

XDC Getting Started Guide

March 13, 2008



IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products	Applications		
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated

Read This First

About This Manual

This document provides an overview and short introduction to the XDC tools.

How to Use This Manual

This document provides installation instructions for XDC in Chapter 1, step-by-step instructions for starting to use XDC in Chapter 2, and information about learning more in Chapter 3.

After you install XDC, you might want to review the release notes in the installation before reading further.

After you have read this document, you should see the *XDС Consumer User's Guide* (SPRUEX4) and the online CDOC reference for more information.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, and interactive displays are shown in a special typeface. Examples use a bold version of the special typeface for emphasis.

Here is a sample program listing:

```
#include <xdc/runtime/System.h>

int main(){
    System_printf("Hello World!\n");
    return(0);
}
```

- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets. Unless the square brackets are in a bold typeface, do not enter the brackets themselves.

Related Documentation from Texas Instruments

XDС Consumer User's Guide (SPRUEX4)

CDOC API Reference online help system (see Section 3.1)

Trademarks

The Texas Instruments logo and Texas Instruments are registered trademarks of Texas Instruments. Trademarks of Texas Instruments include: TI, Code Composer, Code Composer Studio, DSP/BIOS, SPOX, TMS320, TMS320C54x, TMS320C55x, TMS320C62x, TMS320C64x, TMS320C67x, TMS320C28x, TMS320C5000, TMS320C6000 and TMS320C2000.

Windows is a registered trademark of Microsoft Corporation.

Linux is a registered trademark of Linus Torvalds.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

Read This First	iii
About This Manual	iii
How to Use This Manual	iii
Notational Conventions	iii
Related Documentation from Texas Instruments.....	iii
Trademarks	iv
Contents	v
Installing and Setting Up XDC	1
1.1 What is XDC?	1
1.2 Installing XDC.....	2
1.2.1 Notes about Network Installations	2
1.3 Adding XDC to Your PATH	3
1.4 Setting Your XDCPATH	3
1.5 Setting an Environment Variable in Microsoft Windows..	3
1.6 Testing Your XDC Installation	4
Creating Your First XDC-Based Application	5
2.1 Overview of Configuration	6
2.2 Configuring the Application	7
2.3 Writing C Code	8
2.4 Integrating a Configuration into Your Build Flow.....	8
2.4.1 Choosing a Target.....	8
2.4.2 Choosing a Platform.....	9
2.4.3 Running Configuro	9
2.4.4 Integrating Configuration into a CCStudio Project	9
2.4.5 Integrating Configuration into Command Line Build Flows.....	11
2.4.5.1 Integrating Configuration with TI Code Gen Tools in a Makefile.....	11
2.4.5.2 Integrating Configuration with Microsoft C/C++ Tools in a Makefile	12
2.4.5.3 Integrating Configuration with GNU gcc Tools in a Makefile.....	12
Learning More	13
3.1 Using the API Reference Help System (CDOC)	13
3.2 List of XDC Tools	14
3.3 Other Documents	14
3.4 The Package Path.....	15
3.4.1 Managing the Package Path.....	15
3.5 Terminology.....	16

Installing and Setting Up XDC

This chapter describes how to install the XDC tools and prepare to use them.

Topic	Page
1.1 What is XDC?	1
1.2 Installing XDC.....	1
1.3 Adding XDC to Your PATH.....	3
1.4 Setting Your XDCPATH	3
1.5 Setting an Environment Variable in Microsoft Windows	3
1.6 Testing Your XDC Installation.....	4

1.1 What is XDC?

The eXpress DSP Components (XDC, pronounced "ex-dee-see") is a standard for providing reusable software components, called "packages", that are optimized for real-time embedded systems.

XDC includes tools and standards for API development, static configuration, and packaging. The main benefit of XDC is that it standardizes the delivery and use of target content and makes it easier to include target content from others in applications.

This document helps you get started using XDC. It demonstrates a "Hello World" program.

For more information about XDC and how to use it, see the *XDC Consumer User's Guide* (SPRUEX4). The XDC release notes (XDC_INSTALL_DIR/release_notes.html) include information about software version, upgrades and compatibility, host and target device support, validation, and known issues.

An XDC "package" is a named collection of files that form a unit of versioning, update, and delivery from a producer to a consumer. Each package is embodied as a specially named directory (and its contents) within a file system. Packages are the focal point for managing content throughout its life-cycle. All packages are built, tested, released, and deployed as a unit.

An XDC "repository" is simply a directory that contains packages. The dots in the name of a package, interface, or module refer to its location within the repository. For example the ti.sysbios.knl.Task module would be located at ti/sysbios/knl with respect to a repository directory named in the "package path". The package path is a list of repositories containing package installed by the user.

