SLAC-PUB 5230
April 1990
(E/I)

# DUCS - A Fully Automated Code and Documentation Distribution System†

**A.S. Johnson**

Boston University, Dept. of Physics, 590 Commonwealth Ave., Boston, MA 02215

**B. Saitta**

Università di Ferrara, Istituto di Fisica, Via Paradiso 12, I-44100 Ferrara, Italy

**0. Gervasi**

Universita di Perugia, Dipt di Fisica, Via A. Pascoli, I-06100 Perugia, Italy

**G.R. Bower and A. Rothenberg**

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

**A.P. Waite**

University of Victoria, Dept. of Physics, Victoria, BC, Canada V8W 2Y2

Abstract

The Distributed Update Control System (DUCS) is a code distribution system developed, for the SLD collaboration to distribute code, documentation and news items between remote collaborators and SLAC. The system runs on both VM and VMS systems and is currently running at a total of 18 sites on two different continents, using both BITNET and DECNET connections.

Software updates and news items can be submitted from any site where DUCS is installed and are distributed to all other sites. When an update arrives at a remote site it is installed appropriately without any manual intervention. The details of the installation depend on the type of file, but for source code, installation includes compilation and the insertion of the resulting object module into the appropriate library. Whenever an error occurs the error log is returned to the originator of the update.

DUCS maintains both development and production code, subdivided into an arbitrary number of sections. A mechanism is provided to move code from the development area to the production area. DUCS also contains many utilities which enable the status of each node to be ascertained and any manual intervention necessary to correct unanticipated conditions to be performed.

The system has been running now for nearly three years and has distributed over 20,000 code updates. It is proving a valuable tool for remote collaborators who are now able to participate in code development as easily as if they were at SLAC.

Presented at the 8th Computing in High Energy Physics conference,
Santa Fe, New Mexico, April 9-13, 1990.

† This work was supported in part by the Department of Energy contracts DE-AC02-89ER40509 and DE-AC03-76SF00515 and NSERC, Canada.

## Introduction

Most large High Energy Physics collaborations consist of many collaborating institutes, with the majority of physicists spending a large fraction of their time at their -home institutes. In order to make it possible for these physicists to make a significant contribution to both the analysis of data and to the development of reconstruction and analysis software it is essential that they be given ready access to up-to-date software.

In the past this has often been achieved by making it possible for remote collaborators to log into the computers located at the laboratory where the experiment is located, either using a network connection, or by using a dedicated satellite link. This type of connection is ideal for some types of work but for using 3-d graphics and windowing systems such as X-windows, or for printing large output files or graphics output the bandwidth provided by this sort of connection is far from adequate. In addition this mode of working leaves valuable computer resources at home institutes unused.

An alternative approach is to periodically transfer software to home institutes on magnetic tape. While this makes remote computers, printers and graphics devices available, it suffers from the disadvantage that software transferred in this way rapidly becomes out-of-date, especially during code development and early stages of analysis when software is changing rapidly. This in turn often leads to duplication of effort or work which is obsolete before it is even finished.

To overcome these problems the SLD collaboration has developed the Distributed Update Control System (DUCS) described here. The main objectives of the DUCS system are:

- To make all offline code available at home institutes so that code development and use is as simple at home sites as at the central laboratory.
- To make code changes (updates) possible by authorised users at any site and to make code changes available at all sites as rapidly as possible.
- To support many different code sections (e.g. reconstruction, simulation, calorimetry, drift chamber etc.) and to allow for multiple copies of each section (such as development and production versions) to be kept at each site.
- To accept documentation, help files and news and conference items from any site, and to redistribute them to all sites.
- To support different operating systems and allow for the fact that even sites with the same computer systems may be running different operating system releases.
- To work using existing networks and bandwidths with no manual intervention required at home sites.

## Implementation

In order to meet the objectives described above the following design decisions were made early on. First we decided that we would use BITNET as the transport mechanism, since almost all of our collaborators were already connected to it, including both VM and VMS sites, and also since the store-and-forward nature of the RSCS protocol employed by BITNET made the design of the code distribution system simpler. At the same time we attempted to keep the protocols used by the DUCS system reasonably network independent so that we could extend the system to other networks and operating systems later.

Next we decided that the primary method of transporting software would be by transferring source code. This has the advantage that during code updates only routines which are changed need to be sent across the network, and since individual routines tend to be fairly small this reduces the network bandwidth required. In addition source code can be made operating system independent and system release independent while, in general, object libraries and executable modules are not.

