# Building Large Scale Projects on NT with SoftRelTools 2 Porting, Web Tools, and IDE Conversion Tools

*G. Watts*[1].

[1] University of Washington

**Abstract**

The DØ experiment is using the utility Software Release Tools (SRT2) to manage releases of the Run 2 offline, online, and Level 3 Trigger Filter software. DØ, CDF, and BaBar use various versions of this system. SRT2 was designed on and for UNIX. The DØ experiment has adopted Windows NT as one of its official online platforms. With this commitment it was necessary to port the SRT2 environment to NT, and configure it to use NT tools, like the Microsoft C++ compiler. The solution includes using the UNIX on NT freeware solution, cygwin32, as well as an extensive set of custom tools to translate UNIX commands and switches to their NT equivalents. A number of web based support tools have also been written. These tools include the ability to compile SRT2 packages on several different platforms at once, build large releases of 100's of SRT2 software packages, and soon to build the Level 3 Physics Filter software. A second layer of software has been put in place at DØ called CTEST that abstracts some of the complexity of SRT2. Software has been developed to convert these CTEST build files into Microsoft VC++ 6.0 Integrated Development Environment (IDE) workspaces, allowing a developer the full use of the GUI development environment. The SRT2 conversion, the web build tools, and the GUI translation will all be described, along with comments on the experience of working with NT in a HEP environment.

Keywords: Trigger System, Filtering, Level 3, Windows NT Build systems, SoftRelTools

## 1 Introduction

The DØ Experiment, located at the Fermi National Accelerator Laboratory (FNAL), is a large proton-antiproton collider detector. FNAL's accelerator, the Tevatron, will collide beams with a center-of-mass energy of 2 TeV starting in March of 2001 with a collision rate of 2.5 MHz. Two years hence, an upgrade will occur to run at 7.5 MHz. The DØ detector has over one million readout channels; each readout event is approximately 250 KB. A three level trigger system has been designed to reduce the collision rate to the readout rate of 50-70 Hz.

The third level trigger, Level 3, is a farm of commodity SMP CPU boxes. Its input rate is 1000. The high data rate means the farm is fed by custom hardware and software[1]. The farm processors are all running Windows NT (hereafter abbreviated as NT).

The physics decision trigger software must run both on various UNIX platforms and also the NT platform. The software is composed of many modules, called *packages,* stored in the cvs source control system. More than 100 of these packages make up a single trigger executable. All code is written in the C++ programming language. To manage to building of these packages a multilayer, scripted, build system is used: *SoftRelTools 2* and *ctbuild*[2]. Both of these build systems have been ported to Windows NT. There is also an interface to convert the package's build instructions to the Windows NT graphical development environment (IDE).

SRT2 and ctbuild work in terms of *releases*. All packages are *rtag*ged in cvs with a version number. A release consists of a collection of these tagged packages that are extracted and built. Nominally, the versions of the packages all work together. All development of new packages, or modifications of old packages, occurs relative to a release. SRT2 allows one to replace a single package in a build for development purposes. In DØ, the offline releases consist of more than 200 packages now, and the binary code is often more than a gigabyte in size.

Because the trigger software must run on at least 3 platforms we have developed a web based build system. This build system will build packages on all platforms and save results for later viewing.

This paper is divided into sections discussing all of the above topics.

## 2 Using the Windows NT Environment

The build system, SRT2 and ctbuild, are both composed of Unix *sh* scripts. As such, they cannot run on NT in its native mode. However, it was determined at the start of this project that we cannot maintain two different build systems for a project of this complexity. We use cygwin32 subsystem to emulate UNIX on Windows NT[3].

All software at DØ is distributed using the FNAL Computing Division's (CD) *ups/upd* tools[4]. This is a system of *sh* scripts and C code to do product version management and distribution. Setting up a user's machine requires first the installation of the ups/upd packages and the cygwin32 subsystem. The CD has created a single MSDOS script that can be downloaded and run on the users machine. This script will extract the latest versions of the cygwin32 subsystem and the ups/upd tools from a central node via ftp. Once those have been installed, they are used to install several other required tools (*python* and *cvs)*. The decision was made to not use the more normal Windows Installer because the author of the script had little NT programming experience and also wished to keep the script as close to its Unix counterpart as possible. This script was difficult to write, as a script, because a number of NT registry settings, NT's large parameter/configuration database, have to be altered, and a MSDOS script does not easily access it. Also causing difficulty is every machine is configured a bit differently. The MSDOS script was less able to work around these than an Installer would have been.

