

Cell/B.E. SDK 3.0, Part 1: Create an SPU project

A quick-read intro to the new Multicore Accelerator SDK 3.0 IDE

Skill Level: Introductory

Sean Curry (seancurry.ut@gmail.com)
Software Engineer
IBM

13 Nov 2007

This introductory tutorial, designed for the IBM SDK for Multicore Acceleration, Version 3.0 (otherwise known as the Cell Broadband Engine SDK), explores the Cell/B.E. processor IDE and gives developers a click-for-click walk-through of building a simple project in this environment. This tutorial is broken into six quick-perform parts dealing with creating an SPU project, creating a PPU project, creating the Cell/B.E. simulator, configuring the application launcher, debugging and doing performance analysis, using simulator consoles, using the ALF wizard, and setting IDE preferences.

Section 1. Before you start

Learn what to expect from this tutorial, and how to get the most out of it.

This tutorial series

This series of six quick-and-easy tutorials introduces the IBM SDK for Multicore Acceleration 3.0 (the Cell/B.E. SDK 3.0) and explains how to create, build, and run POWER™ Processing Unit (PPU) and Synergistic Processor Unit (SPU)-managed make projects. During this process, you will learn how to use some of the main features of the Multicore Acceleration SDK.

The tutorial series contains the following tutorials:

- [Cell/B.E. SDK 3.0, Part 1: Create an SPU project](#)
- [Cell/B.E. SDK 3.0, Part 2: Create a PPU project](#)
- [Cell/B.E. SDK 3.0, Part 3: Create the Cell/B.E. simulator environment](#)
- [Cell/B.E. SDK 3.0, Part 4: Configure the application launcher](#)
- [Cell/B.E. SDK 3.0, Part 5: Debug and complete dynamic or static performance analysis](#)
- [Cell/B.E. SDK 3.0, Part 6: Use simulator consoles, use the ALF wizard, and set IDE preferences](#)

Objectives

In this tutorial series, you will:

- Get step-by-step instructions on how to create, build, and run PPU- and SPU- managed make projects.
- Understand how to use the Local Cell Simulator environment.
- Walk through ways of using the static and dynamic performance analysis tools.
- Discover how to configure and use the C/C++ Cell Target Application launcher to run and debug your Cell/B.E. applications.
- See a great example usage scenario of the Accelerated Library Framework IDE wizard (ALF).

Prerequisites

Although this is a fairly entry-level tutorial, it is written for readers with some experience installing and using IDEs, especially makefiles. Some past experimentation with the Cell/B.E. SDK version 2.1 is a plus, but not required. This tutorial assumes only minimal familiarity with the Eclipse IDE.

Changes since SDK 2.1

The version 3.0 release of the SDK contains a number of significant enhancements over previous versions of the SDK and *completely* replaces earlier SDK versions. Enhancements include:

- New installation process based on YUM.
- Addition of PPU and SPU Fortran compiler.
- Addition of PPU-only GNU Ada compiler.
- Minor enhancements to XL C/C++ compiler.
- Addition of single-source XL C/C++ compiler.
- Compilers now generating code that is compliant with the [SPE Stall App. Note](#).
- GCC toolchain enhancements, including:
 - GCC C/C++ compilers supporting infix operations on vector data types.
 - GCC support of additional PPU VMX intrinsics.
 - GCC performance enhancements.
 - Link time estimation of SPU stack consumption.
 - Transparent SPE embedding.
 - SPE function descriptor support for embedded executables.
 - Additional POSIX API support in the SPE runtime library.
 - Addition of SPE direct access of PPE address space using `__ea` qualified data types. This feature is supported by the GCC C++ compiler only.
 - Combined debugger enhancements.
- Restructuring of examples and demonstration source code, and more examples.
- Addition of DaCS and DaCS for Hybrid-x86 programming model.
- Major enhancements to ALF framework and addition of ALF for Hybrid-x86.
- Complete implementation of SIMDMath library.
- Addition of BLAS Linear Algebra library.
- Addition of FFT library.
- Addition of SPU virtual clock and timer services.
- Addition of Performance and Debug Tracing tool (PDT and PDTR).
- Updates to Cell Performance Counter, OProfile, and FDPR-Pro

performance tools.

- Addition of Hybrid performance tooling.
- Performance enhancements to the Full System Simulator.
- Updated Full System Simulator sysroot to Fedora 7.

