Application Note

Dialogic Research, Inc.
Dual OS Applications

Creating Telephony Applications for Both Windows® and Linux: Principles and Practice

Executive Summary

To help architects and programmers who only have experience in a Windows® environment move their telephony applications to Linux, this application note provides information that aims to make the transition easier. At the same time, it takes into account that the original code for Windows may not be abandoned. The ultimate goal is to demonstrate how to create flexible OS-agnostic telephony applications that are easy to build, deploy, and maintain in either environment.

Table of Contents

Introduction

With the increasing use of Linux in the enterprise, many companies need to address a growing demand for Linux-based telephony applications. A desirable solution is to port existing applications containing the required functionality from the Windows® operating system to Linux to take advantage of the lower costs and new capabilities available there.

To make existing Windows applications portable so that they will work well in both environments, note the following key design rules:

- Minimize the use of OS-specific features
- Maximize the use of libraries and packages that will work on both operating systems
- Separate user interface and processing code clearly
- Avoid graphical user interfaces (GUIs) specific to either operating system
- Isolate OS-specific code in a single area, rather than spreading it throughout the entire application
- Adopt development practices that avoid conflicts between processes in the operating systems

If the existing application is modular and well designed and organized, applying the principles listed above should not be difficult.

Although this application note is written specifically for developers using Dialogic® telephony boards and software and the Dialogic® R4 API, it concentrates on a general discussion of the basic differences in developing for two operating systems. A number of sections provide general porting guidelines and consider important issues before specific Dialogic R4 API concerns are discussed. In addition, this application note does not focus on developing specifically for Dialogic® boards that use Dialogic® System Release software or Dialogic® Host Media Processing (HMP) software, but instead discusses general issues that apply to both.

Moving to a Dual Operating System Environment

The first step in porting from Windows to Linux is to establish a usable framework for building applications. Under Microsoft Visual C++ or .NET, a workspace or solution contains individual projects. Each of these projects is a separate executable or library. The equivalent under Linux/GNU C++ is a set of project-specific makefiles, each of which defines the source components that are compiled and linked into an executable or library for the application. One or more higher-level makefiles are then used to tie the lower-level makefiles together.

Generally, the simplest approach for moving from an existing Visual C++/.NET to a Linux/GNU C++ environment is to create appropriate makefile templates and then add the information needed to produce specific makefiles for a project. The template includes placeholders for source code (C or C++), header files, directory specifications, and library references for an application.

Adding project-specific information can be automated by writing a script that extracts the component parts from a Visual C++/.NET project file and adds them to the makefile template. See Appendix A for a sample script that does such an extraction.

Unfortunately, this approach results in two separate descriptions of the application. Each will require updating as changes are made, and they are very likely to get "out of sync." A generally more reliable approach involves using a neutral utility to specify and handle the edit/build/debug/deploy process. Some possible tools include CMAKE, Boost Jam, Eclipse, and Visual SlickEdit.

CMAKE

CMAKE is an open source tool that generates OS-independent projects and makefiles. Configuration directives are put into text files in each directory of a project. When CMAKE traverses the directory tree for the project, it adds the necessary options to the OS-specific build environment that it is creating. Thus, it is possible to build both Linux makefiles and a Microsoft® .NET project from the same set of code and directives.

For more information, visit http://www.cmake.org/HTML/Index.html.

Boost Jam

Boost Jam is a build system for creating multi-platform Boost Open Source C++ Libraries, which are described below.

Boost Jam supports a wide variety of Unix and Microsoft platforms, including Visual C++ for Windows and the GNU C Compiler (GCC) for Linux. A jamfile, much like a makefile, consists of a sequence of rules that define the build process and a set of target files on which the rules operate. Conditional statements in the jamfile handle the differences between operating systems. Once the jamfile is defined, it can be used for builds on either platform by invoking the jam executable with an argument specifying the platform in use.

For more information, visit http://www.boost.org/tools/build/v2/index.html.

Eclipse

Eclipse is a comprehensive open source IDE platform written in Java with a host of optional features that may be added as plug-ins. Since it is Java-based, Eclipse will run on platforms that support a Java virtual machine with a native windowing system, including Windows and Linux Motif or GTK.