Once the base environment has been configured, the user must download a complete release of the DØ software. The procedure on NT and Unix is identical. However, it isn't simple and so we have written small scripts, for each release, which will download and install them with a single click from a web browser. This has proved to be quite helpful.

### 2.1. Porting of SRT2 and ctbuild

This was, and continues to be one of the most challenging parts of the project. SRT2 uses makefile fragments to build files, while ctbuild is much more restrictive and uses small configuration files – no build commands are ever specified. The second system is preferred for new packages as it puts the building of files under the complete control of the release builders.

The FNAL CD is responsible for SRT2. The SRT2 project was started after it was clear that the original SRT had grown too big for its underlying design. The SRT2 system is very easy to extend, and, in fact, ctbuild is just an extension of SRT2. While provided by the CD, SRT2 is also modified by DØ itself. These modifications deal with areas not covered by SRT2, like DØ dependency file creation and the ctbuild extensions. While the CD has committed itself to making sure SRT2 runs on NT, the modifications that are made at DØ are not tested on NT as they are written. It cannot be underestimated the amount of time this has required.

### 2.2. Porting The C++ Code

Porting the build system has been much more difficult that porting the C++ packages. There are only two places we have experienced real difficulty, listed below.

There are several compiler issues we have had real trouble with. First, the C++ library provided by Microsoft (written by Dinkumware[5]) is out of date due to a lawsuit against Dinkumware. The library source is circa 1996. Of course, the KAI compiler (by Kuck & Associates), used on Unix, has the latest up-to-date library sources. Another problem we have encountered is that MS C++ can't handle complex templates as well as KAI. This includes nested templates, complex template expressions, and some forms of template member functions.

The second source of problems has to do with the mixed environment of NT and cygwin32. The clash happens mostly for filenames: the cygwin32 format is quite different from the NT format. Not only this, but cygwin32 supports mount points and symbolic links, something that NT 4 does not support, though Windows 2000 does.

### 2.3. Lessons Learned

If we have to rewrite the installer we will use an Installer Writer program. These toolkits are designed to work around exactly the kind of problems we have encountered.

The psychology of a large experiment is as involved in the process of software development as is its coding skill. If all parties aren't pushing in the same direction on a project of this size and complexity, it is very easy to sabotage the project.

The converter has worked very well for us. It is relatively trouble free because it uses, as input, a fairly narrowly defined set of files – the ctbuild specification files.

### 2.4. Future Plans

We are currently testing on Windows 2000. We have observed problems with the new version of the MSDOS scripting implementation – subtle changes in its behavior, especially how it handles the quotation character in string manipulation functions. These problems are very similar to the difficulties we've observed earlier caused by slightly different implementations of *sh*. We have also noted that a number of people have placed code in their SRT2 and ctbuild make files specific for NT. Unfortunately, the method they have used to test was not OS Version independent. We are also planning an upgrade of our cygwin32 subsystem in the near future.

Finally, we are always looking for better compilers. Microsoft is in the process of rewriting their compiler, however it will be a year or two before it first hits the market. We have tested Kuck & Associates solution (KAI) on NT, but found it inadequate. Intel also makes a C++ compiler that has fared better, but is expensive.

## 3 IDE Translation

One of the big advantages of working on NT is the rich visual interface it provides. The integrated development environment, Microsoft Visual Studio, is a prime example of this, including editor, compiler, and debugger in one seamless package (the *IDE*).

At first glance it does not seem possible to convert the SRT2 and ctbuild build instructions for use by the IDE. Indeed, converting an arbitrary makefile fragment is very difficult without simulating make. In fact, running make in a mode that prints out the commands it would execute does not work in the DØ environment because there are often cases in the makefile where decisions are based on files created by previous makefile commands (running test programs from makefiles is an example). Our initial attempt at conversion did just this: we replaced the commands executed by the makefile with commands that recorded information. When the makefile completed, we scanned the collected information and built a set of IDE files. Unfortunately, this would fail quite badly when test cases were run, and was quite slow.

