Gateway SIDK Developer’s Guide

Firmware Version 9.4
Document Version 0.6

BRINGING TECHNOLOGY HOME
www.pace.com

Notice to Users

© 2010 Pace, Inc. All rights reserved. This manual in whole or in part, may not be reproduced, translated, or reduced to any machine-readable form without prior written approval.

Pace PROVIDES NO WARRANTY WITH REGARD TO THIS MANUAL, THE SOFTWARE, OR OTHER INFORMATION CONTAINED HEREIN, AND HEREBY EXPRESSLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE WITH REGARD TO THIS MANUAL, THE SOFTWARE, OR SUCH OTHER INFORMATION, IN NO EVENT SHALL Pace, INC. BE LIABLE FOR ANY INCIDENTAL, CONSEQUENTIAL, OR SPECIAL DAMAGES, WHETHER BASED ON TORT, CONTRACT, OR OTHERWISE, ARISING OUT OF OR IN CONNECTION WITH THIS MANUAL, THE SOFTWARE, OR OTHER INFORMATION CONTAINED HEREIN OR THE USE THEREOF.

Pace, Inc. reserves the right to make any modification to this manual or the information contained herein at any time without notice. The software described herein is governed by the terms of a separate user license agreement.

Updates and additions to software may require an additional charge. Subscriptions to online service providers may require a fee and credit card information. Financial services may require prior arrangements with participating financial institutions.

Pace, the Pace logo, and HomePortal are registered trademarks of Pace, Inc. All other company names may be trade names or trademarks of their respective owners.

11222010

5100-001020-000

Contents

About This Guide v
Audience v
Prerequisites v
Document Structure v
Style Conventions vi

CHAPTER 1 The Pace Gateway SIDK 1

CHAPTER 2 Understanding the SIDK Source Tree 2
The Target Platform 2
The Host Platform 3
Reachover 3

CHAPTER 3 Configuring the SIDK 4
The Project Directory 4
SIDK Variables 4
Environment Variables 4
Make Variables 5
Special Variables 6

CHAPTER 4 Building the SIDK Project using Eclipse 8
Building the Entire SIDK Project 8
Building the Target Image for the Xen Project 9
Building a Specific Make Target 9
Troubleshooting Compilation Errors 10

CHAPTER 5 Integrating External Applications 11
Configuring the SIDK for an External Directory 11
Integrating an External Application 11
Example: Building Samba 14

CHAPTER 6 Debugging using GDB 18
Understanding the Debug Process Flow 18
Prerequisites for Debugging 18
Debugging on a Target Device 19
Debugging on the Host Machine 19
Starting the Debug Process 25
Working in the Debug Perspective 26
Working with Breakpoints 26
Working with Watchpoints 26
Working with Variables 27

CHAPTER 7 Conducting Memory Checks 28
Using Valgrind 28

The Valgrind Log File 29
Example: Memory Leak Message 29

CHAPTER 8 Editing and Recompiling Code 30
Editing Code. 30
Recompiling Code with Changes 30

CHAPTER 9 Managing the Workspace in Eclipse. 31
Synchronizing the Workspace with the CVS Repository 31
Updating the Resource from the Branch 31
Committing Changes to the Branch 31
Committing Changes to the CVS Repository 32
Updating the Workspace with the CVS Repository. 32
Updating the Workspace using the Team Menu. 33
Updating the Workspace using the Synchronize View 33
Merging the Changes Manually 34
Updating and Committing Changes in the Synchronizing Perspective 35
Replacing Resources in the Workspace. 36

CHAPTER 10 Optimizing Eclipse for the SIDK Project 37
Disabling the Build before Launching 37
Disabling the C/C++ Indexer 38
Increasing the Build Console Limit. 38

APPENDIX A Xen Installation and Configuration. 40
Verifying Virtualization Support 40
Installing and Configuring Xen 40

APPENDIX B Flash Partitions. 43

APPENDIX C Boot Sequence 44

About This Guide

This document provides information about using the Pace Software Integration Development Kit (SIDK) code to customize or integrate your own or other third party software code and applications with the Pace gateway firmware.

Audience

This guide is intended for system integrators who are adding, modifying, or deleting features using SIDK code for target platform hardware.

Prerequisites

The following are the essential prerequisites for using this document:

- Successful installation of SIDK as per the procedures defined in the *Gateway SIDK Installation Guide* document.
- Development of a test SIDK build without any changes in the source code.

Note Refer to *Eclipse Help for Eclipse 3.5 C/C++ Development Tools* for detailed information of the Eclipse Integrated Development Environment (IDE).

Document Structure

This document has the following major topics:

- The Pace Gateway SIDK. Describes Pace's SIDK, which you can use to customize the gateway firmware and integrate with a variety of hardware platforms. as well as unique software requirements. It also lists a minimum set of requirements to install and run the SIDK.
- Understanding the SIDK Source Tree. Explains the layout of the SIDK source code. It describes various directories and files that help in creating a build.
- Configuring the SIDK. Describes the procedures to configure the SIDK. To configure the SIDK, you have to work with the project directory and SIDK variables.
- Building the SIDK Project using Eclipse. Describes the procedures to build an SIDK project using Eclipse. It describes the procedures to build a project and specific make targets after configuring the SIDK.
- Integrating External Applications. Describes the procedures to integrate an external application with the SIDK code and create the build. You can refer this chapter to integrate various applications in the SIDK and enhance the build.
- Debugging using GDB. Describes the procedures to debug the application binary using GNU Debugger (GDB). After you create the build for the target platform, you can use GDB to debug the program binary.
- Conducting Memory Checks. Describes the procedures to use Valgrind for conducting memory checks on the target build application. Using Valgrind, you can conduct memory profiling and detect memory leaks for the target build application.
- Editing and Recompiling Code. Contains information about how to edit and recompile code in order to debug and fix memory issues.

- Managing the Workspace in Eclipse. Describes the procedures to manage the source code post build process in Eclipse. After you finalize changes in the source code, you can use the CVS repository to maintain the code.
- Optimizing Eclipse for the SIDK Project. Describes the procedures for optimizing Eclipse features for the SIDK project, such as disabling the build before launching, disabling the C/C++ indexer, and increasing the build console limit.

Style Conventions

The following style conventions are used in this guide:

Note Notes contain incidental information about the subject. In this guide, they are used to provide additional information about the product and to call attention to exceptions.

Caution notes identify information that helps prevent damage to hardware or loss of data.

Warning notes identify information that helps prevent injury or death.

Typographical Conventions

Convention	Used For
Blue Text	Cross references
Bold	Interface elements that are clicked or selected
Italic	Emphasis, book titles, variables, list terms, user inputs
`Monospace`	Command syntax and code
`Monospace Italic`	Variables within command syntax and code

CHAPTER 1

The Pace Gateway SIDK

The Pace Gateway SIDK comprises various components and tool chains, which you can use to customize Pace's firmware to meet specific device requirements.

Using the development kit, you can write new code, modify the code available in the tree, and package the results into target images for all supported platforms. This allows rapid prototyping and releases of features across hardware.

A secondary benefit of the SIDK is that updates can be incorporated from Pace without impacting the external source code, since it is isolated into its own directory structure.

The development kit consists of the following:

- Toolchain for developing against the target hardware
  - GCC 4.2.4
  - GDB 6.8
  - Binutils 2.19.1
  - Flex and Bison
  - Additional tools as required
- Kernel and Modules compiled for the target hardware
- Flash image utilities
  - SquashFS, TL, JFFS2 support
  - Gang Image Format for programming flash
- Libraries, Header files, and Man pages
  - Embedded C Runtime (libc, ld.so)
  - C++ Runtime (libstdc++)
  - Internationalization and UTF-8 support (libicuuc and libiconv)
  - Additional Libraries documented in Man pages
- Applications
  - Web Server (apache)
  - Command Line Interface (tcli)
  - CPE WAN Management through TR-069 (CWMP)
  - Additional applications as documented

CHAPTER 2

Understanding the SIDK Source Tree

The SIDK source tree consists of various directories and files that help in creating the build. It contains the following SIDK directories:

- *apps*. Application sources.
- *cross*. Cross-compilation toolchain for specific targets.
- *dist*. Distribution make rules.
- *doc*. Documentation.
- *external*. External application sources.
- *kernel*. Kernel sources.
- *lib*. Library sources.
- *ports*. Ported sources.
- *project*. Target configuration files and makefile include files for building libraries and programs.
- *skel*. Skeleton of the system.
- *tools*. Host tools sources.

It also contains the following files:

- *BUILDING*. Contains procedure for building Pace NMD from source code.
- *Makefile*. The main Makefile for Pace NMD. It is scripted for GNU make.
- *nmd.mk*. The parent nmd.mk file, which contains the target to build all source directories like apps, libs, and kernel.
- *Umbrella.mk*. Allows the build of Pace NMD to be handled recursively by jumping from the host make (GNU) to the tooldir-based make (1).
- *version.txt*. Contains the version number of the build.

During creation of build, additional directories are created on:

- The Target Platform
- The Host Platform

The Target Platform

Refer to the following table for description of directories created on the target platform during the build process:

Directory	Location	Description
Object Directory	builddir.<project>/obj_dir	Contains all compiled object code for target applications.
Staging Directory	builddir.<project>/obj_dir	Contains release targets copied from the object directory during installation to provide linkage.
Target Directory	builddir.<project>/target_dir	Contains files for target devices like: • Root filesystem • Kernel • Initramfs and flash images generated by 'release' and 'dist' make target rules

The Host Platform

Refer to the following table for description of directories created on the host platform during the build process:

Directory	Location	Description
Object Directory	objdir.<arch>	Contains all compiled object code for host applications and libraries.
Include Directory	objdir.<arch>/include	The include file installation directory for the host libs. Note: Host include files are never installed in the host's /usr/ include directory.
Library Directory	objdir.<arch>/lib	The library installation directory for host libraries. Host libraries are never installed in the host's /usr/lib directory.
Tool Directory	tooldir.<arch>	Contains the tools to build the code for the target architecture. The contents of the tools directory are built and placed in the tooldir.<arch> directory.

Note Refer to the BUILDING file located in the root directory of the source code tree for the latest information about the organization of the SIDK source code.

Reachover

During the build process, most libraries and applications are compiled for the host and target. However, in the source tree only one copy of the source files is maintained. Usually, the source for a specific application or library is located in the apps or libs directory of the target platform. While building the same apps or libs for the host, the source files are extracted or pulled in the source code from the directory. This process is known as reachover.

Generally, applications that use reachover have no source files located in the project directory. For example, the source code for BusyBox is available in the ports/busybox directory. However, the compilation for the target takes place from the apps/busybox directory that contains the make configuration for BusyBox.

CHAPTER 3

Configuring the SIDK

Configuring the SIDK involves working with the project directory and several variables that control the behavior of the build.

The Project Directory

At the start of any project, you have to define a configuration file for the project. You must create the configuration file for the SIDK project in the project directory. The Umbrella.mk file (located in the top directory) creates the mk.conf file in the project directory with default values for building projects, if the file does not exist. The PROJECTS variable in mk.conf file lists all the projects with .conf in the project directory, by default.

Note To build selective projects, edit the mk.conf file and add a new project name to the PROJECTS variable in space-separated format.

SIDK Variables

A variable contains data or a value that you can define and store in it. The following SIDK variables control the behavior of the build:

- Environment Variables
- Make Variables
- Special Variables

Environment Variables

There are many environment variables that control the behavior of the build. These variables are set in the process environment.

Refer the following table for environment variables and their description:

Environment Variable	Value Description

HOST_SH	Contains path name to a POSIX-compliant shell.
	If not set explicitly, the default is set using heuristics dependent on the host platform, or from the SHELL under which Umbrella.mk is executed.
	If the host system's /bin/sh is not POSIX-compliant, add the absolute path to a shell:
	`HOST_SH=/pathtoworkingshell`
	`export HOST_SH`
	`${HOST_SH} make [options]`

HOST_CC	Contains the path name to the C compiler used for creating the toolchain.
HOST_CXX	Contains the path name to the C++ compiler used for creating the toolchain.
HOST_MAKE	Contains the path name to invoke a GNU make.
PROJECTS	Contains a space-separated list of projects to build in the tree. This is a list of target projects that are supported by a build.

Make Variables

There are several variables that you can define in the make(1) configuration file specified by MAKECONF variable.

