (which overrides the old table) and defines the logical names for that particular version. Therefore, different product versions are not required to use the same logical names.

Obtaining the Products (DISTRIBUTE)

DISTRIBUTE provides a system manager on a remote machine the ability to copy products, from an "Archive machine", and install them. Most of the time, DISTRIBUTE is used over DECnet, but it also provides a tape mode, which permits products to be distributed and installed at external sites using magnetic tape as a transfer medium.

DISTRIBUTE interactively queries the user for the information it needs. The questions are self explanatory, so that no documentation is normally required in order to obtain a product. Besides the product name and version, DISTRIBUTE asks where the product should be placed (the disk and Umbrella Root), and whether the product and its version should be declared to SITE_PRODUCTS.



Distribute Processes

When a product is selected by the user, DISTRIBUTE uses that product's Product Maintenance Output (PMO) file (generated during a BUILD) to determine which component products need to be copied over as part of the chosen product. This provides all sites with a complete and consistent view of a product. Products which are not constructed with BUILD and therefore have no PMO file can also be distributed - all files in the directory tree stemming from the product version rooted directory will be taken to comprise the product version.

DISTRIBUTE uses BACKUP save sets compatible with the VMSINSTAL utility (part of VAX/VMS). Because the product conforms to the product specification, only one KITINSTAL file (used by VMSINSTAL) needs to be written for all products. This frees the product developer from writing code used strictly for the purpose of installing a product.

A complete log of software distributed, date, version and to where is maintained on the Archive machine

Conclusions

The organization of products and the procedures described in this paper have been in use for more than a year now. Hundreds of products have been distributed to target sites. The sacrosanct nature of a product version once built has enforced a strict discipline on program development and aided immensely in tracking down complicated problems where any one of a number of hardware and software variables could have been at the root of the problem. The procedures described were first developed for software to be executed on a VAX(VMS). We have found them such a useful aid for distribution, maintenance and archiving that we extended the concepts to cover software for other operating systems in use.

We have found a standard product specification to be extremely useful. Not only has it enabled us to write the management tools described but it has also helped enormously in the ease of understanding, maintaining and supporting our software. New members of the group and new users new to Fermilab can very quickly produce software to conform to the general specifications and obtain and use software that is available. It is much easier for any member of the group, regardless of particular area of expertise to be able to distribute, demonstrate, find bugs in, create a new version of any product. New software products produced elsewhere at Fermilab or at other institutions or vendors can be quickly added to the set of available software and made available in the same uniform way to all the users on site (via the same SETUP command). Following software product "standards" has saved manpower also in enabling us to write general procedures. For example, the arrival of Microvaxes with limited disc space created a need to trim products. A general procedure which omitted all list, map and documentation files from a distribution version could be written because of the standards imposed, thus solving the problem in general for all software which we maintain or distribute.

This entire program of work was undertaken without a proper realization of the size of it - really as a non-serious sideline, which people did a little work on when the need arose. If we were doing it again we would better understand the benefits and scope of the project and would take it further than we have today. The database maintained by SITE_PRODUCTS would be made extensible and easily accessible as a database. Some of the system management procedures would have been written in a high level language instead of DCL. thus increasing both their speed and extensibility.

Acknowledgements

Contributions to the ideas, definitions and procedures have been made at various times by all members of the Data Acquisition Software and DEC Systems Group in the Computing Department at Fermilab - which consist of the authors, David Berg, Eileen Berman, Andy Cohen, Terry Dorries, Arkady Lubinsky, Carmenita Moore, Liz Quigg, Dave Ritchie, Chip Kaliher, Nancy Hughart and Steve Kalisz. We also acknowledge helpful feedback from various users of the system (DISTRIBUTE in particular) ranging from on-site local system managers to experiment participants distributing software over DECnet from Italy.

References

PN's refer to Fermilab "Programming Notes"; IN's refer to Internal Notes. Documentation is available from the Computing Department Program Librarian.

[1] R. Pordes, Data Acquisition Software Group Product Specifications, Fermilab Computing Department Note, IN-141, August, 1985.

[2] P. H. Heinicke, Backup / Distribute Procedure for Product Distribution, Fermilab Computing Department Note, PN-261, September, 1985.

[3] T. H. Nicinski, SITE PRODUCTS / Maintaining Known Products, Fermilab Computing Department Note, IN-140, February, 1986.

[4] T. H. Nicinski, BULLETIN / Maintaining an Electronic Bulletin Board, Fermilab Computing Department Note, IN-141, August, 1985.

[5] P. C. Fanourakis, BUILD Procedure for Product Distribution, Fermilab Computing Department Note, PN-259, July, 1985.

[6] R. Aurbach,Using VMSINSTAL with User-written Applications, Fermilab Computing Department Note, PN-262, October, 1985.

[7] T. H. Nicinski,PRODUCT SETUP User's Guide / Setting Up Products, Fermilab Computing Department Note, PN-269, February, 1986.