1.2 Installing XDC

Some software products install a version of the XDC Tools automatically. If you haven't already installed the XDC Tools as standalone software or as part of another application, follow these steps. (If you have already installed the XDC Tools, skip to the next section.)

- 1) Place the distribution file into a temporary location. The distribution file for Windows is called `xdc_setupwin32_n_nn.exe`, where *n* is a digit in the version number. The distribution file for Linux is called `xdc_setuptoolslinux_n_nn.bin`. Additional digits are added to the version number for patch releases.
- 2) If you are installing on a Windows PC, you should install from an account that has administrator privileges. (The Linux installer does not require root privileges.)
- 3) The distribution file contains a standard installer.
 - **Windows.** Double-click on the file to start the installer.
 - **Linux.** Run the Linux installer on the command line or by double-clicking the file.
- 4) Follow the installation prompts.
 - You can choose an installation directory (`XDC_INSTALL_DIR`) anywhere on your system. The installer automatically adds a subdirectory that includes the version number (for example, `xdc_2_95`) to the location you select.

Important Note: Make sure the location has **no spaces in the full path** to the directory. If you are upgrading to a new version of XDC, be sure to change any hardcoded references in your application files to the old XDC directory.

 - On Windows systems, you can choose either a "Typical" installation or a "Network" installation. The difference is that "Network" installations do not modify the Windows registry to provide support for Code Composer Studio and uninstalling. See Section 1.2.1 for more details. Most single-user systems should have a "Typical" install.

If you later want to uninstall XDC, you can use the uninstaller provided.

For release information about the version you install, see the `release_notes.html` file in the top-level directory.

1.2.1 Notes about Network Installations

The network installation option allows a single installation of the tools to be used by multiple developers over a network filesystem. If you select the network installation, the Windows registry is not modified to provide support for Code Composer Studio and uninstalling. However, since no files or directories outside the installation directory are modified, you can uninstall a "Network" installation simply by removing the installation directory.

Currently, the only XDC package that requires Windows registry modification to be fully functional is `xdc.tools.configuro.ccs`. You can make this registry modification after a Network installation, if you wish. See the documentation for the `xdc.tools.configuro.ccs` package for more information.

1.3 Adding XDC to Your PATH

Setting the PATH definition makes it easy to run XDC's xs command, which is used to run a number of tools. After installing XDC, you can add the top-level XDC directory to your existing PATH definition. See Section 1.5 for how to change your PATH definition on Windows. Only the top-level XDC directory should be added to the PATH definition.

If you do not want to add the XDC directory to your PATH, you can use one of the following alternate ways of running the xs command:

- You can invoke xs from the command line by using the full path <xdc_install_dir>/xs.
- You can make an alias to the xs or xs.exe executable.

1.4 Setting Your XDCPATH

If you have other software that is delivered in the form of packages (such as DSP/BIOS), you also need to set the XDCPATH environment variable to reference the repository containing those packages. The XDCPATH definition allows XDC to find all the XDC-compliant packages that have been installed.

See Section 1.5 for how to change your XDCPATH definition on Windows. See Section 3.4 for more information about the XDCPATH and how packages are located.

The XDCPATH value is a semicolon-separated list of locations.

IMPORTANT NOTE: All directories referenced in XDCPATH, whether in an environment variable or a command line, must use **forward slashes** (/) as separators instead of the Windows-standard backslash (\). For example, use "c:/os;c:/drivers;c:/framework".

If you don't want to specify environment variables, you can use the --xdcpath command-line option to pass the XDCPATH value to all XDC tools that require a package path.

1.5 Setting an Environment Variable in Microsoft Windows

Your Microsoft Windows PATH defines where to look for command-line programs. The PATH is managed by Windows itself (in Windows 2000 and Windows XP) rather than by the autoexec.bat or autoexec.nt files used in previous versions of Windows.

To change the PATH or XDCPATH (and other system environment variables), follow these steps:

- 1) Right-click My Computer and choose Properties.
- 2) In the System Properties window, choose the **Advanced** tab.
- 3) At the bottom of the Advanced tab, click the **Environment Variables** button.
- 4) Highlight "Path" or "XDCPATH" in the "Systems variables" section and click **Edit**. If the XDCPATH variable does not yet exist, click **New**.
- 5) Add the directory in which you installed XDC to the end of the current Path definition. Use a semicolon to separate the locations in the list. For example, you might add the following to your existing Path definition:

;c:\xdc_2_95

- 6) Click **OK** in the Environment Variables window and in the System Properties window.