Refer to the following table for the make variables and their description:

Make Variable	Value Description
BUILDID	Contains the identifier for the build. The identifier will be appended to object directory names, and can be consulted in the make(1) configuration file in order to set additional build parameters, such as compiler flags.
BUILDSEED	Seeds the GCC random number generator using `-frandom-seed` flag with this value. GCC uses random numbers when compiling C++ code. By default, it is set to NMD-(majorversion). Using a fixed value causes C++ binaries to be similar when built from the same sources. Refer to GCC documentation for detailed information about `-frandom-seed`.
MAKECONF	Contains the name of the make(1) configuration file. It can be set only in the process environment. The default name of the make(1) configuration file is mk.conf located in the project directory.
MAKEVERBOSE	Contains the level of verbosity of status messages. The supported values for this variable are: • *0*. No descriptive messages are displayed. • *1*. Descriptive messages are displayed. • *2*. Descriptive messages (prefixed with a '#') and command output is not suppressed. The default value is *2*.
MKCATPAGES	Indicates whether the preformatted plaintext manual pages are created during a build. The supported value for the variable is *yes* or *no*. The default value is *yes*.
MKCRYPTO	Indicates whether cryptographic code is to be included in a build. The variable is provided for the benefit of regions that do not allow strong cryptography. It does not affect use of the standard low-security password encryption system, for example, crypt(3). The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKDOC	Indicates whether system documentation destined for the DESTDIR/usr/share/doc directory is to be installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKHTML	Indicates whether preformatted HTML manual pages are to be built and installed. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKINFO	Indicates whether GNU Info files, used for the documentation for most of the compilation tools, are to be created and installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKLINT	Indicates whether lint(1) is to be run against portions of the Pace NMD source code during the build, and whether lint libraries are to be installed into DESTDIR/usr/libdata/lint. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKMAN	Indicates whether manual pages are to be installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKNLS	Indicates whether Native Language System (NLS) locale zone files are to be compiled and installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*.

Make Variable	Value Description
MKOBJ	Indicates whether object directories are to be created when running "make obj". If set to *no*, then all built files will be located in the regular source tree. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKPIC	Indicates whether shared objects and libraries are to be created and installed during a build. If set to *no*, the entire built system will be linked statically. The supported value for this variable is *yes* or *no*. All platforms except sh3 default to *yes*.
MKPICINSTALL	Indicates whether the ar(1) format libraries (lib*_pic.a), used to generate shared libraries, are installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*.
MKPROFILE	Indicates whether profiled libraries (lib*_p.a) are to be built and installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*. However, some platforms disable MKPROFILE by default, at times, due to toolchain problems with profiled code.
MKSHARE	Indicates whether files destined to reside in the DESTDIR/usr/share directory are to be built and installed during a build. The supported value for this variable is *yes* or *no*. The default value is *yes*. If set to *no*, then MKCATPAGES, MKDOC, MKINFO, MKMAN, and MKNLS variables are set to *no* unconditionally.
MKSTRIPIDENT	Indicates whether program binaries and shared libraries are to be built to include RCS IDs for use with ident(1). The supported value for this variable is *yes* or *no*. The default value is *no*.
MKTTINTERP	Determines if the TrueType bytecode interpreter is turned on for X builds. See http://www.freetype.org/patents.html for details. The supported value for this variable is *yes* or *no*. The default value is *no*.
MKUNPRIVED	Indicates whether an unprivileged install will occur. The user, group, permissions, and file flags, will not be set on the installed items; instead the information will be appended to METALOG file in DESTDIR directory. The contents of the METALOG file are used during the generation of the distribution tar files to ensure that the appropriate file ownership is stored. The supported value for this variable is *yes* or *no*. The default value is *no*.
MKUPDATE	Indicates whether all install operations intended to write to the DESTDIR directory will compare file timestamps before installation. The installation phase is skipped if the destination files are up-to-date. The supported value for this variable is *yes* or *no*. The default value is *no*.

Special Variables

During creation of Makefiles, special variables are used. The values of the special variables are determined during each execution of each rule.

Refer to the following table for the important special variables:

Variable	Value Description
TOPPATH	Contains the path of the root directory of the source tree that contains the Umbrella.mk file.
.OBJDIR	Contains the target directory of the object files.
.CURDIR	Contains the current working directory.

Variable	Value Description
DESTDIR	Contains the install path. The default value is set to builddir.<project>/stage_dir directory.
NMDSRCDIR	Points to the root of the source tree.
.PATH	Contains the path information for the 'reachover'-style Makefile.
MAKEOBJDIR	Contains the value that overrides the object directory.
HOSTOBJDIR	Points to the host object directory.

CHAPTER 4

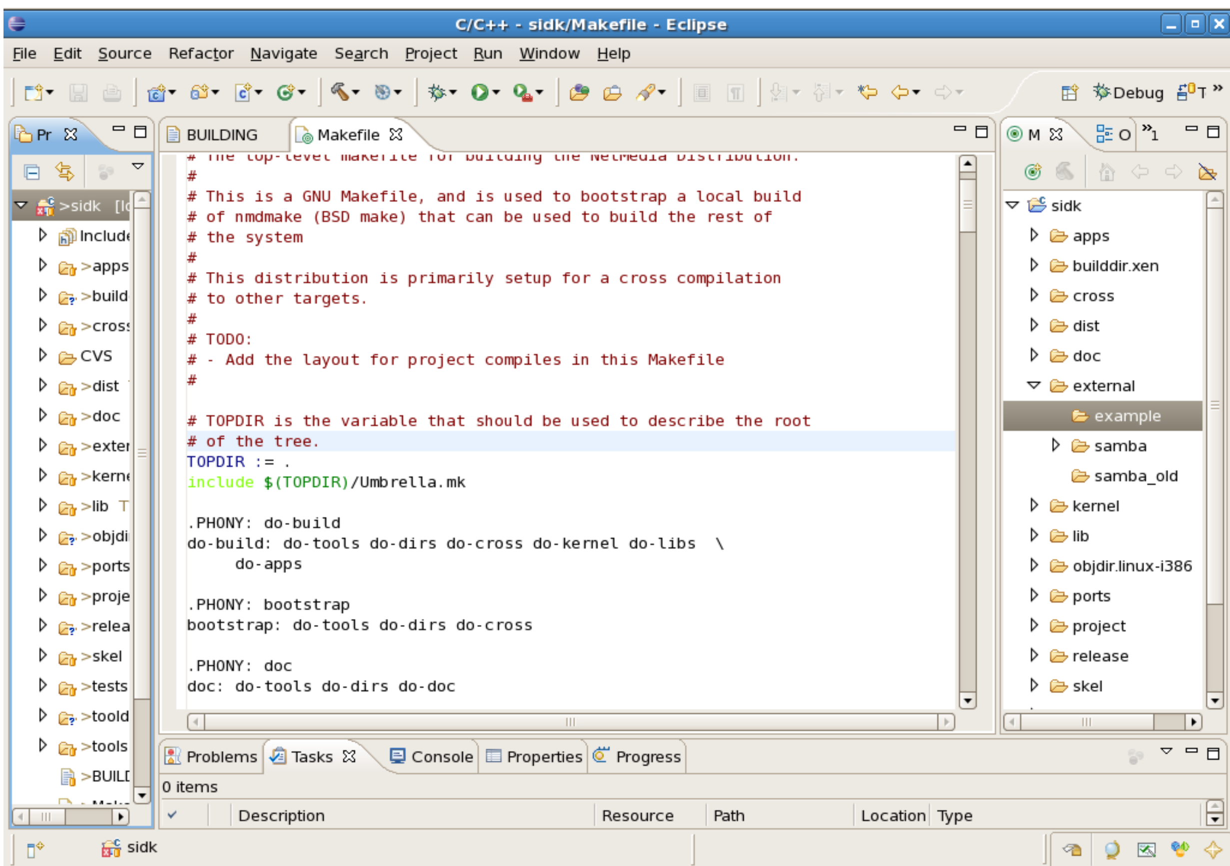
Building the SIDK Project using Eclipse

After configuring the SIDK, you can build the target image to create a firmware image. This firmware image is available in the /dist/*target-platform*/ directory. You can upload the firmware image on the target platform hardware.

You can build an entire project or a specific make target using the SIDK.

Building the Entire SIDK Project

After successful installation of the SIDK with Eclipse as per the procedures described in the *Gateway SIDK Installation Guide* document, you can start the compilation of code (build process).



To start the build process for the SIDK project in Eclipse:

1. On the menu bar, click **Project**.
2. Click **Build Project**.

   -or-

   Click the **Hammer** icon.

The parent Makefile located in the root of the SIDK tree is used to start the build process. The firmware image obtained as a result of the build process is available in the /dist/*target-platform*/ directory. It contains the make

distribution rules, which you can use to create target images. You can then load the target image on the target device and run it.

Building the Target Image for the Xen Project

The Xen project example explains the constituents of the target image.

To build the target image for Xen:

- Ensure that the xen.conf file is available in the Xen project directory.

  The xen.conf file contains the machine and Linux architecture for the target and variables required for the sub-builds.
- Create a directory named xen in the dist directory.

  The xen directory contains the distribution rules, which you can use to create target images for loading and running on the target device.

The Xen project target image has the following directories under dist/zen:

- *xen*. The directory named after the project.
- *config*. Contains distribution rules to obtain all configuration files for the target image.
- *initramfs.install*. Contains distribution rules to build the initramfs installation.
- *initramfs.system*. Contains scripts to initiate the required file system during boot.
- *iso*. Contains distribution rules to build the ISO image.
- *rootfs*. Contains distribution rules to build the filesystem image.
- *ui*. Contains UI files for the Xen build.

The dist/zen directory has the following Xen project target image files:

- *Makefile*. A wrapper to obtain all the paths and information to pass to the rules as stated in the nmd.mk file for the compilation.
- *nmd.mk*. Contains rules for the compilation of code.
- *xen_domU.conf.in*. Contains domain configuration to run xen.

Note 1 The constituents of target images may vary as per the project.

If Xen is not available on your machine, refer to Installing and Configuring Xen.

Building a Specific Make Target

The entry point for the build framework is the HOST_MAKE variable that contains required information to invoke the nmdmake that was generated during the toolchain build. The variable allows invoking nmdmake multiple times for the number of projects defined in the umbrella.mk file. Thus, a single invocation of HOST_MAKE builds multiple targets as defined by the configuration.

The HOST_MAKE make targets facilitate building the entire source tree from the top source level.

Refer to the following table for description of HOST_MAKE top level make targets:

Make Target	Description
do-tools	Builds the tools directory.
do-dirs	Constructs the directory layout for all the projects that you want to build.
do-cross	Builds the project specific cross-compilation toolchain like, compiler, debugger, and other utilities as defined in configuration file for the specific project.
do-kernel	Builds the kernel.
do-libs	Builds libraries and install header files.
do-apps	Builds the apps directory.

Make Target	Description
do-doc	Builds the doc directory.
do-external	Builds the external directory.
do-dist	Builds the target image that can be loaded on the target device.
do-build	Builds the complete code except the external, doc, and dist directories. Invokes do-tools, do-dirs, do-cross, do-kernel, do-libs, and do-apps targets in the serial order.
bootstrap	Builds tools, dirs, and cross compilers. Invokes do-tools, do-dirs, and do-cross targets in serial order.
doc	Builds the doc directory. Invokes do-tools, do-dirs, and do-doc targets in serial order.
dist	Builds the complete code including the target image. Invokes do-build, do-external, and do-dist targets.
release	Builds the doc and dist targets. Creates the release directory with target images in it.
clean	Runs clean target in all subdirectories like apps, dist, lib, and tools. Retains the staging directory and target directory (partially). It may not fix all problems.
realclean	Runs clean target first and deletes the builddir.*, objdir.*, and tooldir.* directories. Restores the tree to the initial check out state.

You can use individual targets to build specific code. However, if the target build is dependent on other targets, ensure that the build of the other targets is ready or build the other targets first.

Refer to Integrating an External Application for to review a sample do-external make target.

Troubleshooting Compilation Errors

You may encounter compilation errors while building the SIDK code. Refer to the following table for the list of common errors and their solutions:

Error	Solution
A failure detected in another branch of the parallel make	Scroll in the editor to identify the exact code where error has occurred and resolve the issue.

Undefined reference to a function	Ensure that the function is correctly declared and defined in the module where the error appears. If the function is a part of another library, then link the required library.
	Example:
	`Error: /home/user/workspace/sidk-xen/`
	`builddir.xen/stage_dir/lib/libdalrpc.so:`
	`undefined reference to 'send_receive_message'`
	
	In the above example, the send_receive_message function is defined in a different location. That is, the libdalrpc.so depends on the library that has function send_receive_message defined in it, which is libnpseal.so in this case.
	
	To resolve this issue, add the following line to the nmd.mk file, located in lib/npgateway/dalrpc/ directory:
	`LIBDPLIBS+=npseal ${.CURDIR}/../npseal`

Broken pipe error	Right-click the relevant project in **Project Explorer** and select **Refresh**. After refresh operation is complete, rebuild the code.