Converting packages that are written using ctbuild, however, is much simpler. We have written a general C++ program that processes ctbuild files and outputs a set of IDE files. It has been designed so that it can later be adapted to output other sorts of IDE files (not just Microsoft's) and also work off inputs other than the ctbuild files.

### 3.1. Lessons Learned

A build system can easily be ported from one OS to another if the flexibility of the code author is limited. While this is not always possible, allowing makefile fragments in author's packages should be the exception, not the rule. In more than 90% of the packages we build at DØ there is no need for makefile fragments. The other 10% do require it, but they are often the low level I/O subsystem packages. If a large group of packages requires a makefile fragment to build correctly, the one should consider rewriting the restricted specifications to add the required support.

There should also be a well-defined and complete test suite that runs on all platforms that tests *all* aspects of the build environment. We do not have this yet at DØ.

## 4 Web Tools

The web offers a possible solution to the multi-platform porting tasks that are faced by DØ. A simple web page can be used to submit several packages for a build on all platforms. The log files can then be collected and reported back to the user. This is, in effect, a specialized batch system.

We have written such a system using Microsoft Web tools. These include FrontPage, C++, COM, the Access Database (any SQL capable database would have done), VBScript, ASP, and some JavaScript. Almost all the scripting occurs on the server side; we attempted to keep the web pages as simple as possible so that they could be viewed on all platforms. All the core server-side logic was contained in C++ objects.

The system worked as follows. The user submits a build request and a page containing ASP code receives it. This page calls the C++ core objects via COM with the requested build. All the information required to perform a build is entered into the database. The back end, which consists of a continuously running program, poles the database for new tasks. After collecting the information for the new task, it creates a *sh* script to execute the build and then executes those commands on one platform. There is one such back-end program for each platform: NT, OSF, IRIX, and Linux. As the log information is returned, the data is fed into a log file. When the user who submitted the request desires, they may inspect the contents of the log file in their web browser.

## 4.1. Lessons Learned

The biggest surprise is that very little, if any, of the server-side web code should be written in C++. All of it should have been written in scripting languages (JavaScript, Java, Visual Basic, or VBscript). Further, as with any project, every effort should be made to break the functional blocks up as much as possible. For example, the code that generates the commands is currently a generic C++ class with a subclass for each type of command required.

As has been noted previously, the build system is in continuous flux. The commands required to build a release are always changing. The current system has the commands hardwired into to the C++ code, requiring a new release of the build code each time the commands change. This was done to ease the inclusion of parameters in the generated commands. The next version will have to include a fairly sophisticated text replacement module so that most commands are generated from a source text file with parameter replacement.

These web tools have proved useful for checking one or two packages to see if they build on all platforms. Unfortunately, it is tedious to enter a large list of packages. This is especially true if this must be done several times in a row (as is often the case when developing)!

## 4.2. Future Plans

We are redesigning the system. We are going to start with a generic batch system that is suited to run almost any type of job (indeed, it is being written to satisfy more than just DØ's needs). There will be better multi-machine support in this system. Most of the core server-side code will be written in VBScript or JavaScript, and in Java when we require a real programming language. Many of the parts of the system will be broken into small COM objects that are then easily, independently, upgradeable. We also hope to have profiles that allow users to save commonly used build configurations.

## 5 Conclusions

Adding a new platform to the list of platforms supported by a large experiment is a large task and requires many people: the code authors and the build system managers. Having come as far as DØ has, it is clear that the NT OS is up to the task at hand. It is also clear that certain areas it is weak (C++ support, file system features), and other areas it is strong (development environment, tools). The web tools have proved valuable, and there is clearly more opportunity for their use.

## References

1 See paper 379, this conference, "The D0 Data Acquisition System and its Operational Control"

2 See paper 202, this conference, "Software Release Tools at Fermilab - A rewrite of the SRT package"

3 The open source cygwin32 website is located at http://sourceware.cygnus.com/cygwin/.

4 See paper 147, CHEP98, "Software Package Management and Distribution for Run 2."

5 Dinkumware, run by PJ Plauger, has an extensive website at http://www.dinkumware.com.