1.6 Testing Your XDC Installation

To test the XDC installation, type the following in a command window:

```
xs --version
```

You should see a message similar to this:

```
xs (XDC Script Interpreter) xdcutils-c11, Jun 9 2007
```

As a second test, type the following command to use one of the XDC tools:

```
xs xdc.tools.echo hello world
```

You should see "hello world" echoed back.

Creating Your First XDC-Based Application

This chapter describes how to create a simple XDC application. This helps introduce the steps in the development process.

Topic	Page
2.1 Overview of Configuration	6
2.2 Configuring the Application.....	7
2.3 Writing C Code.....	8
2.4 Integrating a Configuration into Your Build Flow	8

Creating applications that use XDC-based software packages follows a development cycle that builds on traditional C programming techniques. XDC packages standardize the delivery of TI and 3rd party software and ease the integration of such software.

Using an XDC package adds a configuration step to the conventional compile/link cycle. The configuration processing is easily added to existing build flows. XDC provides a number of tools that simplify the task of integrating and using XDC-based packages.

Once the configuration processing is integrated into your build process, the steps for creating an application that uses XDC-based content are as follows:

- 1) Configuring the application.
- 2) Writing C code.
- 3) Compiling and linking the application.

As with the standard development cycle, the focus is on writing C code. The other steps are simple. XDC provides tools to support the additional configuration and processing steps.

This chapter shows how to develop a “Hello World” C example using these steps. Before that, the section that follows provides an overview of XDC configuration and what it does.

2.1 Overview of Configuration

Any application that uses an XDC package needs a configuration file. This is a file containing statements that configures the contents of the packages used by the application. Your C program code can do additional dynamic configuration, but the XDC configuration defines the starting point.

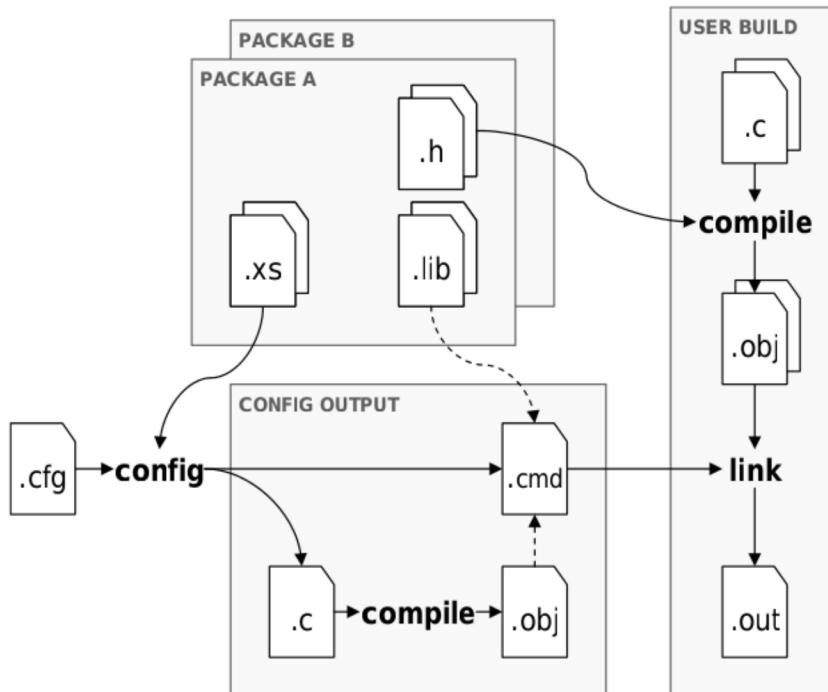
The configuration file uses simple JavaScript (EcmaScript) syntax. For more information about configuration, see the *XDC Consumer User's Guide* (SPRUEX4).

The XDC configuration serves the following purposes:

- Specifies the modules and packages to use and static objects to create.
- Performs integrity checks between specified and dependent packages.
- Sets configuration variables (options) for modules and objects to change their default runtime behavior.

The XDC configuration framework enables packages to customize their content based on your hardware platform and your application's requirements. Configuration scripts specify your application's requirements, and XDC configuration framework uses these scripts to generate files that must be integrated into your normal application build:

The following figure shows how these files fit into the typical build process. XDC provides tools to simplify the integration of these files into your build environment.



The XDC configuration file is a .cfg file. When you process the .cfg file, the files shown in the "Config Output" block are generated. This is a simplified diagram. Actually, processing the .cfg file generates several other files that, while useful, are not required for the examples shown in this document.