CHAPTER 5

Integrating External Applications

Using SIDK source code, you can integrate an external application with the SIDK. The SIDK source tree contains the external directory that relates to external applications.

To integrate external application with SIDK, you have to configure the SIDK to build an external directory and then build an external application using Eclipse.

Configuring the SIDK for an External Directory

To configure the SIDK to build an external directory:

1. Create a subdirectory in the external directory of SIDK code for the external application.
2. Create a Makefile for the new application in the subdirectory.

   Point the TOPDIR variable to the top directory of the workspace and include the Umbrella.mk file:

   ```
   TOPDIR=../..
   include ${TOPDIR}/Umbrella.mk
   ```

   For example, to integrate samba, create a directory named `samba` as defined in the path below:

   `/home/user/workspace/sidk/external/samba`

   In the above example, `sidk` is the top directory of the workspace and contains the umbrella.mk file.
3. Create an nmd.mk file for the new application in the subdirectory.

   Add an entry of the new application using the SUBDIR variable in the nmd.mk file:

   ```
   # External nmd build makefiles
   .include <nmd.own.mk>
   SUBDIR+= example yournewfeature
   .include <nmd.subdir.mk>
   ```

   To script the nmd.mk file, determine the method to build the code. At times, you may have to run a configuration script for the code to be integrated to facilitate a Makefile to build the code.

   Do one of the following to run the configuration script:

   - (Recommended) Create an intelligent nmd.mk file that runs the configuration script with the required arguments passed to it to create the appropriate Makefile for the code to be built.

     You may need a few iterations to determine the required arguments that are to be passed to the configuration script for building the code with desired features and packages.
   - Run the configuration script on the host machine and then convert the resultant Makefile into the nmd.mk file.

Integrating an External Application

To integrate an external application, you have to build the source code in the external directory. Building the source code in the external directory implies compilation and building of the feature source code as defined by the SUBDIR variable in the nmd.mk file, located in external directory.

For example, consider the following entry in the nmd.mk file:

```
# External nmd build makefiles
.include <nmd.own.mk>
SUBDIR+= feature1 feature2
.include <nmd.subdir.mk>
```

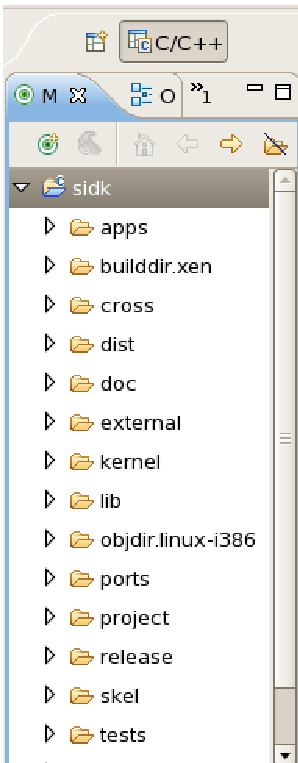
If the value of the SUBDIR variable contains feature1 and feature2, both the features are built.

If you want to build feature3 as well, ensure that feature3 is a value of the SUBDIR variable in the nmd.mk file located in the external directory.

Note You should build the entire project before integrating an external application.

To build an external application with Eclipse:

1. Click the **Make Targets** tab on the right pane.



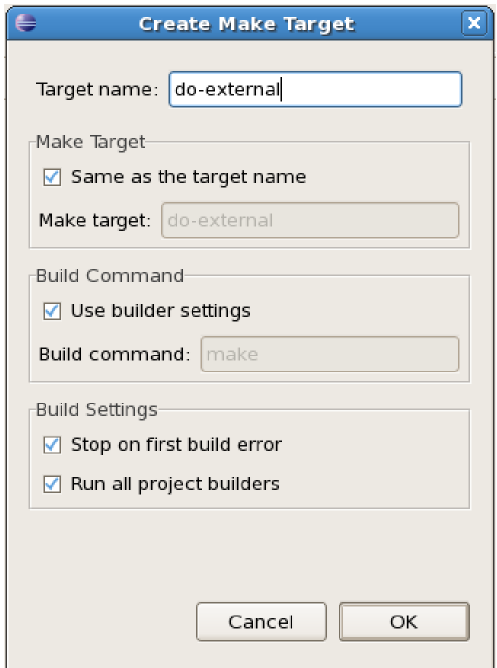
2. Select the root directory of the project **sidk**.
3. Right-click and select **New**.

   -or-

   Click the **New Make Target** icon on the Make Targets toolbar.

4. On the **Create Make Target** window, enter target name in the **Target Name** field.

   For example, you can enter the target name as `do-external` for creating a do-external make target.

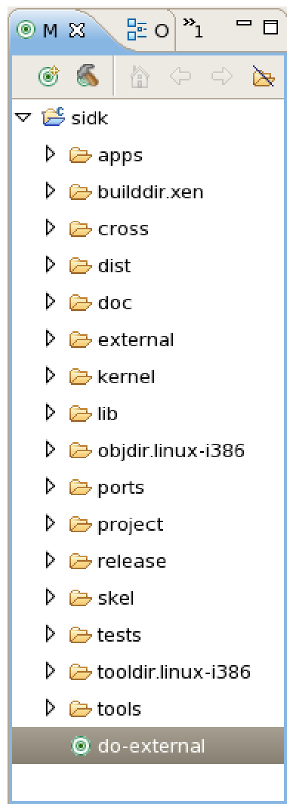


5. Leave the values of the **Make Target**, **Build Command**, and **Build Settings** panels as they are.

Note To have separate values for **Target name** and **Make Target**, clear the **Same as the target name** check box and enter the value for Make target in the **Make Target** field.

6. Click **OK**.

   The `do-external` target appears below the sidk tree on the right pane.



7. Double-click the ***do-external*** target on the right pane to start the build.

Example: Building Samba

The following procedure describes an example for building samba, an external application, with the SIDK.

To build samba with the SIDK:

1. Download the samba source code from the samba Web site: http://www.samba.org/

   The *samba-3.0.37.tar.gz* file is considered as the downloaded samba source code for reference.
2. Create a samba directory in the external subdirectory of the sidk directory.
3. Copy the samba source code in the samba directory.
4. Untar the samba source code file in the external directory.

   The path of the file is external/samba/samba-3.0.37/*source code directories and files*.
5. Create the nmd.mk file for samba in the external/samba directory.

   The content of a sample nmd.mk file for samba is displayed below:

   The comments are lines preceded by #.

```
#NetMedia Distribution Make Rules
# following included file should be put in the beginning of nmd.mk, it includes
nmd.own.mk that contains
# source tree configuration parameters and few global 'feature configuration'
parameters.
```

```
.include <nmd.init.mk>

SAMBA_VER=3.0.37
SAMBA=samba-${SAMBA_VER}
SAMBA_DIR=${.OBJDIR}/${SAMBA}/source

SRC_DIR=$(.CURDIR)

CONFIGURE_ENV+= \
    PATH="${TOOLDIR}/bin:${DESTDIR}/usr/bin:${PATH}"

# Below are arguments to configure script, their usage depends on the way source code
needs to be build.
# You may not get this list in the one go and may require iterations to make it as
required.

CONFIGURE_ARGS+= \
--build=`uname -m` \
--host=${MACHINE_GNU_PLATFORM} \
--target=${MACHINE_GNU_PLATFORM} \
--prefix=${DESTDIR:Q} \
--bindir=$(DESTDIR:Q)/bin/ \
--localstatedir=/var \
--with-lockdir=/var \
--with-piddir=/var \
--with-privatedir=/var \
--with-logfilebase=/var \
--with-configdir=/var \
--with-libiconv=${DESTDIR:Q} \
--without-ldap \
--without-ads \
--without-acl \
--with-included-popt \
--with-included-iniparser \
--disable-shared-libs \
--disable-static \
--disable-cups \
--disable-iprint

TARGET_ALL?=all
TARGET_INSTALL?=install

# The following FILES* variables imply inclusion of nmd.files.mk
FILES=smb.conf
FILESDIR=/var

.PATH: ${.CURDIR}
```

```
# following target copies the source code into object_directory to build it
.copy_done: $(SRC_DIR)
@mkdir -p ${SAMBA} 2>/dev/null || true
@rm -rf $(SAMBA_DIR)
@cd ${SAMBA}
@cp -r $(SRC_DIR)/samba-3.0.37/* ./
@cd -
@touch $@

# similar to configure arguments even the following configure environment variables
need to be used
# according to the manner in which the code should be build.

.configure_done: .copy_done
@(cd ${SAMBA_DIR}; rm -f config.cache; ./autogen.sh; \
     /usr/bin/env ${CONFIGURE_ENV} \
        samba_cv_HAVE_GETTIMEOFDAY_TZ=yes \
        samba_cv_USE_SETREUID=yes \
        samba_cv_HAVE_KERNEL_OPLOCKS_LINUX=yes \
        samba_cv_HAVE_IFACE_IFCONF=yes \
        samba_cv_HAVE_MMAP=yes \
        samba_cv_HAVE_FCNTL_LOCK=yes \
        samba_cv_HAVE_SECURE_MKSTEMP=yes \
        samba_cv_HAVE_NATIVE_ICONV=no \
        samba_cv_CC_NEGATIVE_ENUM_VALUES=no \
        samba_cv_fpie=no \
        SMB_BUILD_CC_NEGATIVE_ENUM_VALUES=yes \
        HAVE_GETGROUPLIST=no \
        ./configure ${CONFIGURE_ARGS})
@touch $@

.build_done: .configure_done
@(cd ${SAMBA_DIR}; \
 /usr/bin/env ${CONFIGURE_ENV} \
        ${HOST_MAKE} proto; \
 /usr/bin/env ${CONFIGURE_ENV} \
        ${HOST_MAKE} ${TARGET_ALL})
@if [ ! -f $@ ] || \
  [ -n "$$(find ${SAMBA_DIR} -type f -newer $@ -print)" ]; \
    then touch $@; fi

.install_done: .build_done
    ${_MKMSG_INSTALL} samba
    @(cd ${SAMBA_DIR}; \
        /usr/bin/env ${CONFIGURE_ENV} \
            ${HOST_MAKE} ${TARGET_INSTALL})
```

```
    @touch $@

realall: .build_done
realinstall: .install_done

clean: clean.sambaprogs
clean.sambaprogs:
-rm -rf .*_done src build

# following nmd.*.mk files, when required, are always included in the end of nmd.mk.
.include <nmd.obj.mk>
.include <nmd.inc.mk>
.include <nmd.files.mk>
```

Note The content of the nmd.mk file may differ with respect to applications that are to be integrated or developed with the SIDK. Refer to the nmd.README file in the project/mk directory for details about .include <nmd.*.mk>

6. Create the Makefile for samba in the external/samba directory.
7. Invoke the do-external target to build samba with the other external application.

   Refer to Integrating an External Application for procedures to invoke do-external target.

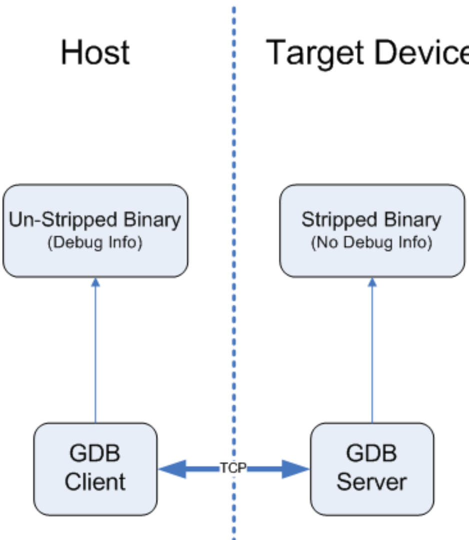
CHAPTER 6

Debugging using GDB

The GNU Debugger (GDB) is the standard debugger for the GNU software system. Using GDB, you can debug an application binary. You can perform the debugging process on a target device, as well as on a host machine through CLI and Eclipse.

Understanding the Debug Process Flow

The following figure displays the basic communication between the GDB server and client:



You have to run the un-stripped binary (with debug information) on the host machine. The binary to be run on the target device can be either stripped or unstripped, depending upon the memory of target device.

Prerequisites for Debugging

The following are prerequisites for debugging using GDB:

- Edit the nmd.mk file.

  Modify the nmd.mk file of the application to add compile flags to CFLAGS/CPPFLAGS variables for debugging:

  ```
  CFLAGS = -g# -O0
  ```

  -or-

  ```
  CPPFLAGS = -g# O0
  ```

  The `-g#` switch in the flags sets the GNU Compiler Collection (GCC) debug level.