You can also use these related products with the SDK 3.0 components for additional capability:

- [XL C/C++ compilers](#)
- [Fortran dual-source compilers](#)
- [alphaWorks Visual Performance Analyzer \(VPA\)](#)

System requirements

The SDK 3.0 has specific hardware and software requirements.

Hardware

The following table shows the recommended minimum configuration for each hardware platform.

Hardware requirements

System	Recommended minimum configuration
x86 or x86-64	2GHz Pentium® 4 processor
PowerPC®	64-bit PPC with a clock speed of 1.42 GHz 32-bit PPC platforms not supported
BladeCenter® QS20	Revision 31 or greater and minimum firmware level of QA-06.14.0-0F (7.21)
BladeCenter QS21	Minimum hardware firmware level of QB-01.08.0-00

Additionally, all systems must have:

- Hard disk space: 5GB (minimum) to install the source package and the accompanying development tools
- 1GB RAM (minimum) on the host system

Note: If you use the Full System Simulator, the minimum amount of RAM installed

must be twice the amount of simulated memory. For example, to simulate a system with 512MB of RAM, the host system must have at least 1GB of RAM installed.

Software

This tutorial requires Fedora 7, which must be installed before you install the SDK.

SELinux: The SELinux policy files that are included in the Fedora 7 base distribution prevent `spufs` from loading correctly on boot. To install the SDK, you must either turn off SELinux or update the `selinux-policy` and `selinux-policy-targeted` RPMs to the latest version. The preferred method is to update the RPMs. To update, type the following commands as root: `yum update selinux-policy selinux-policy-targeted`.

expat: The DaCS for Hybrid-x86 daemon for both X86_64 and the BladeCenter QS20 and QS21 platforms requires the expat XML parsing library. Install expat by typing the following command as root: `yum install expat`.

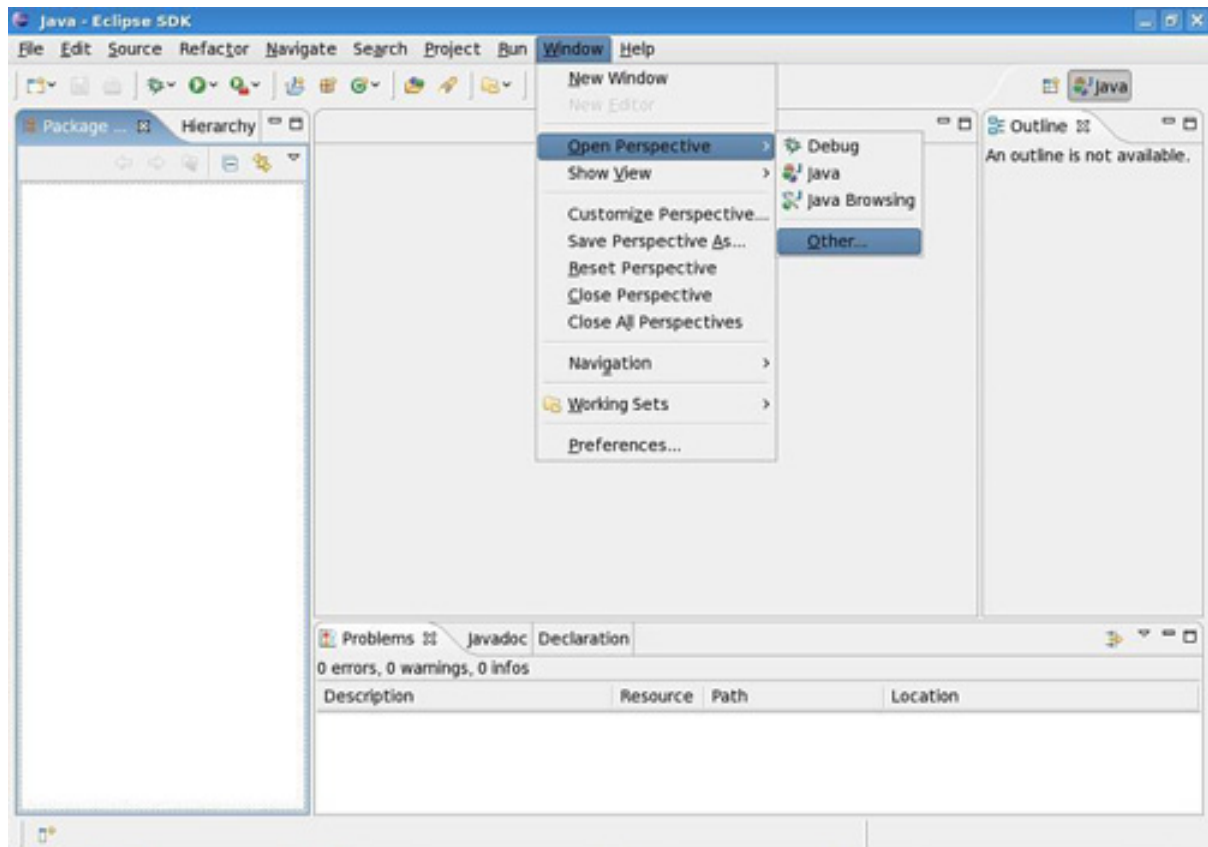
SDK utility software: The SDK requires the packages `rsync`, `sed`, `TCL`, and `wget`. To install these dependencies, type the following command as root: `yum install rsync sed tcl wget`.