While Eclipse runs on Windows, it mainly uses the GNU compiler and linker to produce executables that must be run using Cygwin, a Linux API emulation layer that runs on top of Windows. However, native Windows executables can be produced with Eclipse in one of two ways:

- **Mingw** — uses GNU toolsets but still produces native Windows code. For more information, visit http://www.mingw.org/.
- **Microsoft Visual C++ 2005 Express Edition**— performs native Windows compilation and linking and is freely available from Microsoft at http://msdn2.microsoft.com/en-us/visualc/aa336490.aspx

The Eclipse Tools Project provides a C/C++ plug-in to the basic platform. Its Linux implementation includes an integrated editor, a front end for the standard GNU Debugger (GDB), a search engine, class viewer, and makefile generator. User-supplied makefiles are an option for more complicated makefile needs than can be handled by the automatic generator.

For more information, visit http://www.eclipse.org/tools/.

Visual SlickEdit

Visual SlickEdit is a commercially available multi-platform Editor/IDE that can provide a common look and feel for development functions such as workspace/project definition, editing, navigating between objects, and re-factoring.

SlickEdit also has a C-like macro language that can be used to automate text editing and navigation tasks on either platform. For more information, visit http://www.slickedit.com/.

Using Open Source Portable Libraries

Many open source projects are available that are portable and non-OS-specific. These projects provide functionality suitable for many different kinds of applications and can save development time.

Boost Libraries

Boost Libraries are intended to be extensions of the C++ Standard Library.

A large number of libraries are available. A partial list includes:

- Strings and text processing
- Containers and data structures
- Memory management
- Concurrent programming/threading

For more information about the libraries and the move to make the Boost Libraries a standard, see the Boost website at http://www.boost.org/.

Log4cplus

A robust event logging system enables an application to be successfully debugged and deployed. Since the functionality of an event logger is fairly straightforward, it seems best to use an existing logging package, particularly if that event logger is operating system independent. Such a package is the open source library called log4cplus, an offshoot of the Java log4j logging environment. Options include several logging levels (from DEBUG through ERROR) and configurable log file size and rollover time.

For more information about log4cplus, visit http://sourceforge.net/projects/log4cplus/.

Xerces XML

Another element often needed when developing applications is an eXtensible markup language (XML) interpreter. Applications need such an interpreter for configuration files or for parsing information in text-oriented communications protocols. For example, OpenVXI, a VoiceXML interpreter, uses Xerces as its core technology. SIP makes use of XML for organizing and transferring data.

Functional Area	Windows Approach	Linux Approach
Timers	Multimedia timer subsystem	Alarm signals and callbacks; interval timers for microsecond resolution
Timestamping	GetLocalTime, related time structs, and formatted printable text list of components	Ftime, ctime, related time structs, and formatted printable text list of components
Thread IDs	Windows threading and GetCurrentThreadId	Posix threading and pthread_self
Socket and IP-related functionality	Winsock subsystem — needs startup with its own data structs	Unix sockets — no startup, different data structs than Windows
Serial inter-process communications	Named and unnamed pipe subsystem	FIFOs — device-specific files in the Linux file system
Shared memory/ memory mapped files	Windows-specific shared memory and memory mapping subsystem	Linux-specific shared memory and memory mapping subsystem
Interrupt handling	Signals and callbacks	Signals and slightly different callback definitions

Table 1. Functional Areas and Their Implementations

The Xerces Open Source XML parsers are one component of a suite of XML products in the Apache XML project. Domain Object Model (DOM) or Simple API for XML (SAX) parser libraries are available for use with an application. Sample applications are provided to illustrate library use.

For more information, visit http://xerces.apache.org/.

Common System-Level Operations in Windows and Linux

Most applications, including telephony applications, need to perform certain tasks that use OS-level routines. Since these routines are often dissimilar even though they do the same thing, their behaviour must be normalized into a set of functions and data that an application can handle in an OS-independent way. In simple cases, only minor differences will need to be addressed; for example, a small difference in a data type or struct. In many cases, portable open source libraries, such as the Boost Libraries, can be used to normalize these functions. However, developers will have to handle these routines themselves in some cases.