Note	Refer the GCC man pages and set the required debug level.

The O0 switch in the flags disables optimization.

Note The CFLAGS/CPPFLAGS flags are important for appropriate functioning of the GDB, as they create required debug symbols in the application binary.

- Create a softlink for the GDB client library.

  To create a softlink for the GDB client library in the builddir-xen/stage_dir/lib directory:

  a. Navigate to the builddir-xen/stage_dir/lib directory:

  ```
  cd /home/user/workspace/sidk/builddir-xen/stage_dir/lib
  ```

  b. Create the softlink for the client library:

  ```
  ln -s /home/user/workspace/sidk/builddir.xen/stage_dir/usr/i686-linux-gnu/lib/
  libstdc++.so.6 libstdc++.so.6
  ln -s /home/user/workspace/sidk/builddir.xen/stage_dir/usr/i686-linux-gnu/lib/
  libgcc_s.so.1 libgcc_s.so.1
  ```

Debugging on a Target Device

To debug the application binary on target devices, execute the following command:

- If the application is not running:

  ```
  gdbserver TARGET_IP_ADDRESS:PORTNUM TARGET_PROG TARGET_PROG_ARGS
  ```

  -or-

- If the application is already running:

  ```
  gdbserver -attach TARGET_IP_ADDRESS:PORTNUM TARGET_PROG_PID
  ```

Refer to the following table for the description of parameters in the above commands:

Parameter	Description
TARGET_IP_ADDRESS	IP address of the target device.
PORTNUM	Port number on which the GDB server is started. This can be any unused port. Ensure that it is allowed through the firewall.
TARGET_PROG	Program binary on the target device.
TARGET_PROG_ARGS	Arguments to be passed to the program binary.
TARGET_PROG_PID	Proportional–Integral–Derivative (PID) of the running application on the target device.

Debugging on the Host Machine

To debug the application binary on host machines, you can use either the command line or Eclipse.

Debugging using the Command Line

To debug using the command line:

1. Run the cross-compiled GDB that resides in the builddir.TARGET/stage_dir/usr/bin/ directory.

   For example, if TARGET is xen and host is i686, GDB resides in the builddir.xen/stage_dir/usr/bin/i686-linux-gnu-gdb directory. To debug the dropbear application, execute the following command:

   ```
   builddir.xen/stage_dir/usr/bin/i686-linux-gnu-gdb builddir.xen/obj_dir/apps/
   dropbear/server/dropbear
   ```

   The GDB prompt appears.

2. Execute the following command to connect to the remote GDB server:

   ```
   target remote TARGET_IP_ADDRESS:PORTNUM
   ```

Refer to the following table for the description of parameters in the above command:

Parameter	Description
TARGET_IP_ADDRESS	IP address of the target device.
PORTNUM	Port number on which GDB server is running.

After execution of the above command, the target GDB server indicates that a remote debugger is attached and lists its IP address.

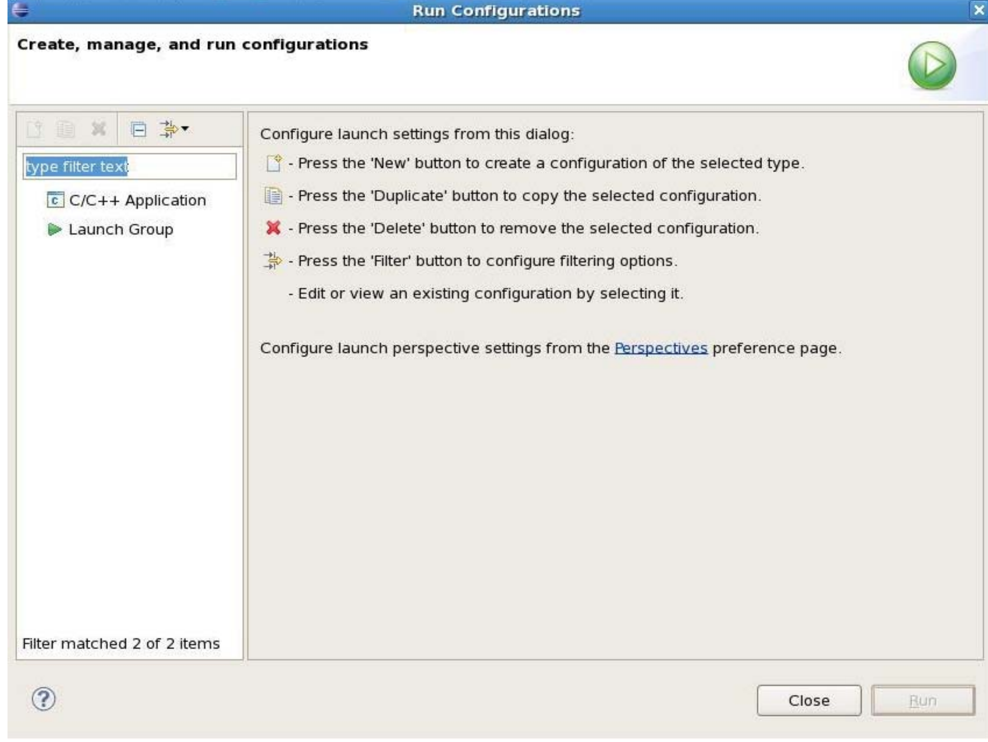
Debugging using Eclipse

To debug using Eclipse, you have to define the settings for the Run Configurations and Debug Configurations features.

To define settings for the Run Configurations feature:

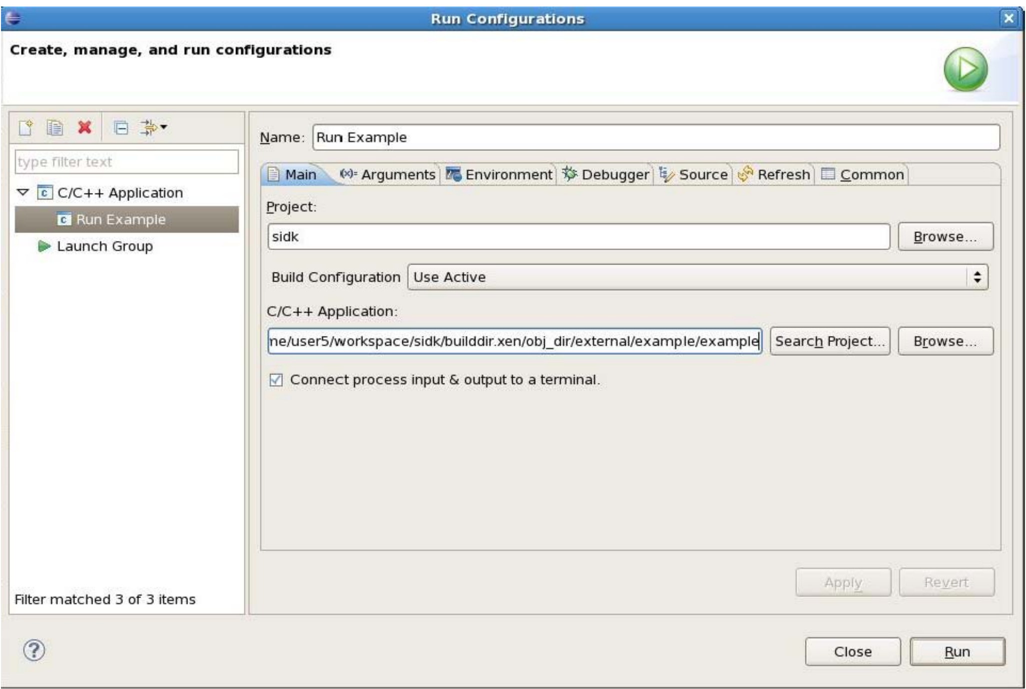
1. On the menu bar, click **Run**.
2. Click **Run Configurations**.
3. On the left pane, double-click **C/C++ Applications**.

   A default entry with the current project name is created. For example, if the name of the current project is `sidk`, the entry is created with the name **sidk**.

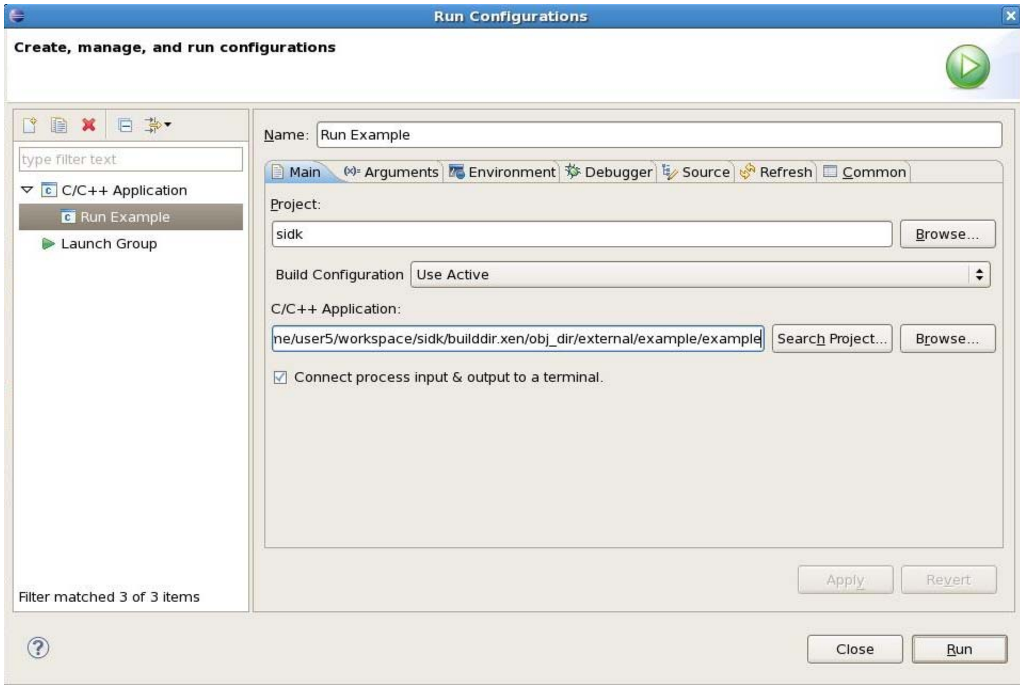


4. Edit the application name in the **Name** field.

   For example, you can enter the application name as `Run Example` in the **Name** field.

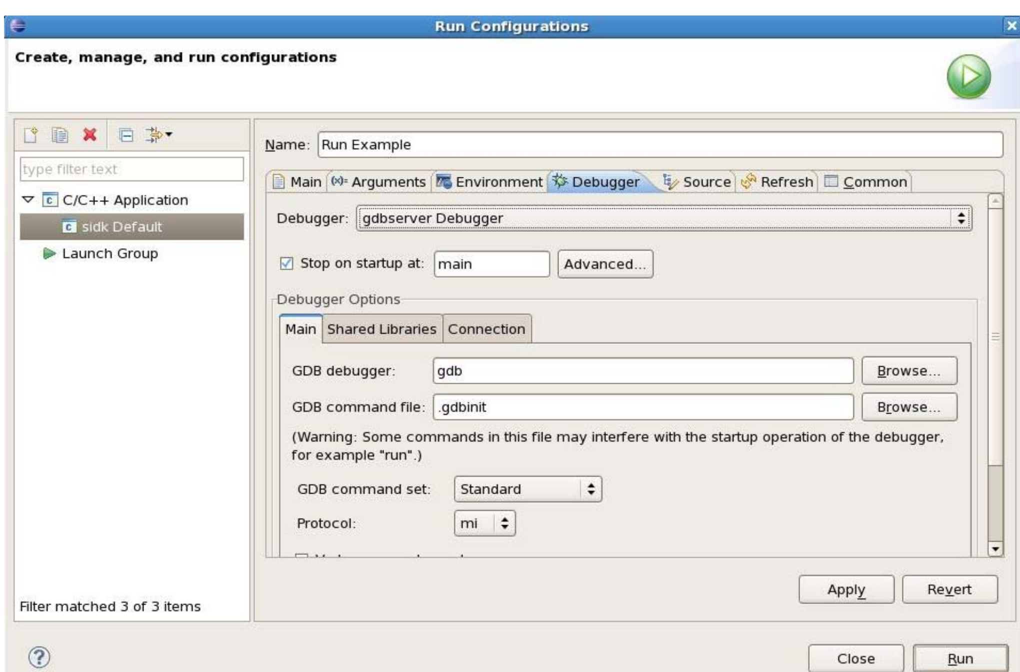


5. Click the **Build Configuration** drop-down list box and select **Use Active**.
6. Click **Browse** near **C/C++ Application** to add the application.
   The application binary resides in the builddir.*target*/obj_dir/*path of installation* directory.