Finally it was decided that installation of files at home sites should be totally automated, since the installation of many source files at each site would otherwise require a prohibitive amount of manpower. This requirement necessitated the design of a system which would know how to install each type of file sent over the network, be it source code, documentation, help files etc.

**Figure 1: Schematic diagram of the DUCS system**

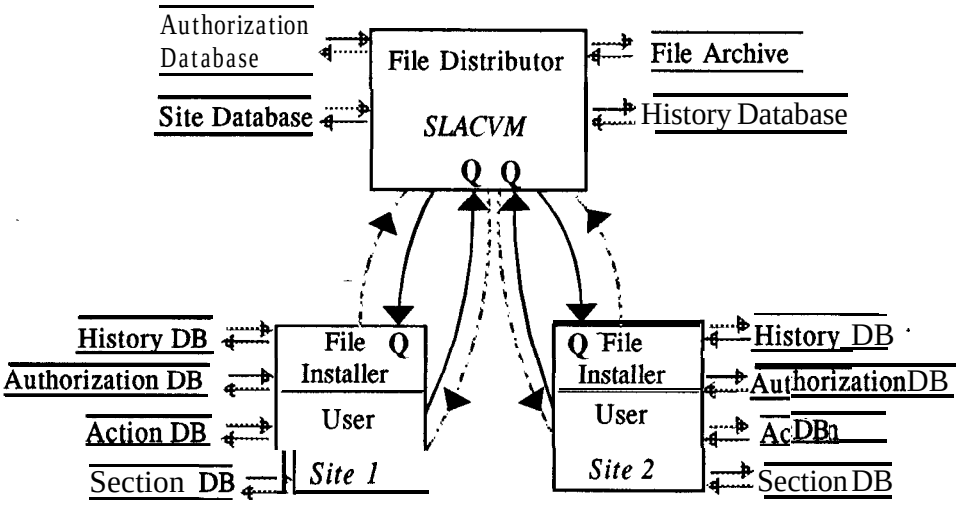



Figure 1 shows in schematic form the overall organisation of the DUCS system. The hub of the system is the file distributor which runs at one site. The file distributor receives updates sent by users from any site. On receipt of a file the code distributor checks that the user is authorised to install the file and if so forwards the file to each site at which it should be installed. The file distributor keeps a history of each transaction and also archives each update so that any file which gets "lost" traversing the network can be resent. At each site a file installer receives the file and installs it appropriately. If, for any reason, an error occurs during file installation (for example if a source file fails to compile correctly) an error log is generated and returned, via the file distributor, to the update's originator. Unlike most other code distribution systems[1] this results in a system that is symmetric in that files are installed identically at all sites, including the central site where the file distributor runs.

**Table 1: DUCS network protocols**

| Format | Purpose |
|---|---|
| DEFAULT | Used for all text files such as source code, news, EXEC files etc. |

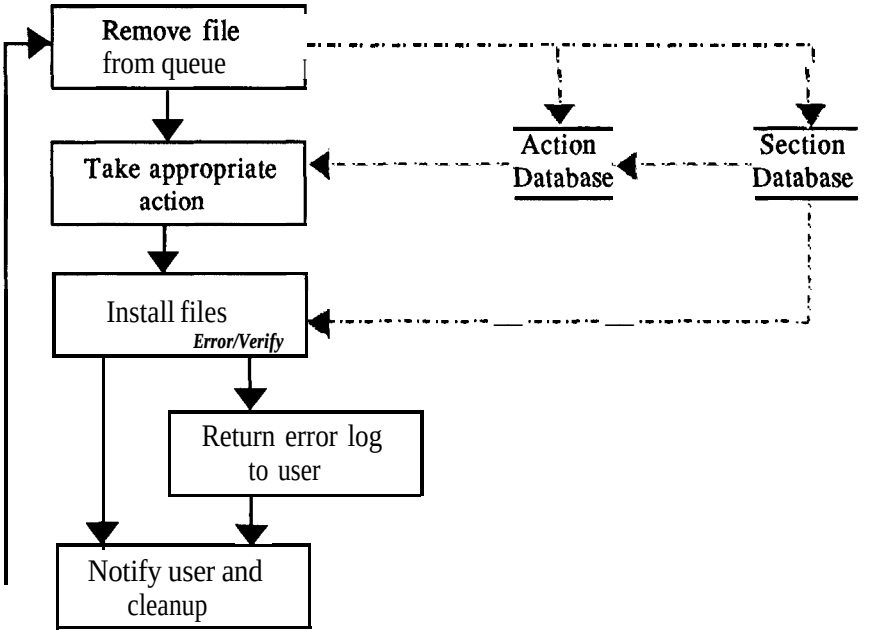
**Table 1 (Cont.): DUCS network protocols**

| Format | Purpose |
|---|---|
| **PRINT** | Used for documentation with carnage-control characters |
| BINARY | Used for binary data |
| **VMS/VM** | Transparent transfer of system dependant files (executable modules, libraries) |