Table 1 below lists a number of functional areas and summarizes how the functions are handled differently in Windows and Linux. For working sample code that implements an OS-neutral version of the functionality, download the file at http://www.dialogic.com/goto/?10563. The download includes Microsoft® Visual Studio project files and Linux makefiles.

Application Code Organization

A number of differences in OS-level calls commonly used by applications were highlighted in the last section. These differences are usually handled by C preprocessor directives embedded in the source code. Depending on the compiler setting, the preprocessor ignores or includes sections of code that pertain to Windows or Linux.

Although this beneficially reduces the size of the executables by removing unneeded code before compilation, it also results in code that is difficult to read because it is riddled with `#ifdef/#else/#endif` directives. To avoid this, all OS-specific code and all OS-related preprocessor directives can be placed in appropriate C/C++ modules in a single directory. There, neutral methods/functions can be written around pieces of code that perform the same function. See the following “sleep” function for a simple illustration.

```
void appSleep (unsigned long millisecs){
#ifdef LINUX
                usleep (millisecs * 1000);
#else
                Sleep (millisecs);
#endif
}
```

All calls in the main body of code can now use `appSleep()`. Repetitive instances of five unwieldy lines of code are collapsed into one. Changes, fixes, or adding support for a third operating system need only be done in one place. All the functional areas listed in Table 1 have been streamlined with this technique in the sample code available with this application note. The downloadable file is accessible at http://www.dialogic.com/goto/?10563.

User Interface Strategies

In most cases, telephony applications function well as system services because they are not highly interactive applications and use keyboard, screen, and mouse input sparingly. The main uses for conventional telephony user interfaces are configuration, administration, and management. A convenient, usable interface is still needed, but it is normally intended to be used by an administrator rather than an end user. Because of this, a simplified, portable user interface, as discussed in this section, is appropriate.

Command Line

A simple portable interface runs a service in console or text mode, relying on C or C++ standard I/O functions. These functions are portable across C/C++ implementations on both Linux and Windows. Because a configuration file or database is likely to be part of the system, and the application is run as a service, little or no input must be submitted from the command line. Output in the form of informational, debug, or error messages can be directed via a logging subsystem to a log file, or can be displayed in the console window. This approach is simple and often adequate.

Web Interface

A telephony service can be accessed using the graphical components of a web service. A graphical web interface has two major advantages: portability across operating systems and local or remote accessibility from any system that supports a web browser. A web interface is also the type of interface that most end users now expect. Configuration information, stored in a local database, can be displayed and modified easily. Logging output can be displayed in a scrollable window with filtering and search capabilities.

The open source Apache web server is available for both Windows and Linux at http://httpd.apache.org/. Common Gateway Interface (CGI) with PERL or JavaScript programming can be used to link the Apache web server interface to the telephony application, and used with either operating system.

Java Abstract Window Toolkit

Java has become popular due to its simplicity, ease of use and object-oriented model. Its Abstract Window Toolkit (AWT) package provides a comprehensive set of objects that can be used to implement most common GUI components. Controls make it easy to display and manipulate the telephony application service's configuration and to display log output.

Interfacing the GUI to the C/C++ telephony application can be done in a standalone Java application in two ways:

- **Direct Function Calls** — C functions can be called directly from the Java Native Interface (JNI) package. The C interface functions, along with their typed parameters, are defined as part of the Java AWT application. Java classes that contain the interface functions are invoked from callbacks triggered by GUI events. Events on the C side of the application can also trigger the invocation of Java classes, which can, for example, display an error message or set an alarm.
- **Messaging Protocol** — The GUI and telephony application interface can be abstracted into a user-defined XML-based protocol. An inter- or intra-system socket connection can then pass messages. Socket communications classes are currently available in both C++ and Java.

AWT can also be used in an applet that is embedded within an HTML document and delivered via a web page request. When the applet is on the target system, it is run in the web browser and communicates back to the host system to control the telephony service.

For more information, visit http://java.sun.com/j2se/1.4.2/docs/guide/awt/.

Tcl and Tk

Tcl and Tk can be combined to implement an operating-system-independent GUI. Tcl is an interpreted command language with a relatively simple syntax. Tk uses the X Window System to implement a ready-made set of controls with a Motif-like look and feel.