As a consumer of XDC-based content, you don't need to know about all of the generated files. Two files of particular interest are:

- `<appcfg>.obj` – An object file created by compiling a generated `<appcfg>.c` file that contains static data structures and functions referenced by the packages named (directly or indirectly) by the configuration script.
- `<appcfg>.cmd` – A linker command file, also generated by the configuration process, that contains the list of libraries supplied by the packages referenced by the configuration script as well as platform-specific memory definition and placement information.

When you are learning to use XDC, you will likely use the `xdc.tools.configuro` utility described later in this chapter to process your configuration file. That utility generates the following additional important files from your configuration file:

- **compiler.opt** – Contains compiler command-line options for your target and platform. This file allows you to easily compile your application's sources using the same options used to create `<appcfg>.obj`. You can thus avoid bugs that would result from linking together files compiled with mismatched runtime assumptions.
- **linker.cmd** – Corresponds to the `<appcfg>.cmd` file. Contains the linker options required to use the content from the packages required by the application.

It is these two files that are commonly integrated into your build flow. As you develop and troubleshoot your own application, you may want to review the figure earlier in this section to see which files need to use and reference others.

2.2 Configuring the Application

To develop your "Hello World" C example, first create a new directory called "hello" to contain your test application. The directory can be anywhere, but it is best to avoid directory paths that contain spaces. For example, your location might be `C:\myprojects\hello` or `/home/me/projects/hello`.

In the "hello" directory, use a text editor to create a configuration file called `mycfg.cfg` for the application. Type the following JavaScript statement in the file:

```
var System = xdc.useModule("xdc.runtime.System");
```

When you save the file, be sure to save it as plain text (not as a Microsoft Word or other word processor file).

This `xdc.useModule()` method enables an application to use the `xdc.runtime.System` module.

In more complex applications, you will add additional configuration statements to this file to use various modules, set properties of modules, and create and configure instances of the objects managed by these modules. For example, where "Mod" is a module you want to use, the statements might be similar to the following:

```
var Mod = xdc.useModule("pkgname.Mod");
Mod.cfgParam1 = value1;
Mod.cfgParam2 = value2;
```

As you develop applications, the configuration step is one you may revisit as you add additional functionality to your C code.

2.3 Writing C Code

In your “hello” directory, use a text editor to create a `hello.c` C code file for the application. Type the following statements in the file:

```
#include <xdc/std.h>
#include <xdc/runtime/System.h>

int main(){
    System_printf("Hello World!\n");
    return (0);
}
```

The first `#include` statement is required in all C code that uses XDC APIs. The second `#include` statement allows the C code use the same `xdc.runtime.System` module that you enabled in the `mycfg.cfg` file. The `System_printf()` function is provided by that module.

For more information about coding for XDC-based applications, see the *XDConsumer User’s Guide* (SPRUEX4).

2.4 Integrating a Configuration into Your Build Flow

In this step, you use the “configuro” tool provided with XDC to process the `mycfg.cfg` configuration file you created. Processing the file generates a `compiler.opt` file to be used when you compile the application and a `linker.cmd` file to be used when you link the application. The `compiler.opt` file allows you to easily compile your application’s sources using the same options used to create `<appcfg>.obj`. To simplify your link step, the `linker.cmd` file is the `<appcfg>.cmd` file with an added reference to `<appcfg>.obj`.

Before a configuration script can be processed, you need to specify the “target” compiler and the hardware “platform” required to run your application. In general terms:

- A target identifies a specific compiler and an ISA and runtime model supported by the compiler. For example, the TI ‘C6000 compiler for the C64+ ISA running in big endian mode.
- A platform identifies the hardware execution environment as seen by your application. For example, a DM6446 EVM with 64 MB of DDR2 external memory.

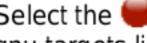
XD uses simple string names to identify targets and platforms.

2.4.1 Choosing a Target

Before you can process a configuration script, you need to find the target and platform strings to specify when processing the file and you will specify the target and platform in the configuro command line you use to process your configuration file. See Section 2.4.3.

Some example target strings are:

<code>ti.targets.C64</code>	<code>ti.targets.C64_big_endian</code>
<code>ti.targets.C64P</code>	<code>ti.targets.C64P_big_endian</code>
<code>gnu.targets.Linux86</code>	<code>gnu.targets.Mingw</code>
<code>microsoft.targets.Net32</code>	<code>microsoft.targets.Win32</code>

To see the latest list of targets, open the CDOC online documentation (Section 3.1). Select the  (XDC) view. You can expand the `ti.targets`, `microsoft.targets`, and `gnu.targets` lists. See the *XDConsumer User’s Guide* (SPRUEX4) for a list of targets and details about compiler options required by various targets.