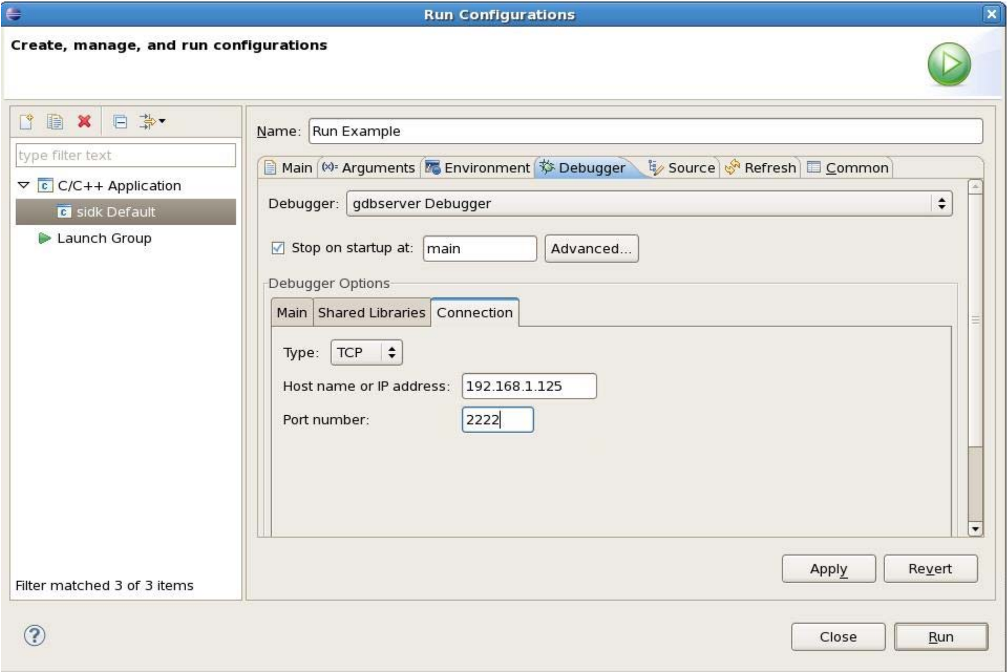


7. Click the **Debugger** tab.

8. Click the **Debugger** drop-down list box and select **gdbserver Debugger**.



9. Clear the **Stop on startup at:** check box to let the program run until it is interrupted manually or hits a breakpoint.
10. On the **Debugger Options** panel, click the **Connection** tab and configure the following settings:
    a. Click the **Type** drop-down list box and select **TCP**.
    b. Enter the server name or IP address of the GDB server (IP address of the target device) in the **Host name or IP address** field.
    c. Enter port number of the GDB server in the **Port Number** field.



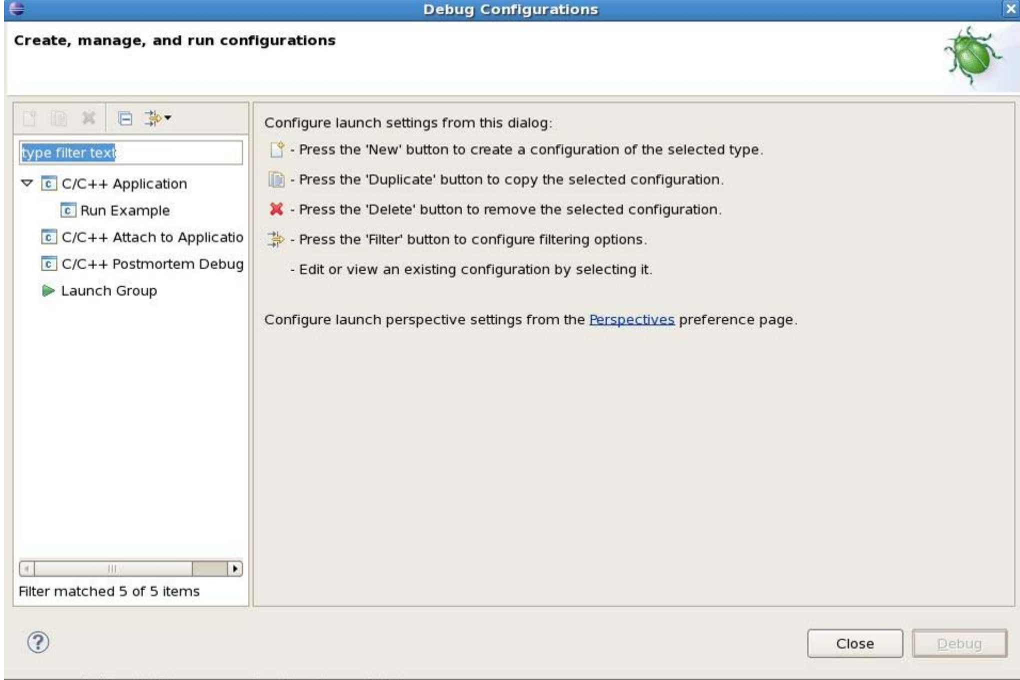
11. Click **Apply**.
12. Click **Close**.

The Run Configurations setup is complete.

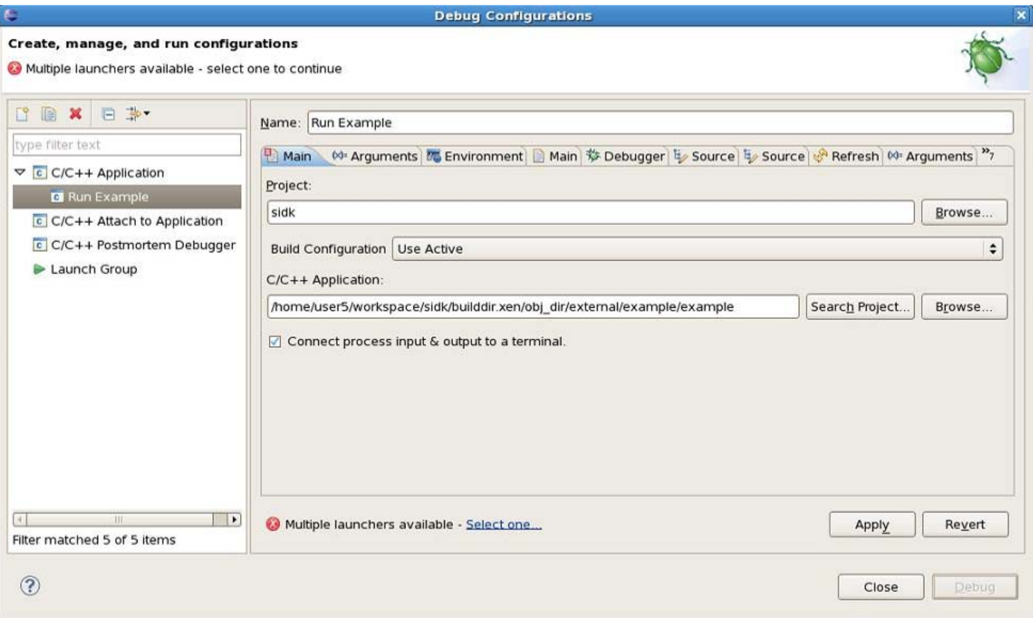
To define settings for Debug Configurations feature:

1. On the menu bar, click **Run**.
2. Click **Debug Configurations**.
3. On the left pane, expand **C/C++ Application** and double-click the application that you had created while defining settings for **Run Configuration** feature.

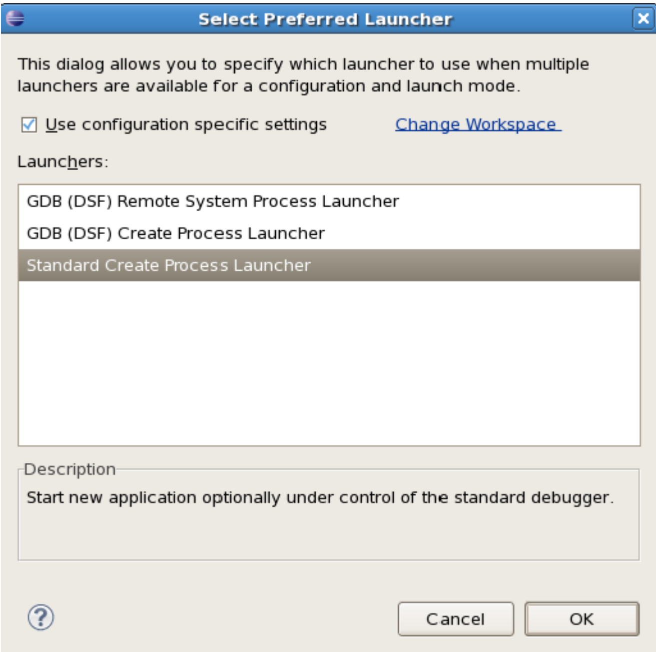
   For example, you can double-click **Run Example**.



On the **Debug Configurations** window, the **Multiple launchers available** error message appears at the top and bottom of the window.

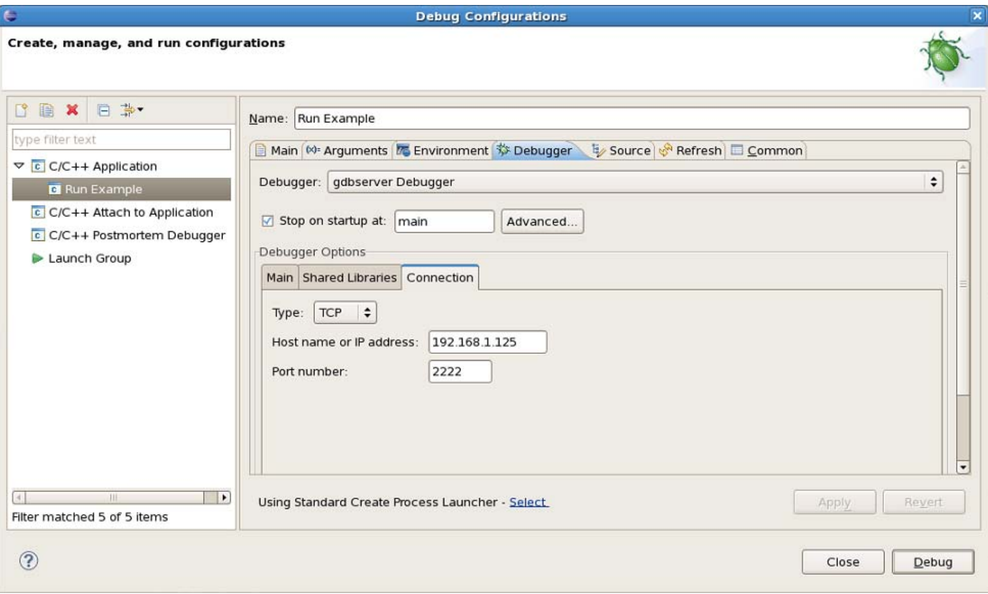


4. Click the **Select one** link at the bottom of the window to resolve the error.
5. On the **Select Preferred Launcher** window, select the **Use configuration specific setting** check box.
6. Select **Standard Create Process Launcher** from the list of Launchers.



7. Click **OK** to save and close the window.
8. On the **Debug Configurations** window, click the **Debugger** tab.
9. Click the **Debugger** drop-down list box and select **gdbserver Debugger**.
10. Clear the **Stop on startup at:** check box to let your program run until it is interrupted manually or hits a breakpoint.

11. On the **Debugger Options** panel, click the **Connection** tab and configure the following settings:
    a. Click the **Type** drop-down list box and select **TCP**.
    b. Enter the server name or IP address of the GDB server (IP address of the target device) in the **Host name or IP address** field.
    c. Enter the port number of the GDB server in the **Port Number** field.



12. Click **Apply**.
13. Click **Close**.

The Debug Configuration setup is complete.

Starting the Debug Process

After defining settings for the Run Configuration and Debug Configuration features, start the debug process.

To start the debug process:

1. Start the GDB server on the target device using command line:

```
gdbserver 192.168.1.125:2222 example <enter>
```

Note	Refer to Debugging on a Target Device for descriptions of parameters in the above command.

```
# gdbserver 192.168.1.125:2222 example
Process example created; pid = 620
Listening on port 2222
_
```

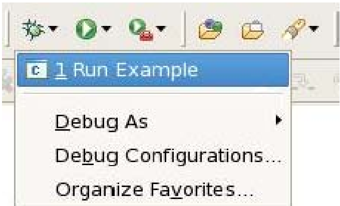
2. In Eclipse, click **Windows** on the menu bar.
3. Click **Open Perspective**.
4. Click **Debug**.