A bidirectional interface is available between Tcl and C/C++ and can be used to tie the telephony application to the Tcl/Tk GUI. The Tcl scripting is actually embedded in the C/C++ application, allowing it to start a Tcl interpreter and communicate with it when the GUI is invoked. Events and commands move between the GUI and the application to carry out various operations.

This approach is best suited to a telephony application in which the GUI will always reside on the same system. A number of Dialogic® diagnostic utilities for Dialogic® telephony boards with DM3 architecture use Tcl/Tk for their GUIs.

Common Source File Problems

Text file formats and the use of different editors can present problems. While not insurmountable, the problems are annoying to programmers used to working under a single operating system.

Text File Incompatibilities

Even though they use the same ASCII encoding, Linux and Windows text files differ because Linux line endings consist only of a newline character while Windows files end in carriage return/newline.

While most Linux editors handle both line endings, the GNU C/C++ compiler will generate warnings, and shell scripts under Linux may fail to run when a carriage return is encountered. Generally, a Windows-based source code control system will be the reason for this problem. Check the settings of the source code control system before checking out files. A setting to append a newline character to the end of each line instead of the newline/carriage return should be available. Make sure this feature is activated when the target operating system is Linux.

Checking out a body of code on one operating system, transferring it to the other, working on it, and then checking it back in on the second operating system can also lead to unpredictable results.

Winzip is another potential source of carriage return issues. Under `Options->Configuration->Miscellaneous`, verify that the `Tar file smart CR/LF conversion` box is **not** checked. This will prevent automatic insertion of carriage returns into text files that are unzipped and then moved to a Linux system. You should also have a simple script that strips carriage returns from files that were intended for use on Windows, but end up on a Linux system. See Appendix B for a sample script.

Tab Size Compatibility

The number of spaces in a tab can vary in text editors on Windows and Linux. While this does not affect parsing and source code compilation, it does produce misaligned, hard-to-read code.

When a TAB key is hit, an editor may insert a predefined number of spaces, an ASCII TAB character, or both. If code created in one editor is opened in another editor, the spaces and TAB characters may be interpreted and displayed differently, again affecting alignment and readability.

Before beginning work on code brought over to Linux, decide on a tab policy, and set all the text editors you are likely to use on either operating system to handle tabs in the same way. The best solution may be to convert tabs into the same number of spaces on insertion. This tactic, coupled with displaying code in a standard fixed width font such as Courier, should maintain alignment between editors.

Source Code Beautifiers

Source code beautifiers are utilities that will process a collection of source files and format them in the same way. The content is not changed, and the result when the files are compiled is the same. Formatting (indentation, bracket placement, spacing, etc.) is normalized, improving the code's readability. Using a source code beautifier to process files that are edited under both Linux and Windows regularly helps keep unwanted changes to a minimum once all the developers at an installation agree on a common format.

The Linux `indent` utility is an example of a source code beautifier. Since dozens of options to control formatting are possible, some experimentation may be needed to choose the proper command-line switches to achieve the desired results. Again, tab settings need special attention, and tabs should be converted to a consistent number of spaces. You can set up a script to traverse a directory hierarchy regularly and to format all source code files found in the tree.

Differences in Dialogic® Products for Windows and Linux

Generally speaking, few differences in functionality exist between Windows and Linux versions of Dialogic System Release software and Dialogic HMP software. This is particularly true for the Dialogic R4 API itself. Most of the differences are found only when installing and configuring these products.

The following section introduces developers who are familiar with Dialogic products in a Windows environment to the differences they will encounter in the Linux versions of the same products.

Programming Models

Creating a portable application for Dialogic products restricts programming model selection options. Two models cannot be used: asynchronous with Windows callback and asynchronous with Win32 synchronization. In addition, a synchronous model or a mixture of synchronous and asynchronous models are to be avoided.

An application will work best on both Linux and Windows — be easiest to write and debug, have fewer load and timing-related problems, and scale to a maximum possible density — if it is written or rewritten to conform to the following:

- Fully asynchronous programming model, polled or callback
- Single process with multiple threads
- Single thread handling multiple (~25) channels
- Single thread handling an SRL and application event processing queue

If these guidelines are followed, compatibility issues should be minimized.