2.4.2 Choosing a Platform

The “platform” is a name that describes a specific board and its settings. It specifies a particular device and memory map on which an application will run. It is used during the configuration process by packages that require platform-specific information, such as the memory map, the initial CPU clock speed, or the revision of the device running the application.

The full list of platforms available is visible in the CDOC online documentation in the list of packages under `ti.platforms`. (See Section 3.1 for information about using CDOC.) Examples include:

```
ti.platforms.sim6xxx  
ti.platforms.dsk5510  
ti.platforms.evmDM6437
```

If you are developing for a platform that is not listed above, you may be able to use the `ti.platforms.generic` platform. For more information about how to define a platform instance using this platform package, see the *XDConsumer's User Guide* (SPRUEX4).

2.4.3 Running Configuro

XD provides a tool called “configuro” that processes XDC configuration files like the one you created in Section 2.2.

Generate files from the configuration you created using the following command line:

```
xs xdc.tools.configuro -t <target> -p <platform>  
-c <compiler_location> mycfg.cfg
```

For example:

```
xs xdc.tools.configuro -t ti.targets.C64 -p ti.platforms.sim6xxx  
-c c:/CCStudio_v3.3/C6000/cgtools mycfg.cfg
```

- The `-t` option specifies the target package.
- The `-p` option specifies the platform package.
- The `-c` option specifies the location of the compiler you want to use.

You can view further command-line options for the configuro tool with this command:

```
xs xdc.tools.configuro --help
```

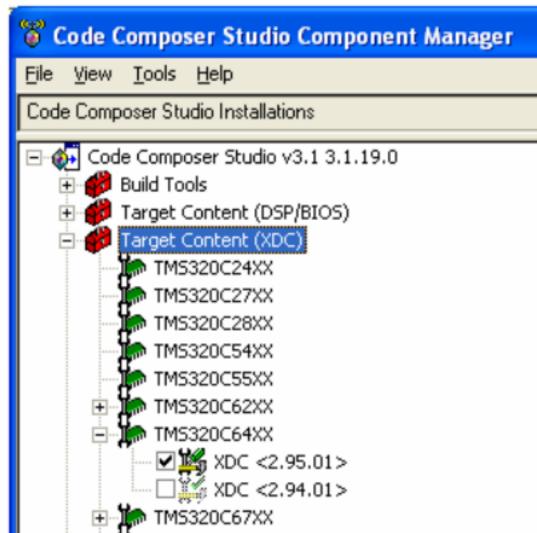
You can integrate the step of running configuro into your makefile or Code Composer Studio project file as described in the subsections that follow.

For details about using configuro, see the *XDConsumer User's Guide* (SPRUEX4).

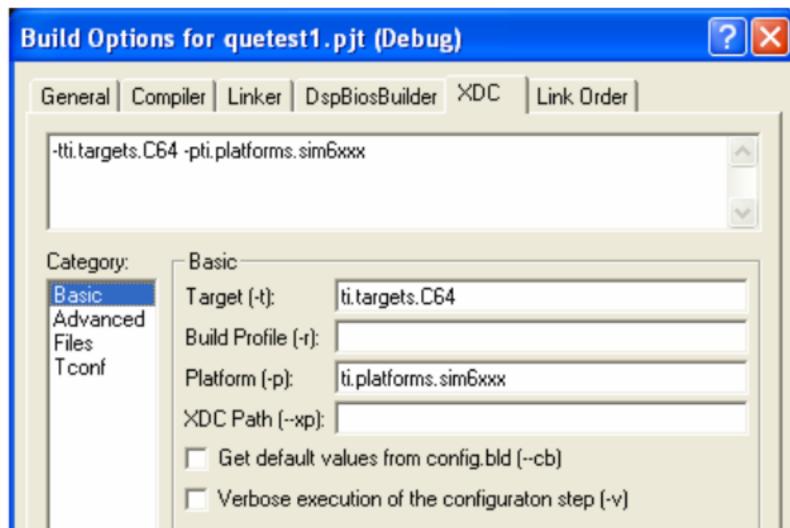
2.4.4 Integrating Configuration into a CCStudio Project

On Windows systems, if you have installed both XDC and Code Composer Studio, you can use an XDC tab in the Build Options dialog in Code Composer Studio.

- 1) First, use the Component Manager (part of CCStudio) to make sure the latest version of XDC is enabled. To run the Component Manager, use your Windows Start menu to choose **All Programs->Texas Instruments->Code Composer Studio->Component Manager**. Within Component Manager, expand Target Content (XDC). Then expand your target family. Make sure the version of XDC you want to use has a checkmark. When you close Component Manager, save your changes.