Note	You can also toggle between perspectives using tabs at the top right corner of the Eclipse window.

5. Click **Run**.
6. Click **Debug to trigger debug**.

   -or-

   Use the debug shortcut icon on the menu bar.



Working in the Debug Perspective

In the Debug Perspective, you can perform tasks such as adding or removing breakpoints, watching variable values, and viewing register values.

Working with Breakpoints

A breakpoint is set on an executable line of a program. If the breakpoint is enabled when you debug, the execution suspends before the line of code for which you have set the breakpoint.

To enable or disable breakpoints:

- Right-click the selected breakpoint and click **Enable** or **Disable**.

To add a breakpoint point:

- Double-click the marker bar located in the left margin of the **C/C++ Editor** beside the line of code where you want to add it.

```
example.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sysexits.h>

#define WDATA "Write this string into the file\n"

void write_file(FILE *, char *);
```

To remove a breakpoint:

1. Right-click the selected breakpoint and click **Remove**.

   You can click **Remove All** to delete all of the breakpoints.

Working with Watchpoints

A watchpoint is a special breakpoint that stops the execution of an application upon change of value of a given expression. Unlike breakpoints (which are line-specific), watchpoints are associated with files. They take place whenever a specified condition is true, irrespective of when or where it occurred. You can set a watchpoint on a global variable by highlighting the variable in the editor, or by selecting it in the Outline view.

To enable or disable watchpoints:

- Right-click the selected watchpoint and click **Enable** or **Disable**.

To set a watchpoint on a global variable:

1. Highlight the variable in the editor or select it in the **Outline** view.
2. Click **Run**.
3. Click **Toggle Watchpoint**.
4. Choose any of the following:
   - Select the **Read** check box to stop execution when the watch expression is read.
   - Select the **Write** check box to stop execution when the watch expression is written.

The watchpoint appears in the **Breakpoints** view list.

To remove watchpoints:

- Right-click the selected watchpoint and click **Remove**.

You can click **Remove All** to delete all of the watchpoints.

Working with Variables

During a debug session, you can view variable types, and change or disable the variable values.

To view variables types:

- Click **Show Type Names** in the **Variables** view.

To change the variable value during debugging:

1. Right-click a variable in **Variables** view and select **Change value**.
2. Enter a new value for the variable.

Note When you change the value of a variable, you can test the application's ability to handle a particular value or to speed through a loop.

To disable the value of a variable while debugging:

- Right-click the variable in **Variables** view and select **Disable**.

Note Disabling the value of a variable prevents the debugger from reading the variable's value from the target.

Refer to Eclipse Help for detailed information about topics related to Watchpoints, Outline View and Variables View.

CHAPTER 7

Conducting Memory Checks

For conducting memory checks on applications of target build, you can use Valgrind. Valgrind is an instrumentation framework for building tools for dynamic analysis. It includes a set of tools used for memory debugging, memory leak detection, and profiling for application enhancements.

Using Valgrind

You can check the application of the target build for memory profiling. You can also find memory leaks using the Memcheck tool of Valgrind.

To run the application with Valgrind and its Memcheck tool, execute the following command:

```
valgrind --tool=memcheck --log-file=file --show-reachable=yes --leak-check=full --
track-fds=yes --error-limit=no --num-callers=40 <application_to_run> &
```

Refer to the following table for description of parameters in the above command:

Parameter	Description
--tool =memcheck	Memcheck detects memory-management problems. This option can detect when your program: • Accesses undesired memory (such as areas not yet allocated, areas freed, or areas past the end of heap blocks). • Uses uninitialized values that may result in a program crash. • Leaks memory. • Frees the heap blocks incorrectly (double frees, mismatched frees). • Passes overlapping source and destination memory blocks to memcpy() and related functions.
--leak-check	Helps obtaining the total memory leak check.
--log-file	Represents the file name in which Valgrind reports all the logs.
-show-reachable=yes	Displays the reachable blocks in the leak check.
--track-fds=yes	Tracks open file descriptors upon exit.
--error-limit=no	If set to `no`, does not restrict the number of errors to be displayed.
&	Appears at the end of the command, to allow it to run in the background.

You can capture the Valgrind log for your application, which is a daemon that runs continuously.

To capture the Valgrind log:

1. Enter `fg` to bring the command in the foreground.
2. Press CTRL+C to stop Valgrind.

This creates the Valgrind log file for examination.

Note Abruptly killing the Valgrind process may create an incomplete log file.

The Valgrind Log File

The content of a sample Valgrind log file is as follows:

```
==19182== Invalid write of size 4
==19182== at 0x804838F: f (example.c:6)
==19182== by 0x80483AB: main (example.c:11)
==19182== Address 0x1BA45050 is 0 bytes after a block of size 40 alloc'd
==19182== at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182== by 0x8048385: f (example.c:5)
==19182== by 0x80483AB: main (example.c:11)
```

In the above log file:

- `19182` is the process ID.
- `Invalid write of size 4` indicates an error and implies that the program is using memory that is not allocated to it due to a heap block overrun.
- Below the first line is a stack trace that describes where the problem occurred.

  Stack traces can be voluminous. If the stack trace is too small, use the `--num-callers` option to increase its volume.

  Note It is recommended that you read the stack traces from the bottom of the log file.

- The code addresses (for example, `0x804838F`) are used for tracking down specific bugs.

Example: Memory Leak Message

The code snippet of a sample memory leak message is as follows:

```
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19182== at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130
==19182== by 0x8048385: f (a.c:5)
==19182== by 0x80483AB: main (a.c:11)
```

In the above memory leak message:

- The stack trace specifies the location of the memory. Memcheck cannot specify the reason of the memory leak.
- The stack trace also specifies the type of leak.

  The two most important leak types are:

  - *definitely lost*. There is a memory leak in the application that requires resolution.
  - *probably lost*. There is a possibility of memory leak in the application, with respect to the pointer. For example, the pointer is moved to the middle of the heap block, causing less memory to be freed than expected.

Note Refer the Valgrind User Manual for comprehensive information about Valgrind and its tools.

CHAPTER 8

Editing and Recompiling Code

You may need to edit the source code to add new features or make corrections in the source code for fixing memory errors. After making changes to the code, you have to recompile the code to create the build.

Editing Code

If you want to make changes in the code, edit the required file in the Eclipse editor. This procedure updates only the local copy in current workspace.

Note If you need the original version of the file (prior to making changes), you can restore it by checking out the file from the CVS repository.

After you finalize the changes, you need to update the file in CVS repository. Refer to Committing Changes to the CVS Repository to commit the file changes to the repository.

Recompiling Code with Changes

If you have made changes to the part of the code that requires the other Make target for the build process, then you have to create the new Make target. If the other Make target is already available, then you can use it to compile the code.

You can verify whether the changes are properly compiled by doing a local compilation. For example, to verify the compilation of code for any changes you have made in iptables, you have to create a local target in the iptables subdirectory located in apps directory to build it locally. Before you do run 'make' you should first do 'make clean' and also remove earlier version of build files from the builddir.<project>/obj_dir/iptables/ directory. This ensures that your changes are incorporated in the new binary or library.

To build the complete image after making changes in the code, refer to Building the SIDK Project using Eclipse.

CHAPTER 9

Managing the Workspace in Eclipse

There are various procedures to manage your workspace in Eclipse, which you may require for managing the source code while working with SIDK projects. You can perform the following procedures in Eclipse:

- Synchronizing the Workspace with the CVS Repository
- Committing Changes to the CVS Repository
- Updating the Workspace with the CVS Repository

Synchronizing the Workspace with the CVS Repository

In the CVS team programming environment, there are two distinct processes involved in synchronizing resources:

- Updating with the latest changes from a branch.
- Committing changes to the branch.

When you make changes in the workspace, the resources are saved locally. You can commit the changes to the branch to enable other users to view them. You must update your workspace to view changes committed by other users to the branch.

Using filters in the Synchronize view, you can determine whether you want to view only:

- *Incoming changes from the branch*. If accepted, they update the workspace resource to the latest version currently committed in the branch.
- *Outgoing changes from the workspace*. If committed, they change the branch resources to match those currently present in the workspace.

Irrespective of the filter selected, the Synchronize view always displays conflicts that arise when a resource with a more recent version in the branch gets locally modified. In this case, you can take any of the following actions:

- Update the resource from the branch.
- Commit your version of the resource to the branch.
- Merge your work with the changes in the branch resource.

Updating the Resource from the Branch

To update the resource from the branch in C++ perspective:

1. Select the required project for synchronization from **Project Explorer** on the left pane.
2. Right-click and select **Team**.
3. Click **Synchronize with Repository**.
4. Open the **Team Synchronizing** perspective, if prompted.

Committing Changes to the Branch

To commit your changes to the branch:

1. On the menu bar, click **Window**.
2. Click **Open Perspective**.
3. Select **Team Synchronizing**.

   The **Team Synchronizing** perspective and **Synchronize** window appear on the left pane.
4. Right-click the project you want to synchronize.
5. Select **Synchronize**.

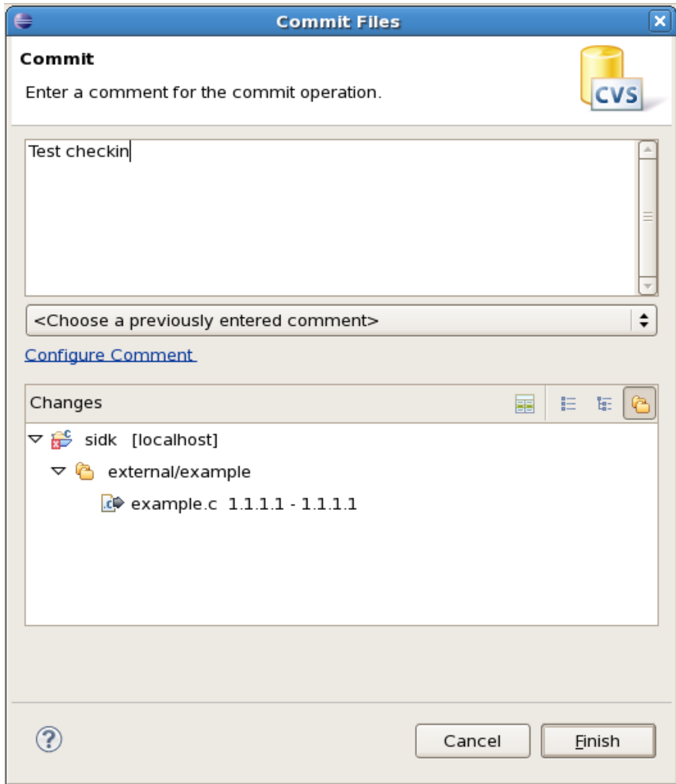
Committing Changes to the CVS Repository

You can commit the finalized changes to the CVS repository to enable other users to view them. To commit the changes, use any of the following two methods:

- Team menu
- Synchronize View

To commit changes to the CVS using the Team menu:

1. Right-click the file in the left pane under **Project Explorer** window.
2. Click **Team**.
3. Click **Commit**.
4. Enter the appropriate comment for the modifications in file in the **Commit Files** window.



5. Click **Finish**.

Note You should update before you commit the changes to avoid conflicts with the resources in your workspace as well as in the branch. Refer Updating the Workspace with the CVS Repository.

After committing the changes successfully, note the change in revision number of the file in the Project Explorer window in the left pane.

Updating the Workspace with the CVS Repository

While a project is in progress in a workspace, multiple users may commit changes to the copy of that project in the repository. You may want to update the workspace so that these changes are reflected in the branch. These changes are specific to the branch that your workspace project is configured to share. You control when you choose to update.

You can update the workspace by either:

- Updating the menu
- Synchronizing the view

In order to understand the difference between these two methods, it is important to know the three different types of incoming changes:

- *Non-conflicting change*. Occurs when a file is changed remotely but is not modified locally.
- *Auto-mergeable conflicting change*. Occurs when an ASCII file is changed both remotely and locally (that is, it has non-committed local changes), but the changes are on different lines.
- *Non-auto-mergeable conflicting change*. Occurs when one or more lines in an ASCII file, or a binary file is changed both remotely and locally (binary files are not auto-mergeable).