Section 2. Getting started

Using Eclipse

Okay, so you've already got the CDT and the Cell/B.E. IDE installed into your Eclipse directory. Now open Eclipse and switch to the C/C++ perspective by clicking **Window > Open Perspective > Other**.

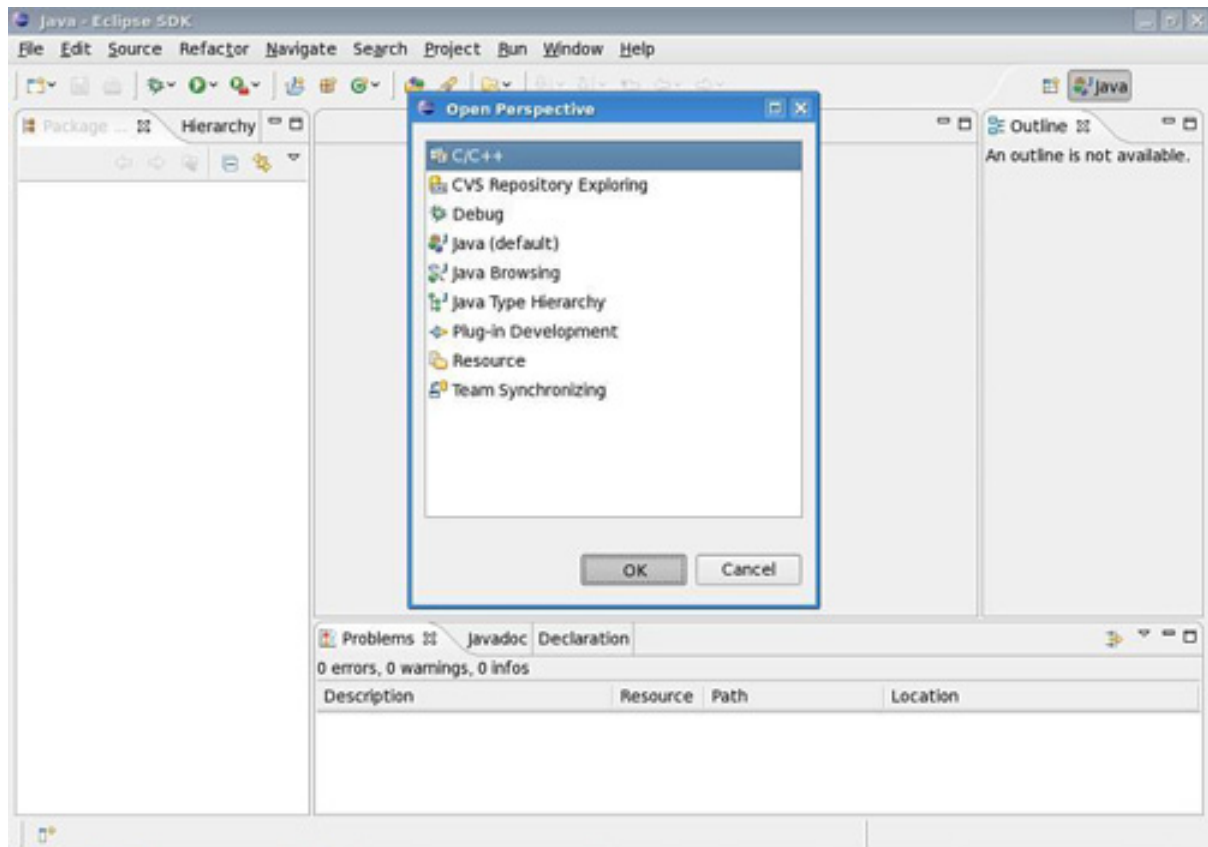
Figure 1. Getting started with Eclipse



Choose the C/C++ perspective

Choose C/C++ and click **OK**.