- 2) Next, add the CFG file you created to your CCStudio project.
- 3) Then, open the **Project->Build Options** dialog and move to the XDC tab:



- 4) With the Basic category selected, specify the target and platform packages you want to use. Optionally, you can also specify the -r (build profile) and --xp (XDCPATH) command-line options for configuro here.
- 5) If you use the CCStudio IDE, you can add the compiler.opt file to the project in the **Project->Build Options** dialog. In the Compiler tab, select the Files category and add \$(Proj_dir)/xdcconfig/compiler.opt in the Options File field. (Or, you can add an option in the XDC tab like -o mycfg, where mycfg is an output subdirectory, and then you can use \$(Proj_dir)/mycfg/compiler.opt in the Options File field of the Compiler tab.)
- 6) If you are using CCStudio, add the linker.cmd file to your project using the Add Files to Project command.
- 7) If you haven't already added a RTS library to your CCStudio project, use the Add Files to Project command. For example, your RTS library might be \$ (CGTOOLS)/lib/rts6400.lib.

NOTE: The RTS library must come after the linker.cmd file in the link order in a makefile or CCStudio project.

2.4.5 Integrating Configuration into Command Line Build Flows

The three sections that follow show how to run the configuro tool and the compilers and linkers for TI, GNU, and Microsoft. They use the compiler.opt and linker.cmd files generated by configuro.

The example makefiles in the following sections all use gmake, which is delivered with XDC. In gmake makefiles, always use a tab character to indent, not spaces. Also, all commands should be on a single line. Variable definitions that contain spaces (for example, CGTOOLS often does on Windows) must be surrounded by quotes.

When you run configuro from the command line, you see a message similar to the following:

```
making package.mak (because of package.bld) ...
generating interfaces for package mycfg (because
package/package.xdc.xml is older than package.xdc) ...
configuring mycfg.x64 from package/cfg/mycfg_x64.cfg ...
cl64 package/cfg/mycfg_x64.c ...
```

Re-run configuro on your CFG file whenever you do any of the following:

- Change the CFG file
- Change the target or platform
- Update a package used by your application

The configuro command uses specially generated makefiles to avoid running any unnecessary steps when you make the changes above. So, if you are not sure whether to run configuro, go ahead and run it; if nothing needs to be done, configuro quickly detects this and does nothing.

The compile and link syntax you use is compiler-dependent. See the *XD Consumer User's Guide* (SPRUEX4) for details about the compiler options placed in compiler.opt for various targets, including TI, GNU, and Microsoft targets. Additional examples are provided with XDC in the examples/configuro directory.

2.4.5.1 Integrating Configuration with TI Code Gen Tools in a Makefile

The following example makefile runs the TI Code Gen Tools compiler and linker.

```
CGTOOLS = C:/CCStudio_v3.3/C6000/cgtools

CC = $(CGTOOLS)/bin/cl6x
LNK = $(CGTOOLS)/bin/lnk6x
RTS = $(CGTOOLS)/lib/rts6400.lib

CONFIG = mycfg
XDCTARGET = ti.targets.C64
XDCPLATFORM = ti.platforms.sim6xxx

%/linker.cmd %/compiler.opt : %.cfg
    xs xdc.tools.configuro -c $(CGTOOLS) -t $(XDCTARGET) -p $(XDCPLATFORM) $<
%.obj : %.c $(CONFIG)/compiler.opt
    $(CC) -@$(CONFIG)/compiler.opt -c $<
hello.out : hello.obj $(CONFIG)/linker.cmd
    $(LNK) -o hello.out -c hello.obj $(CONFIG)/linker.cmd $(RTS)
```

2.4.5.2 Integrating Configuration with Microsoft C/C++ Tools in a Makefile

The following example makefile runs configuro and uses the compiler.opt and linker.cmd files generated by configuro in the command lines for the Microsoft Visual Studio compiler and linker.

```
CGT00LS = "C:/Program Files/Microsoft Visual Studio 8"
CC = $(CGT00LS)/vc/bin/cl
LNK = $(CGT00LS)/vc/bin/link
RTS = -nodefaultlib -libpath:$(CGT00LS)/vc/lib kernel32.lib msrvct.lib

CONFIG = mycfg
XDCTARGET = microsoft.targets.Win32

%/linker.cmd %/compiler.opt : %.cfg
    xs xdc.tools.configuro -c $(CGT00LS) -t $(XDCTARGET) $<
%.obj : %.c $(CONFIG)/compiler.opt
    $(CC) $(shell cat $(CONFIG)/compiler.opt) -c $<
hello.exe : hello.obj $(CONFIG)/linker.cmd
    $(LNK) -out:@ hello.obj @$ (CONFIG)/linker.cmd $(RTS)
```

No platform specification is needed because each target in microsoft.targets uses its own default platform.