Installation

Generally a "Wizard" is used to lead an operator through a step-by-step software install/uninstall on Windows. The equivalent procedure on Linux can be accomplished through the use of the Red Hat Package Manager (RPM) controlled by interactive scripts. The procedure does not have a graphical interface and may be less elegant, but the same functionality is available.

An `install.sh` script checks operating system dependencies and queries for desired installation options before launching a package installation of the system components. The package installation moves all needed files into place, sets permissions, and updates any applicable system configuration files.

The `dlguninstall.sh` script reverses the process. System services are stopped, packages removed, and installation files are cleaned up.

Depending on the Dialogic System Release software for Linux that is in use, additional steps may be necessary at installation. Here are some examples:

- Dialogic® System Release 6.1 requires the installation of the Linux Streams (LiS) drivers (supplied with the system release) and kernel source packages when PCI boards with Springware architecture are being used.
- Dialogic HMP Software Release 1.2 for Linux requires a Linux kernel upgrade before the software is installed.
- Simple Network Management Protocol (SNMP) under Linux requires the installation and build of the modified external SNMP package called Net-SNMP.

Any additional procedures are documented in the installation guides for Dialogic products running on Linux. Some of the procedures are automated while others require manual intervention.

For Dialogic documentation, visit http://www.dialogic.com/manuals/default.htm.

Configuration and System Service Startup

An option to configure the system is offered as part of the Linux installation process, and a script called `config.sh` leads the operator through the configuration of the various system components. This script is the Linux equivalent of the Dialogic Configuration Manager (DCM) interface used on Windows. Like the DCM, `config.sh` contains options for systematically configuring a specific board or technology. Unfortunately `config.sh`, like `install.sh`, is text-based and lacks a convenient GUI. Note that `config.sh` is run not only during installation but also whenever system configuration must be changed.

Another DCM component that exists as a separate script on Linux is the startup and shutdown of Dialogic® system services. Two simple commands (`dlstart` and `dlstop`) are used. On installation, system services are configured to start automatically. On the X Window System for Linux, they may be set to manual by unchecking the ct_intel box found under

```
RedHat -> System Settings-> Server
Settings -> Services
```

and saving the configuration. The same may done on the command line with

```
chkconfig --del ct _ intel
```

Dialogic HMP software releases on Windows have a separate license manager utility. This functionality, which adds a new license, deletes an old, and checks the system MAC address, is part of the `config.sh` script on Linux and may be reached by rerunning the script.

On Windows, DCM uses the Native Configuration Manager (NCM) API, which provides an interface to system configuration functions. Since other methods are used on Linux, the NCM API is not available for use as a general purpose OA&M programming interface. In SR 6.1 for Linux, an object-oriented OA&M API may be used to perform many of these same functions for PCI boards.

Logging and Diagnostics

This section lists the most useful logging and diagnostic tools for debugging telephony applications that use Dialogic® DM3 boards. Availability on Linux and the differences between how the tools are used in Windows and Linux are discussed.

System Services Logging

System services event and error messages for Dialogic products are logged via the Windows Event Viewer. A similar log viewing utility on the X Window System for Linux may be reached through

```
RedHat -> System Tools-> System Logs
```

and selecting the `System Log` option. The underlying system log file itself is `/var/log/messages` and may also be read directly with any Linux editor such as vi, emacs, or kedit.

Debugangel

Debugangel is a low-level firmware tracing tool for boards with DM3 architecture that writes to a console or a log. At present, there are minor differences in the options available for Windows and Linux.

Devmapdump

Devmapdump writes a hierarchical map of device components and their settings for boards with DM3 architecture to a console or a log. There are no differences currently between its use on Windows and Linux.

DM3Insight

DM3Insight is a tool for logging message and stream traffic from device drivers on boards with DM3 architecture. It is not available on Linux at present.

Gc_basic_call_model

Although gc_basic_call_model is actually a demo program, it can also be used as a diagnostic tool for Global Call. It is usually run in loopback mode to verify that inbound and outbound calls function correctly. Currently, the configuration and use of gc_basic_call_model is identical on Windows and Linux.