**Table 2: Fields associated with DUCS update files**

| Field | Size | Use |
|---|---|---|
| Section | 8 bytes | Code section in which file belongs |
| Mode | 1 byte | Code version (D=Development, P=Production) |
| Group | 3 bytes | Allows files to be grouped together |
| Update | 4 bytes | Ensures file ordering is maintained |
| User | 8 bytes | Userid of update's originator |
| Node | 8 bytes | Node where update originated |
| Notify | 1 byte | NOTIFY option |
| Verify | 1 byte | VERIFY option |
| Dist | 1 byte | Destination system (VM/VMS/Both) |
| Comment | 80 bytes | Explanation of update |

**Figure 2: Schematic diagram of the file installer**



Whenever files are sent across the network a DUCS specific network protocol is used. Table 1 summarizes the types of files supported while Table 2 shows the auxiliary information sent with each file. The use of a store-and-forward network simplifies the task of the file distributor since it can send files without having to wait until the network connection to the target node is complete, but it can also result in tiles arriving in a different order to that in which they were sent. To solve this problem DUCS files sent over the network are each assigned a sequential update number. The receiving process then places the files in a queue (Table 4) and only processes files when the next file in the update sequence is at the front of the queue. The queue mechanism has a number of other advantages, for example the queue provides protection against "lost" files (which result in the files becoming "held' until the missing file is resent), and files can be left in the queue and only processed at a particular time of day In addition files can be grouped into blocks and only processed when all of the files in the block are present in the queue. This provides a useful mechanism for installing sets of files which must all be installed simultaneously.

An essential part of the DUCS system is the file installer which runs at each site. Figure 2 shows a schematic diagram of the file installer. Files are removed one at a time from the input queue and the auxiliary information associated with each file is used, together with information in the file installer's databases, to decide what action is to be taken with each file. The action depends on the filetype of the file and in which section it resides. Typical actions include compiling source code and installing the resulting object file into the appropriate library, installing help files into libraries, and installing news and conference items into bulletin boards[2] . Additional files generated during the installation (object or listing files for example) can also be installed under the control of the action database. If any error occurs during the installation of a file then a complete error log is generated and returned to the user who originally installed the file.

Two special filetypes are recognised to allow for special actions, such as building executable modules or manipulating code within sections (Table 3).

The main user interface to the system is via the TODUCS command which allows users to send individual files, or blocks of files, into the system. Table 5 summarizes some of the capabilities of the TODUCS command. In addition, a set of interactive commands can be sent to the file distributor or file installer to allow users to query the status of the system, and to allow the DUCS manager to correct problems (Table 6).

## Operating experience

DUCS has been in use by SLD for over 3 years and has handled over 20,000 updates. At present the DUCS system is running at 15 VMS sites and 3 VM sites (Table 7). All SLD offline software and some online software is installed and maintained through DUCS, as are all SLD announcements, meeting minutes, discussion conference, documentation, help files and support EXEC's.

**Table 3: Commands available inside a DUCSCTRL file**

| Command | Use |
|---|---|
| DELETE fn ft | Delete specified file |
| DELLIB fn | Remove specified entry from library |
| ACTION fn ft | Re-perform appropriate action on file |

**Table 3 (Cont.): Commands available inside a DUCSCTRL file**

| -Command | Use |
|---|---|
| **MOVEPROD fn ft** | Move file from DEV to PROD |
| **TIDY** | Delete DEV files which are duplicated in PROD |
| **REBUILD** | Rebuild library by recompiling **all code** |