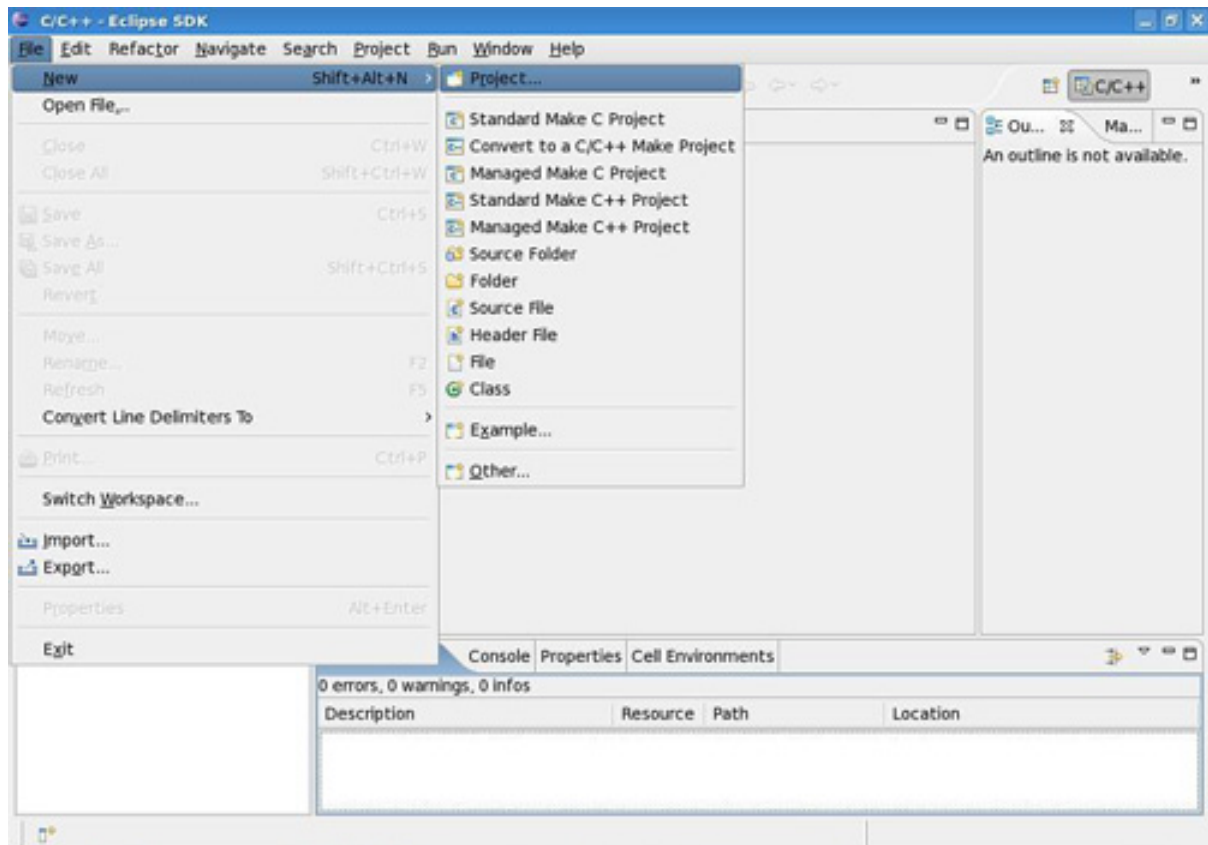
Figure 2. Getting the C/C++ perspective



Section 3. Creating an SPU project

Create a new C project by clicking **File > New > Project**.

Figure 3. New C project



Using the New Project wizard

You now see the New Project wizard. For this example, expand the section C, and select **Managed Make C Project**.