2.4.5.3 Integrating Configuration with GNU gcc Tools in a Makefile

The following example makefile runs configuro and uses the compiler.opt and linker.cmd files generated by configuro in the command lines for the GCC compiler and linker.

```
CGT00LS = $(TOOLS)/vendors/gnu/gcc/3.2.1/Linux
CC = $(CGT00LS)/bin/gcc
LNK = $(CGT00LS)/bin/gcc
RTS = -lstdc++

CONFIG = mycfg
XDCTARGET = gnu.targets.Linux86

%/linker.cmd %/compiler.opt : %.cfg
    xs xdc.tools.configuro -c $(CGT00LS) -t $(XDCTARGET) $<
%.o : %.c # forget built-in rule
%.o : %.c $(CONFIG)/compiler.opt
    $(CC) $(shell cat $(CONFIG)/compiler.opt) -c $<
hello : hello.o $(CONFIG)/linker.cmd
    $(LNK) -o @$ (CONFIG)/linker.cmd $(RTS)
```

No platform specification is needed because each target in gnu.targets uses its own default platform.

The GCC compiler does not directly support an option to include other command line options from a file. However, this example accomplishes this by using GNU make's "\$(shell ...)" function to read in the options before calling GCC.

Learning More

This chapter describes where to get additional information about the XDC tools used to both consume and produce packages.

Topic	Page
3.1 Using the API Reference Help System (CDOC)	13
3.2 List of XDC Tools	14
3.3 Other Documents.....	14
3.4 The Package Path	15
3.5 Terminology	16

3.1 Using the API Reference Help System (CDOC)

To open the API Reference online help system, run the following command from the command prompt—for example by opening the Microsoft Windows Start menu and choosing Run.

```
xs xdc.tools.cdoc.sg
```

You see the "cdoch" window with a tree view of the XDC packages available to you. All of these packages provide documentation that can be displayed by the CDOC viewer.

Click "+" next to a repository to expand its list of packages. Click "+" next to a package name to see the list of modules it provides. You can further expand the tree to see a list of the functions provided by a module.

Double-click on a package or module to see its reference information.

Notice the icons in the upper-right corner of the window. The following icons are useful:

-  View C interface view of documentation.
-  View XDC interface view of documentation.
-  Close all page tabs.
-  Maximize tab contents.

For each topic you view, there is a tab across the top of the page area. You can use these to quickly return to other pages you have viewed. You can also use the arrows next to "Views" to move backward and forward in your history of page views.

To close a page and remove its tab, click the X on the tab.

3.2 List of XDC Tools

The following XDC tools are useful when creating applications that use XDC-based content. See the CDOC online documentation for information about each of these tools. (See Section 3.1 for information about using CDOC.)

Table 1. XDC Tools

xdc.tools.cdoc.sg	Online documentation viewer for XDC-based packages
xdc.tools.configuro	Processes an XDC configuration file to generate compiler.opt and linker.cmd files
xdc.tools.path	Displays a list of packages on the XDCPATH
xdc.tools.path.sg	Displays a list of packages on the XDCPATH in a graphical window
xdc.tools.repoman	Command-line based repository manager
xdc.tools.repoman.sg	Graphical display repository manager

3.3 Other Documents

You can read the following additional documents to learn more about XDC:

- *XDС Release Notes* (XDC_INSTALL_DIR/release_notes.html). Includes information about software version, upgrades and compatibility, host and target device support, validation, and known issues.
- *XDС Consumer User's Guide* (SPRUEX4). Provides more information about using XDC packages, XDC tools, and the XDC runtime APIs.
- "CDOC" reference documentation. Contains full reference information about all installed packages and their modules, APIs, XDC configuration, data structures, etc. (See Section 3.1)
- *XDС Producer User's Guide* (SPRUEX5--Not yet written.) Provides information about creating XDC packages.
- *XDС Documentation List* (XDC_INSTALL_DIR/doc/index.html). Provides links to several other XDC documents.
- *DSP/BIOS 5.30 Textual Configuration User's Guide* (SPRU007H) and *DSP/BIOS Tconf Language Coding Standards* (SPRAA67). Tconf was the precursor to XDC, and these documents may be useful in learning XDC configuration.

3.4 The Package Path

Client application code and XDC packages often need to reference files contained in other packages. Files contained in a package are always found along the “Package Path,” which is simply a semicolon-separated list of package “repositories”. A repository is a directory that contains installed packages.