**Table 4: DUCS input queue**

Next DUCS update: 1222
DUCS priority: 3
Next update time: 11 -DEC-1986 OO:OO:OO.OO

| Status | Entry | FIlename | Filetype | Mode | Section | User | Node | Received |
|---|---|---|---|---|---|---|---|---|
| Waiting | 1222 | JZBADD | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 10-DEC 11:14 |
| Waiting | 1223 | JZBDEL | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 10-DEC 11:15 |
| Waiting | 1224 | JZBDMI | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 10-DEC 11:15 |
| Held | **1226** | **JZBEXP** | **PREPMORT** | **DEV** | **JAZELLE** | **TONY** | **SLACSLD** | 10-DEC 11 :15 |
| Held | 1227 | JZBFND | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 10-DEC 11 :15 |
| Held | 1228 | JZBLOC | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 10-DEC 11:15 |
| Held | 1229 | JZBNXT | PREPMORT | DEV | JAZELLE | TONY | SLACSLD | 1 0-DEC 11 :15 |

**Key to Status**

| Status | Meaning |
|---|---|
| Empty | Queue is empty |
| Active | Entry is being processed |
| Pending | Entry is ready to be processed |
| Waiting | Entry would be pending but for scheduler |
| **Held** | Earlier update is missing |
| **-->n** | First file in block which finishes at entry n |

**Table 5: The TODUCS command**

**Format: TODUCS filespec[1][,filespec[1] . . . ]**

| Qualifiers | | Use |
|---|---|---|
| /LOG [4] | /NOLOG [4] | Generates message saying what was sent |
| ./SEND [4] | /NOSEND [4] | Generates message and inhibits send |
| /NOTIFY [4] | /NONOTIFY [4] | Inform uses when installation complete |
| /VERIFY | /NOVERIFY | Always return LOG to user |
| /NEXT | | Show next update number |

[1] Filespec may contain wildcards.
[4] These qualifiers always apply to all files, others can also be applied to individual files.

## Table 5 (Cont.): The TODUCS command

| Format: TODUCS **filespec**[1] **[,filespec**[1] . . . **]** | | |
|---|---|---|
| Qualifiers | | Use |
| /UPDATE-n [3] | | Set next update number |
| **/MODE=mmm** | | Control file mode (PROD or DEV) |
| **/SECTION=xxx** [2] | | Select section for file |
| **/DIST=sss** [2] | | Control destination (VM, VMS or BOTH) |
| /CONTROL | /NOCONTROL | Filespec interpreted as list of files |
| /DELETE | /NODELETE | Delete file after sending |
| { /c c | **/NOCC** } | Select **network** protocol |
| { /BINARY | **/NOBINARY** } | |
| { **/VMS** | **/VM** } | |

[1] Filespec may contain wildcards.
[2] **If** this qualifier is not specified TODUCS will scan file for line containing QUALIFIER: VALUE and use this value.
[3] **This** is a privileged command.

## Table 6: DUCS interactive commands

| File distributor commands | |
|---|---|
| Command | Use |
| HELP | Show summary of commands available |
| STATUS | Show summary of distributor status |
| QUEUE [site] | Show input queue for site |
| RESEND nnnn mmmm [site] | Resend specified updates to site |
| SET UPDATE nnnn [site] | Set update number for files to site |
| SET NEXT nnnn [site] | Set update number for files from site |
| DELETE nnnn [site] | Delete entry from site's input queue |
| HISTORY | Send update history |
| DISABLE | Disable distribution of files |
| ENABLE | Enable distribution of files |
| FREEZE section \| mode | Freeze updates to specified section or mode |
| THAW section \| mode | Undo FREEZE |
| AUTH section user [site [level]] | Authorize user |
| **AUTH/LIST** section user [site [level]] | List authorized users |
| AUTH/DELETE section user [site] | Remove user's authorization |

**Table 6 (Cont.): DUCS interactive commands**

| Command | Use |
|---|---|
| File installer commands | |
| HELP | Show summary of commands available |
| STATUS | Show summary of installer status |
| SET NEXT nnnn | Set next update number |
| SET [NO]TIME time | Set/cancel time at which updates are applied |
| SET PRIORITY n | Set installer process priority |
| DELETE nnnn | Delete entry from input queue |

**Table 7: Current list of DUCS sites**

| VMS - BITNET nodes | | | VM - BITNET nodes | VMS - DECNET nodes |
|---|---|---|---|---|
| CALTECH | Padova | SLACTBF | Perugia | Ferrara |
| Colorado | Pisa | Tennessee | Pisa | |
| Frascati | Rutgers | Vanderbilt | SLACWM | |
| Illinois | Santa Barbara | Yale | | |
| MIT | SLACSLD | | | |

Users adapted quickly to using DUCS and rapidly became used to writing code which would work on both VM and VMS operating systems. The current implementation is fairly open and relies on user's motivation to fix problems. For example if a source file is submitted which compiles on VM but not on VMS then the code is installed on VM but not on VMS. An error log is returned to the user and in general people have shown a willingness to fix such problems rapidly. This has obviated the need for a sophisticated error treatment within DUCS itself.