Updating the Workspace using the Team Menu

When you select Update under the Team menu, the contents of the local resources are updated with the incoming changes of all the above three types. You can specify the expected update behavior in the Preferences window. The available options are:

- *Preview all incoming changes before Updating*. All changes are displayed in either the Synchronization view or in a pop-up window, depending on your settings. You can then merge each change line by line, or update all non-conflicting changes at once and then deal with the remaining conflicts.
- *Update all non-conflicting changes and then preview the remaining changes*. All non-conflicting incoming changes are merged in automatically and any remaining conflicts are displayed either in the Synchronization view (default) or in a pop-up window. You can specify the location to display the conflicts from the Update/Merge preference page.
- *Never preview and use CVS text markup to indicate conflicts*. This option automatically merges all changes in without any user interaction. Conflicting changes are merged in using the CVS text markup as follows:

  ```
  <<<<<<< original file revision
  [original code]
  = = = = = = =
  [incoming code]
  >>>>>>> incoming file revision
  ```

  You can then go into each file that contains a merged conflict and edit the file to the desired final state.

Updating the Workspace using the Synchronize View

Before updating any local resources, you have to know about the incoming changes. You can use the Synchronize view to know about the incoming changes.

To open the Synchronize view in incoming mode:

1. In one of the navigation views, select the resources (files or directories) that you want to update.
2. Right-click and select **Team**.
3. Select **Synchronize with Repository**.
4. On the toolbar of the Synchronize view, click the **Incoming mode** icon to filter out any modified workspace resources (outgoing changes).

Using the Update Option in Incoming Mode

In incoming mode, you can view the changes committed to the branch since last update. The view indicates the type of each incoming change. There are two update commands (available from the context menu of any resource in the view) to deal with the different types of conflicts: Update and Override and Update.

- *Update*. This command in the Synchronize view processes all the selected incoming and auto-mergeable conflicting changes. The conflicts that are not auto-mergeable are not updated. The files that are successfully processed are removed from the view.

  The Structure Compare pane at the top of the Synchronize view contains the hierarchy of resources with incoming changes.

  To update non-conflicting and auto-mergeable files:

  - Select all conflicting files and choose **Update** from the pop-up menu.

  This updates the selected resources that are either incoming changes or auto-mergeable conflicts and remove them from the view. Conflicts whose contents are not auto-mergeable are still present the view.
- *Override and Update*. This command operates on conflicts and replaces the local resources with remote contents. This option is not recommended.

The behavior of the Override and Update command described above only applies to the incoming mode of the Synchronize view. In the Incoming/Outgoing mode of the view, the behavior for incoming changes and conflicts is as described here, but the command reverts the outgoing changes to the repository contents. Exercise great caution while using this command in Incoming/Outgoing mode.

Merging the Changes Manually

If your local workspace contains any outgoing changes that are not auto-mergeable with incoming changes from the branch, then, instead of using the Override and Update command, you can merge the differences into your workspace manually.

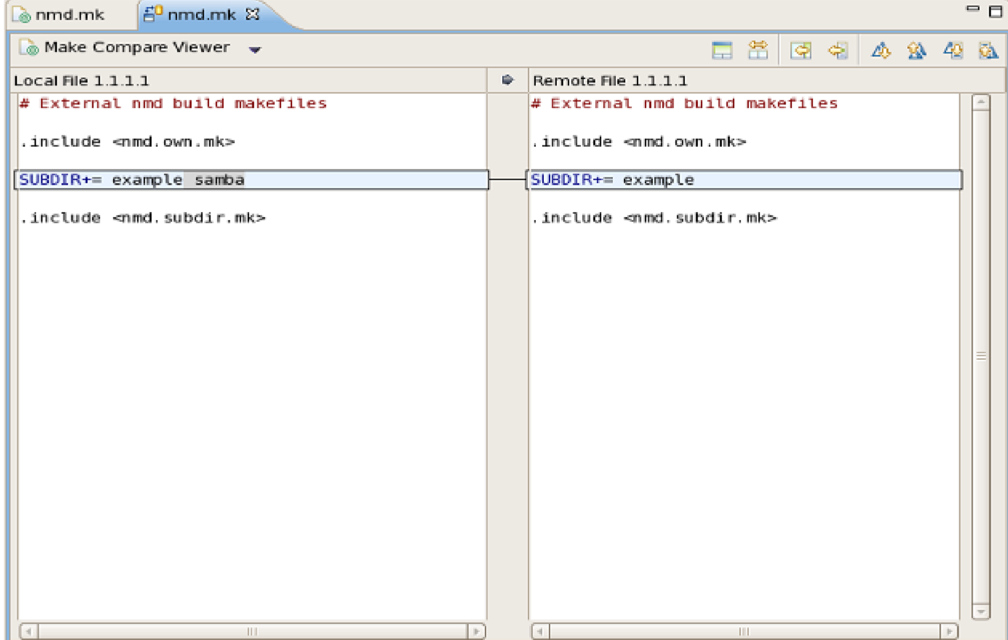
To merge the changes manually:

1. In the **Structure Compare** pane, double-click the resource to open the resource or select **Open** in **Compare Editor** from the context menu (if there is a conflict in the resource list represented by red arrows).
2. In the **Text Compare** area of the compare editor, examine the differences between the local workspace data (on the left) and repository branch data (on the right).
3. Use the **Text Compare** area to merge any changes.

   You can copy changes from the repository revision of the file to the workspace copy of the file and save the merged workspace file.
4. Choose **Mark as Merged** from the pop-up menu in the Synchronize view after merging the remote changes into a local file.

   This marks the local file as updated and allows your changes to be committed.

The following figure displays the Compare Editor with file nmd.mk file in external directory. You can open the file by right-clicking the file in the Synchronize view and selecting **Open in Compare Editor**.



The repository contents are not changed upon update. When you accept incoming changes, these changes are applied to the workspace. The repository is only changed when you commit the outgoing changes.

In the Synchronize view, selecting an ancestor of a set of incoming changes performs the operation on all the appropriate children. For instance, selecting the top-most folder and choosing Update processes all the incoming and auto-mergeable conflicting changes and leaves all other incoming changes unprocessed.

Updating and Committing Changes in the Synchronizing Perspective

This section describes the procedure to commit and update changes in the Team Synchronizing perspective.

To update in synchronizing perspective:

1. Click the **Update all Incoming Changes** icon.
2. Click the **Incoming Mode** icon to view all incoming changes.

To commit changes:

1. Click the **Commit all Outgoing Changes** icon.
2. Click the **Outgoing Mode** icon to view all outgoing changes.

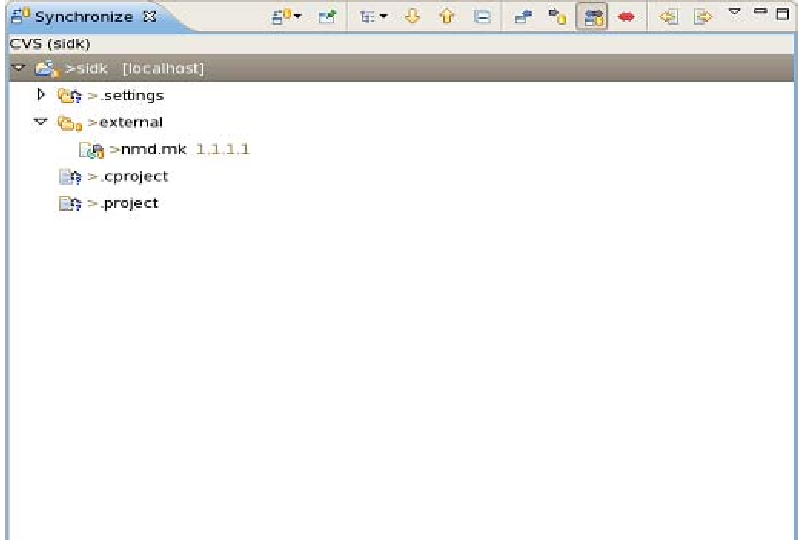
Note You can also perform Update and Commit operations on the resources by selecting the resources and selecting the desired option from the right-click menu.

If you do not want to commit some of the changed resources, you can remove them from the view.

To remove the changed resources from the view:

- Right-click on the resource in Synchronize view and select **Remove from View**.

For example, in the following figure .settings, .cproject and .project in Synchronize view are created by Eclipse. You can delete them from the Synchronize view to remove them from the repository.



Replacing Resources in the Workspace

To replace workspace resources with versions in the repository:

1. Select a resource in one of the navigation views.
2. From the resource's pop-up menu, select one of the following menu items:
   - *Replace With > Latest From <Branch Name> or <Version>*. Replaces the workspace resource with the latest resources currently committed to the branch that the local project is shared with or if the local project is checked out as a version then replaced with the same version.
   - *Team > Revert to Base*. Replaces the workspace resource with the last checked out revision. Deleted files are restored.
   - *Replace With > Another Branch or Version*. Replaces the workspace resource with a specific version or branch that you select in the repository. When you select this option, a window appears that enables you to browse through the branch and version tags in the repository.
   - *Replace With > History...*. Replaces the workspace file with another revision of the file. Based on your preference settings, either the History view or a pop-up opens that contains a table of all available revisions of the selected file. Selecting one of the revisions displays the differences between the workspace revision and the selected revision. To replace the workspace revision, select the revision and choose Get Contents from the context menu. This replaces the contents of the workspace file with the selected revision.

     Note This option is available only on a file.

   - *Team > Switch to Another Branch or Version*. Replaces the workspace resources with those on the tag specified in the Update dialog. Unlike the above replace operations, the uncommitted changes in the workspace are not replaced but are "moved" to the resources from the selected tag.

While replacing with a branch or version or while updating, you can specify a particular date instead of a version or branch tag. Right-click **Dates** category in the tag selection list and select **Add Date**. A **Date Tag** pop-up appears that allows you to specify a date and optionally a time. After you click **OK**, the date tag appears in the tag selection list.

CHAPTER 10

Optimizing Eclipse for the SIDK Project

There are various settings in Eclipse that you can optimize for the SIDK project. You can perform the following tasks to optimize Eclipse for the SIDK project:

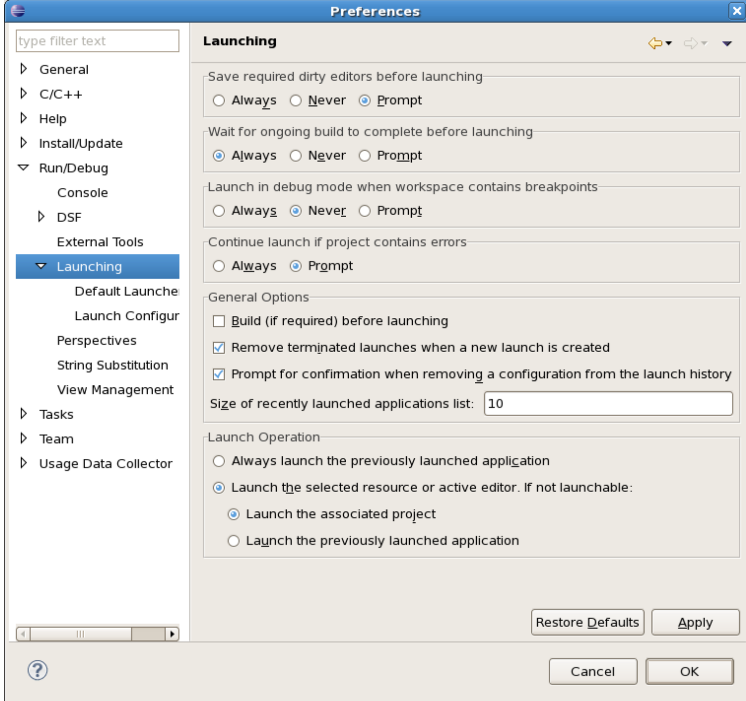
- Disabling the Build before Launching
- Disabling the C/C++ Indexer
- Increasing the Build Console Limit

Disabling the Build before Launching

While running or debugging the application, the build is automatically triggered first. It occurs because the option to build before launching is enabled by default.

To prevent the build from triggering automatically before running or debugging:

1. Select **Window**.
2. Select **Preferences**.
3. On the **Preferences** window, expand **Run/Debug** in the left pane.
4. Select **Launching**.