Isdntrace

Isdntrace is an ISDN message trace utility for boards with DM3 architecture that writes to a console or a log. There are no differences at present between its use on Windows and Linux.

Lineadmin

Lineadmin is used to set and monitor the alarm states on T-1 or E-1 lines on boards with DM3 architecture. Its Tcl/Tk GUI works on both operating systems, and currently there are no differences between its use on Windows and Linux.

Listboards

Listboards provides a high-level look at boards with DM3 architecture available in a system. It writes to a console or a log, and there are no current differences between its use on Windows and Linux.

PSTNDiag

PSTNDiag is a tool for troubleshooting PSTN connectivity problems on boards with DM3 architecture. Its Tcl/Tk GUI works on both operating systems, and at present there are no differences between use on Windows or Linux.

RTF Logging

The Runtime Trace Facility (RTF) provides a comprehensive library-level logger for most Dialogic® telephony components. Output is directed to the specified log file, and is configured using RtfConfigLinux.xml in Linux and RtfConfigWin.xml on Windows.

RtfConifig is a Tcl/Tk GUI that provides a way to change RTF logging options without directly editing the XML configuration file.

Sysinfo

Sysinfo is a tool that collects system configuration information, which Dialogic technical support can provide on request. It is not available on Linux at present.

Usage Differences for the Dialogic® R4 API on Windows and Linux

The following sections discuss differences in the use of the Dialogic R4 API on Windows and Linux.

Manipulating Voice Data Files for Play and Record

Under Linux, native OS-level file I/O calls are used to open and close voice data files, and then to locate, fetch, and store binary data from them. Dialogic Voice Library routines do the equivalent under Windows. Errno, the standard system error indicator, is part of the Linux operating system and is available as an equivalent routine on Windows after an error indication for the voice file manipulation routines is received. Table 2 shows the corresponding API calls on the two operating systems.

Windows	Linux
dx_fileopen()	open()
dx_fileclose()	close()
dx_fileseek()	seek()
dx_fileread()	read()
dx_filewrite()	write()
dx_fileerrno()	errno()

Table 2: Voice File Access API Calls

Arguments to both sets of routines are similar. Consult the *Dialogic® Voice API for Windows Library Reference* and the Linux man pages (online help) for details.

Two API calls that help manage the file descriptors used by the system file access functions are available only under Linux. sr_getfdcnt() and sr_getfdinfo() return the number of file descriptors in use by the R4 libraries, and an array of the descriptors themselves.

System Runtime Library Differences

When using the asynchronous callback mode, the definition of the callback function used to obtain System Runtime Library (SRL) events differs slightly on Linux and Windows.

The Windows event callback is invoked with an event handle passed in as a parameter so that it can be used when calling sr_getevttype(), sr_getevtdatap(), sr_getevtlen(), and sr_getevtdev() to fetch information on a specific event. Linux calls to the sr_getevtxxx() functions do not need the handle, so it is not passed to the event callback. In addition, consult OS-specific documentation for the sr_getevtxxx() and sr_getUserContext() functions to understand the differences in scoping and data validity between Windows and Linux.

Four other functions are currently available on Windows only:

- **sr_NotifyEvent( )** — Tells the SRL to send an event notification to a window.
- **sr_getboardcount()** — Retrieves the number of boards of a particular type in use.
- **sr_GetDllVersion()** — Retrieves the SRL DLL version number.
- **sr_libinit()** — Initializes the SRL DLL.

Certain sr_getparm() and sr_setparm() parameters are not valid across operating systems.

- SR_MODEID — Fetches the value for the event notification mode (Linux only).
- SR_MODELTYPE — Fetches the value for the model type (Windows only).
- SR_WIN32INFO — Describes a Win32 synchronization data (Windows only).
- ATDV_IOPORT() — Returns the base port address used by a device (Linux only).

sr_waitevt() and sr_waitevEx() only have a granularity of one second under Linux. Obtaining error information also differs between Windows and Linux environments.

Use of gc_Start and gc_Stop in Global Call Applications

Under Linux, gc_Start and gc_Stop must be called from the parent process if the application forks into child processes.