Having identical versions of code available on both VM and VMS even at one site has proved to be extremely useful. Many users prefer to work in a mode where code is developed and debugged using VMS's superior code development facilities, and then run on VM where there is generally more processing power available.

Many people have used DUCS to enable code to be run at home sites and a smaller number of people have developed significant amount of code at home sites. Only one home site has made significant use of the ability to install files directly, with most other users preferring to send code manually to SLAC for installation.

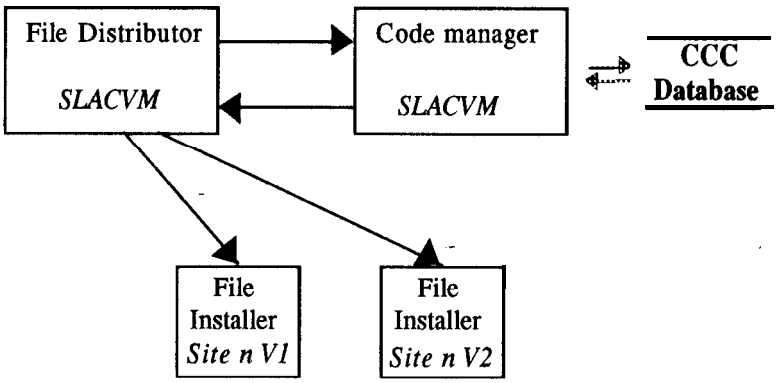
Supporting many remote VAXen has proved fairly easy. Maintenance of the entire network of 15 sites takes only a small fraction of one person's time. The only significant problems have been occasional extended network outages and finding sufficient disk space at home sites to accommodate SLD's ever expanding software. A simple and efficient mechanism has been developed for sending the complete software suite, including DUCS itself, to a home site and then installing it and using DUCS to keep it up-to-date.

By comparison supporting multiple VM sites has proved to be considerably more problematic. Every site seems to modify VM for their own needs, and often this is done so seamlessly that users (and experienced system programmers) become totally unaware of the distinction between "native" VM features and local modifications. This makes it very difficult to train people to produce site independent code. In addition, different releases of CMS and CP often require modifications to application software making support of different operating system versions very difficult.

## Future plans

In the future we plan to extend DUCS to other systems, probably including some UNIX based RISC workstations, and to other networks (we have already implemented a DECNET version of DUCS working through a BITNET-DECNET gateway process).

**Figure 3: The interface of DUCS to a complete code management system**




The other extension that we would like to make is in the area of code management. At present the code management facilities provided by DUCS are rudimentary, for instance if a user changes a common block it is his responsibility to send in a DUCSCTRL file to recompile all the code which references the common block, and there is no check that this is done correctly To this end SLAC has purchased a commercial code management system, CCC[3] , and we plan to interface this to DUCS. The idea is to use one central code management facility to provide code management capabilities at all sites. The proposed mechanism is shown in Figure 3. Under this scheme when a file is sent in to the file distributor it would first be sent to the code management process. This would use the file to update its internal representation of the code and generate a control file to be sent in addition to the original file to file installers at all sites. Multiple file installers could be run at a single site to provide shadow copies of different versions of the software on different disks.

## Conclusions

The DUCS system has been used successfully to provide a mechanism for home collaborators to use and help develop analysis, Monte-Carlo and reconstruction software for a large HEP collaboration. Such systems should prove very useful for future collaborations and we hope that the experience we have gained in producing the DUCS system can be of use to others.

## Acknowledgments

We would like to acknowledge all the members of the SLD collaboration for making the development of DUCS possible, especially those with the patience to use the system -and report problems during the early days of the system.

## References

1. K. Chadwick, R. Hollebeek, PK. Sinervo; Comput.Phys.Commun.45,1987:409

2. A set of utilities to support machine independent HELP files and conferences have been written in conjunction with DUCS. Information is available from A.S.Johnson (TONYJ@SLACVM.BITNET).

3. CCC is a product of Softool Corporation, 340 South Kellogg Avenue, Goleta, CA 93117.