The Package Path is always defined as follows:

```
XDCPATH;XDC_INSTALL_DIR/packages;^
```

In this definition, XDCPATH is a semicolon-separated set of repositories specified by the user. The XDC_INSTALL_DIR/packages is the repository of packages included with the XDC tools. The “^” denotes the current package’s repository (if there is a current package). For most tools, the current package is defined as the package named by a package.xdc file in the current working directory. If no such file exists, then any repository name containing the “^” character expands to an empty string “”.

All source files, whether they are part of a package or your application, should always reference files contained in a package by prefixing the file name with the package’s directory name. For example, to include the Task.h header in the ti.bios package, your source files should #include <ti/bios/Task.h>.

When compiling your sources, you should add a -I option for each repository named in the Package Path in the same order that they appear in the Package Path. Following this pattern allows you to easily add/remove/override packages without having to update your build system. You can simply install (or remove) a package into one of the repositories, add (or remove) the reference from your sources, and rebuild.

Since virtually all build tools support an option similar to -I, the pattern works for linker command files, assembly language files, XDC scripts, and makefiles.

3.4.1 Managing the Package Path

In the Package Path, you control the XDCPATH portion. This prefix is a string of ‘;’ separated directories that contain packages.

XDCPATH may be specified as an environment variable or on the command line of any tool that uses the Package Path. See the documentation of the particular command-line tool to find the appropriate option.

IMPORTANT NOTE: All directories referenced in XDCPATH, whether in an environment variable or a command line, must use **forward slashes** (/) as separators instead of the Windows-standard backslash (\). For example, use "c:/os;c:/drivers;c:/framework".

Multiple versions of the same package can appear along the Package Path. The Package Path can name multiple repositories that can contain a package directory with the same name. When searching for a package, the first repository that contains a directory matching the package’s name is used. Thus, even if two packages with the same name appear in the package path, only one will ever be found—the first one in the order specified in the package path.

Thus, it is possible to quickly update selected packages by adding an “updates” repository at the front of XDCPATH. If the updates are successful, you can “permanently” remove the old versions; otherwise you can quickly revert to the previous versions by simply removing the updates repository from XDCPATH.

3.5 Terminology

- **Client applications.** The application that consumes packages and calls functions in modules to perform application actions.
- **Interfaces.** These are effectively "abstract modules"—they are a specification without an implementation. They encapsulate a set of related constants, types, functions (target domain content), and configuration parameters (meta domain content). Modules and interfaces can inherit interfaces. Interface names often begin with an uppercase "I".
- **Meta content.** Host-based content that plays an active role in the design-time build and configuration as well as the run-time analysis of target programs.
- **Modules.** An implementation of an interface. These encapsulate a related set of types and functions. Roughly speaking, a module is comparable to a C++ or Java class; all of the identifiers defined by module are "scoped" by the module's name and it optionally defines and manages a set of instance objects. Module names start with a capital letter and use camel case. For example, `ti.sysbios.knl.Task`.

XDC modules have a dual existence: each module exists at design-time before an application is linked (where its configuration parameters can be set and its properties examined) and at runtime (where its functions are called by the application).
- **Packages.** These serve as named containers for modules and interfaces as well as other software artifacts. Packages are the focal point for managing content throughout its life-cycle. All packages are built, tested, released, and versioned as a unit. Packages are used for delivery from a producer to a consumer.
- **Package Path.** A sequence of repositories that are searched to locate packages.
- **Platform.** A platform encapsulates all information required to build and run an executable—including a memory map and peripherals. Platforms are named instances of a module provided by a "platform" package that describes a specific hardware board or software simulator. Platform packages provide a default instance and the name of this default instance is simply the package name; other platform instances can be defined in the user's config.bld file. The XDC distribution includes several XDC platform packages, such as `ti.platforms.sim6xxx`.
- **Repositories.** A directory containing one or more packages. The dots in the name of a package, interface, or module refer to its location within the repository. For example, the `ti.sysbios.knl` package is located in the `ti/bios/knl` directory with respect to the repository directory.
- **Target.** A target encapsulates all information required to build a library—the ISA (Instruction Set Architecture), runtime model, location of the compiler tool-chain, and required compiler and linker options. Targets are implemented as metaonly modules; as a result, they are delivered in packages, are identified using their package and module name, and can be configured by a user. The XDC distribution contains a `ti.targets` package that supports many TI targets, a `gnu.targets` package that supports GCC, and a `microsoft.targets` package that supports the Microsoft C/C++ tool-chain.
- **Target content.** Software bound into an application program executing on a particular hardware platform.
- **XDC Tools.** A set of host-independent tools that enable the creation and use of packages, modules, and interfaces.