Appendix A: Sample Scripts to Aid in Cross-Operating-System Builds

This appendix contains two sample scripts: extract_ms-project and remove_cr.

extract _ ms _ project

```
#!/bin/sh
#
#   Extract useful pieces from a Windows VC++ 6 project file for
#   use in building a Linux Makefile.
#
#   Results are processed for easier inclusion in Makefiles, and
#   sent to stdout.
#
#################################################################

if test $# != 1; then
   echo usage: $0 project _ file _ name
   exit 1
fi

if ! test -f $1; then
   echo File $1 not found
   exit 1
fi

PROJ _ FILE=$1

# Extract list of header files first
echo
echo Header files in project:
echo
grep 'SOURCE=' $PROJ _ FILE | tr  '\\' / | grep '\.h$' | sed s/SOURCE=//
echo
echo

# Then list of .cpp files
echo C++ files in project:
echo
grep 'SOURCE=' $PROJ _ FILE | tr  '\\' / | grep '\.cpp$' | sed s/SOURCE=//
echo
echo

# Extract a list of -I (include) options for the compilation
echo Include directories referenced by the compiler:
echo
grep 'ADD CPP' $PROJ _ FILE | \
awk '{ for (i = 0; i < NF; ++i) {if ($i == "/I") { print $(i+1)}}}' | \
tr '\\' / | \
sed -e s/^\"/-I/ -e s/\"$// | \
sort | uniq
```

```
echo
echo

# Extract a list of -D (compiler directive) options for the compilation
echo Compiler directives used:
echo
grep 'ADD CPP' $PROJ_FILE | \
awk '{ for (i = 0; i < NF; ++i) {if ($i == "/D") { print $(i+1)}}}' | \
tr '\\' / | \
sed -e s/^\"/-D/ -e s/\"$// | \
sort | uniq
echo
echo

# Extract a list of libraries loaded
echo Libraries loaded:
echo
grep 'ADD LINK32' $PROJ_FILE | \
awk '{ for (i = 0; i < NF; ++i) {print $i}}' | \
grep 'lib$' | \
sort | uniq

echo
echo
```

remove_cr

```
#!/bin/sh
#
#   Script to remove any instances of CRs (carriage returns)
#   in a text file.
#
###############################################################

if test $# != 1; then
       echo usage: $0 text_file_name
       exit 1
fi

if ! test -f $1; then
       echo File $1 not found
       exit 1
fi

TXT_FILE=$1

cat $TXT_FILE | tr -d '\r' > tmpfile.$$
mv $TXT_FILE $TXT_FILE.bak
mv tmpfile.$$ $TXT_FILE
```

Acronyms

API	Application Programming Interface
AWT	Abstract Window Toolkit
CGI	Common Gateway Interface
DCM	Dialogic® Configuration Manager
DOM	Domain Object Model
FIFO	First In First Out
GCC	GNU C Compiler
GDB	GNU Debugger
GNU	Gnu's Not Unix
GTK	GIMP Tool Kit
GUI	Graphical User Interface
IDE	Integrated Development Environment
ISDN	Integrated Services Digital Network
JNI	Java Native Interface
LiS	Linux Streams
NCM	Native Configuration Manager
OS	Operating System
PSTN	Public Switched Telephone Network
RPM	Red Hat Package Manager
RTF	Runtime Trace Facility
SAX	Simple API for XML
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
SRL	System Runtime Library
XML	eXtensible Markup Language

For More Information

The following references are cited in this application note:

Apache HTTP Server	http://httpd.apache.org/
Boost	http://www.boost.org/
Boost Jam	http://www.boost.org/tools/build/v2/index.html
CMAKE	http://www.cmake.org/HTML/Index.html
Eclipse Tools Project	http://www.eclipse.org/tools
Java AWT	http://java.sun.com/j2se/1.4.2/docs/guide/awt/
Log4Cplus	http://sourceforge.net/projects/log4cplus/
Mingw	http://www.mingw.org/
Visual C++	http://msdn2.microsoft.com/en-us/visualc/aa336490.aspx
Visual SlickEdit	http://www.slickedit.com/
Xerces XML	http://xerces.apache.org/
For Dialogic documentation, visit	http://www.dialogic.com/manuals/default.htm.

To learn more about Dialogic® products, go to **www.dialogic.com**.

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9