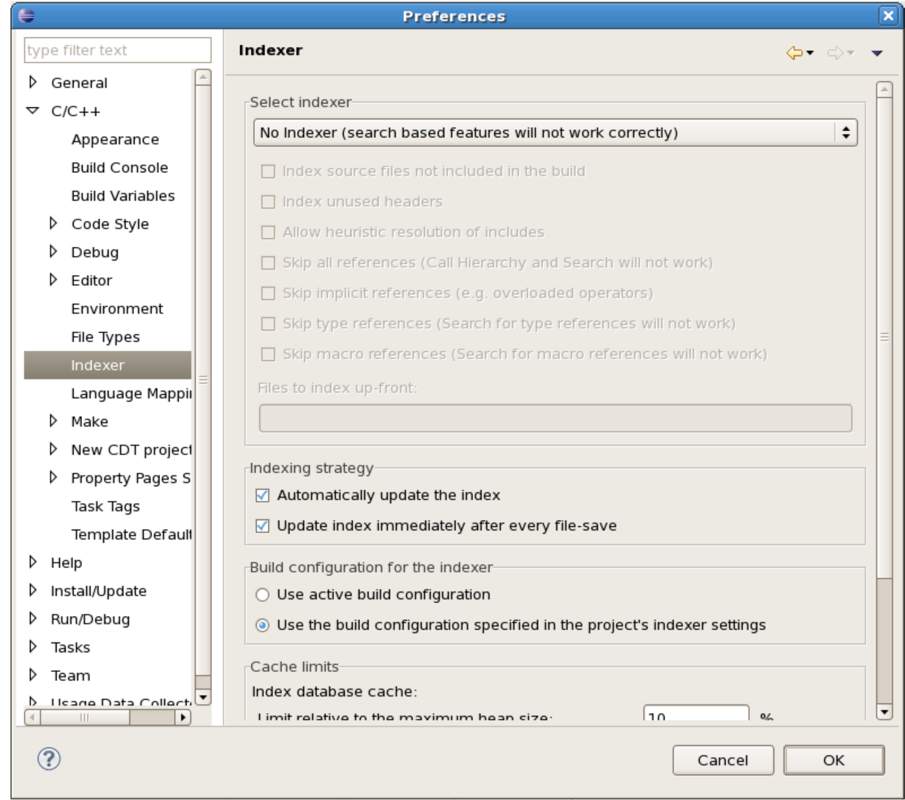
5. On the **General Options** panel, clear the **Build (if required) before launching** check box.
6. Click **Apply**.
7. Click **OK**.

Disabling the C/C++ Indexer

While creating a build using Eclipse or while modifying and updating the code, the C/C++ indexer runs to make indices for the source code. It may hamper the performance of Eclipse. This can be prevented by disabling the C/C++ indexer.

To disable the C/C++ indexer:

1. Select **Window**.
2. Select **Preferences**.
3. On the **Preferences** window, expand **C/C++** in the left pane.
4. Select **Indexer**.
5. On the **Select Indexer** panel in the right pane, select **No Indexer (search based feature will not work correctly)** from the drop-down list.



6. Click **Apply**.
7. Click **OK**.

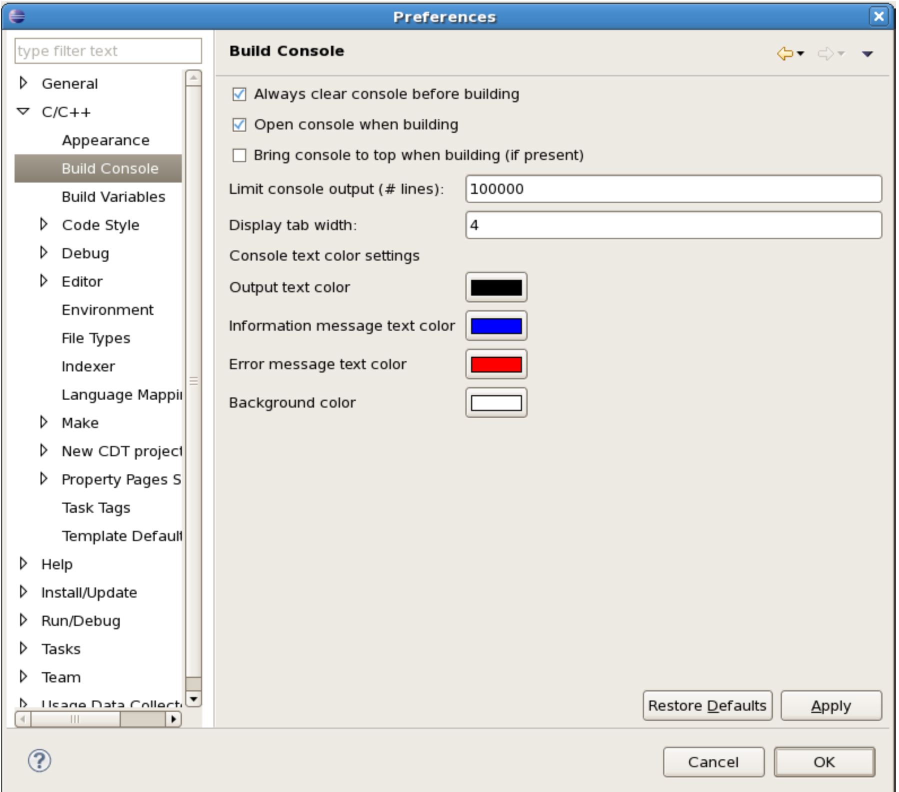
Increasing the Build Console Limit

By default the build console limit is set to 500 lines. You may need to increase it to view more build information at a time.

To increase the build console limit:

1. Select **Window**.
2. On the **Preferences** window, expand **C/C++** in the left pane.

3. Select **Build Console**.



4. Edit the value of **Limit console output (# lines)** textbox to the desired value.
5. Click **Apply**.
6. Click **OK**.

Similarly, you can also configure other settings (such as bringing console to top when building or changing the output text color) for the console with the available options.

APPENDIX A

Xen Installation and Configuration

This section describes the procedures for installation and configuration of Xen. You must have the Xen virtualization environment to build target image for the Xen project.

Verifying Virtualization Support

Prior to installing Xen, verify that the processor on your machine supports virtualization.

- To verify virtualization support for Intel processor, use the following command:

  ```
  ~# grep vmx /proc/cpuinfo
  ```

  -or-

  ```
  ~# cat /proc/cpuinfo | grep vmx
  ```

  The appearance of vmx flag in the output confirms Intel processor support.

- To verify virtualization support for AMD processor, use the following command:

  ```
  ~# grep svm /proc/cpuinfo
  ```

  -or-

  ```
  ~# cat /proc/cpuinfo | grep svm
  ```

  The appearance of svm flag in the output confirms AMD processor support.

If the processor supports virtualization, ensure that the flag is turned on in the BIOS settings.

Installing and Configuring Xen

To install and configure Xen:

1. Verify that SELinux is disabled or permissive:

   ```
   [root@server1 ~]# vi /etc/sysconfig/selinux
   ```

   ```
   # This file controls the state of SELinux on the system.
   # SELINUX= can take one of these three values:
   #       enforcing - SELinux security policy is enforced.
   #       permissive - SELinux prints warnings instead of enforcing.
   #       disabled - SELinux is fully disabled.
   SELINUX=disabled
   # SELINUXTYPE= type of policy in use. Possible values are:
   #       targeted - Only targeted network daemons are protected.
   #       strict - Full SELinux protection.
   SELINUXTYPE=targeted
   ```

   Reboot the system if you edited the selinux file located in /etc/sysconfig/ directory:

   ```
   [root@server1 ~]# reboot
   ```

2. Run the following command to install Xen and Xen kernel on CentOS:

   ```
   [root@server1 ~]# yum install kernel-xen xen
   ```

3. Verify the GRUB bootloader configuration.

   Edit the menu.lst file located in the /boot/grub/ directory and add `xenconns=tty6` for both kernel:

   `[root@server1 ~]# vi /boot/grub/menu.lst`

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#          initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-164.15.1.el5xen)
        root (hd0,0)
        kernel /xen.gz-2.6.18-164.15.1.el5
        module /vmlinuz-2.6.18-164.15.1.el5xen ro root=/dev/VolGroup00/
LogVol00 rhgb quiet xencons=tty6
        module /initrd-2.6.18-164.15.1.el5xen.img
title CentOS (2.6.18-164.el5xen)
        root (hd0,0)
        kernel /xen.gz-2.6.18-164.el5
        module /vmlinuz-2.6.18-164.el5xen ro root=/dev/VolGroup00/LogVol00
rhgb quiet xencons=tty6
        module /initrd-2.6.18-164.el5xen.img
```

4. Configure `xend` to start automatically:

   `[root@server1 ~]# /sbin/chkconfig xend on`

5. Reboot the system:

   `[root@server1 ~]# reboot`

   The system should automatically boot the new Xen kernel.

6. Verify that the system has booted the new Xen kernel:

   `[root@server1 ~]# uname -r`

   `2.6.18-164.15.1.el5xen`

7. Verify xend status:

   `[root@server1 ~]# /etc/init.d/xend status`

8. Start xend if it is not running:

   `[root@server1 ~]# /etc/init.d/xend start`

9. Verify that Xen has started:

```
[root@localhost sbin]# /usr/sbin/xm list
Name                                      ID Mem(MiB) VCPUs State   Time(s)
Domain-0                                   0     2588     8 r-----    103.7
```

   In the output, `Domain-0` (dom0) indicates that xen has started.

10. Edit xend-config.sxp and replace `network-script network-bridge` with `(network-script network-multi-bridge)`:

    `[root@server1 ~]# gedit /etc/xen/xend-config.sxp`

11. Create the network-multi-bridge script:

    `[root@server1 ~]# gedit /etc/xen/scripts/network-multi-bridge`

    Add the following to the empty file:

```
#!/bin/sh
 dir=$(dirname "$0")
 "$dir/network-bridge" "$@" vifnum=0 bridge=xenbr0 netdev=eth0
```

12. Set the file permissions for the script:

    `[root@server1 ~]# chmod a+x /etc/xen/scripts/network-multi-bridge`

13. Edit the xen_domU.conf file located in /sidk/builddir.xen/target_dir/:

    `[user5@localhost target_dir]$ vi xen_domU.conf`

    Comment the original “vif” line and add a new one:

```
import os, re
arch = os.uname()[4]
if re.search('64', arch):
    arch_libdir = 'lib64'
else:
    arch_libdir = 'lib'

kernel = '/usr/lib/xen/boot/hvmloader'
builder = 'hvm'
memory = '512'
device_model = '/usr/' + arch_libdir + '/xen/bin/qemu-dm'
#vif = [ "type=ioemu, bridge=xenbr0",
#        "type=ioemu, bridge=xenbr1" ]
vif = [ "type=ioemu, bridge=xenbr0" ]

disk = [ 'file:/home/user5/workspace/sidk/builddir.xen/target_dir/
virt_disk,ioemu:hda:disk,w',
         'file:/home/user5/workspace/sidk/builddir.xen/target_dir/
boot.iso,ioemu:hdc:cdrom,r' ]
name='nmd'
boot='d'
vnc=1
```

14. Configure /etc/sudoers to run the run_xen script.

    Add `userX ALL=(ALL)ALL` under `root ALL=(ALL)ALL`.

15. Start run_xen:

    `[user1@server1 ~]# ./workspace/sidk/builddir.xen/target_dir/run_xen`

16. Start virt-manager:

    `[root@server1 ~]# /usr/sbin/virt-manager`

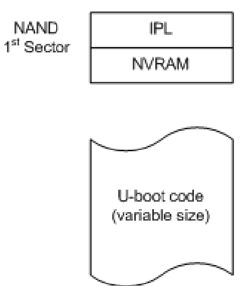
17. Double-click **nmd** in the virt-manager window.

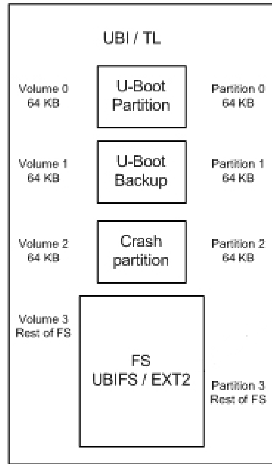
    A window appears that runs the SIDK image.

APPENDIX B

Flash Partitions

This section describes the partitions of Flash memory on the target platform.





The Flash memory is distributed as per the following partitions:

- The initial flash partition comprises NAND. The first sector of NAND is shared by IPL and NVRAM data.
- The next partition is of random size for U-Boot.
- After the U-Boot partition, the Flash memory contains UBI volumes or TL partitions.
  - Volume 0/Partition 0. Contains 64 KB U-Boot parameters.
  - Volume 1/Partition 1. Contains a 64 KB backup of U-Boot parameters.
  - Volume 2/Partition 2. Contains a 64 KB crash partition.
  - Volume 3/Partition 3. Contains a UBIFS or EXT2 file system. The rwdata folder has sys1 and sys2 folders that contain the kernel image and SquashFS file system.

APPENDIX C

Boot Sequence

This section describes the boot sequence on the target platform after the target image is flashed on it.

At the end of kernel booting, initramfs (which is a part of kernel) is mounted. The script in initramfs mounts a squashfs image present in the jffs2 file system. This squashfs becomes the new root file system, and the jffs2 file system is mounted under the rwdata directory.

Appd continues to bring up the user space applications and loads the kernel modules.