A Standard Make C/C++ project requires you to provide a makefile. A Managed Make project creates a makefile for you. So, if your project already has a makefile, or if you would like to create your own makefile, select a **Standard Make** project.

Click **Next**.

Naming the project

For Project name, type in `SPU`. Click **Next**.

Selecting the project type

Select the appropriate project type (such as Cell/B.E. PPU Executable, Cell/B.E. SPU Static Library, and so on) for the project you are creating. When you select a

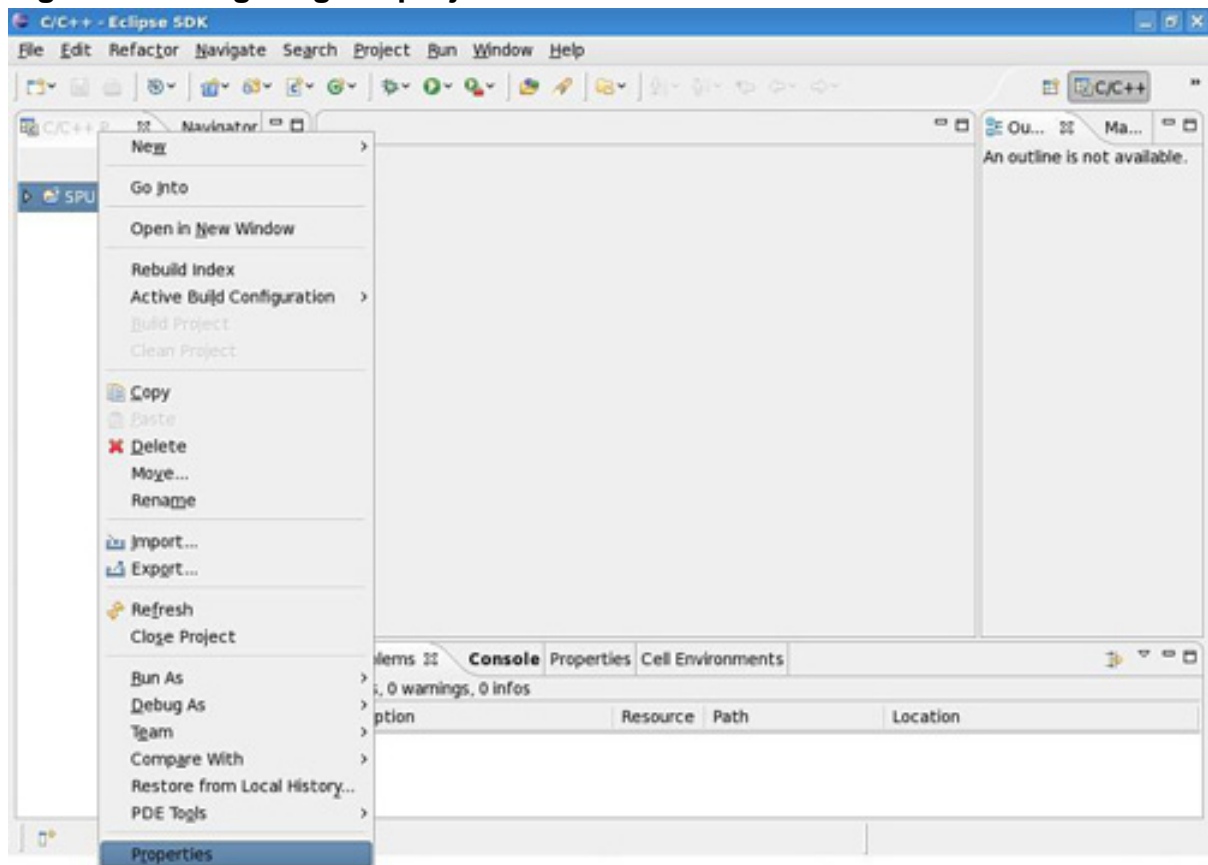
project type, you will see the default configurations available for that project type. Later in the tutorial, you will learn how to create new build configurations, as well as how to modify the existing built-in configuration settings.

For the example project type, select **Cell/B.E. SPU Executable**, and click **Finish**.

Section 4. Configuring the SPU project

In the C/C++ Projects view, right-click the SPU project, and select **Properties**.

Figure 4. Configuring the project

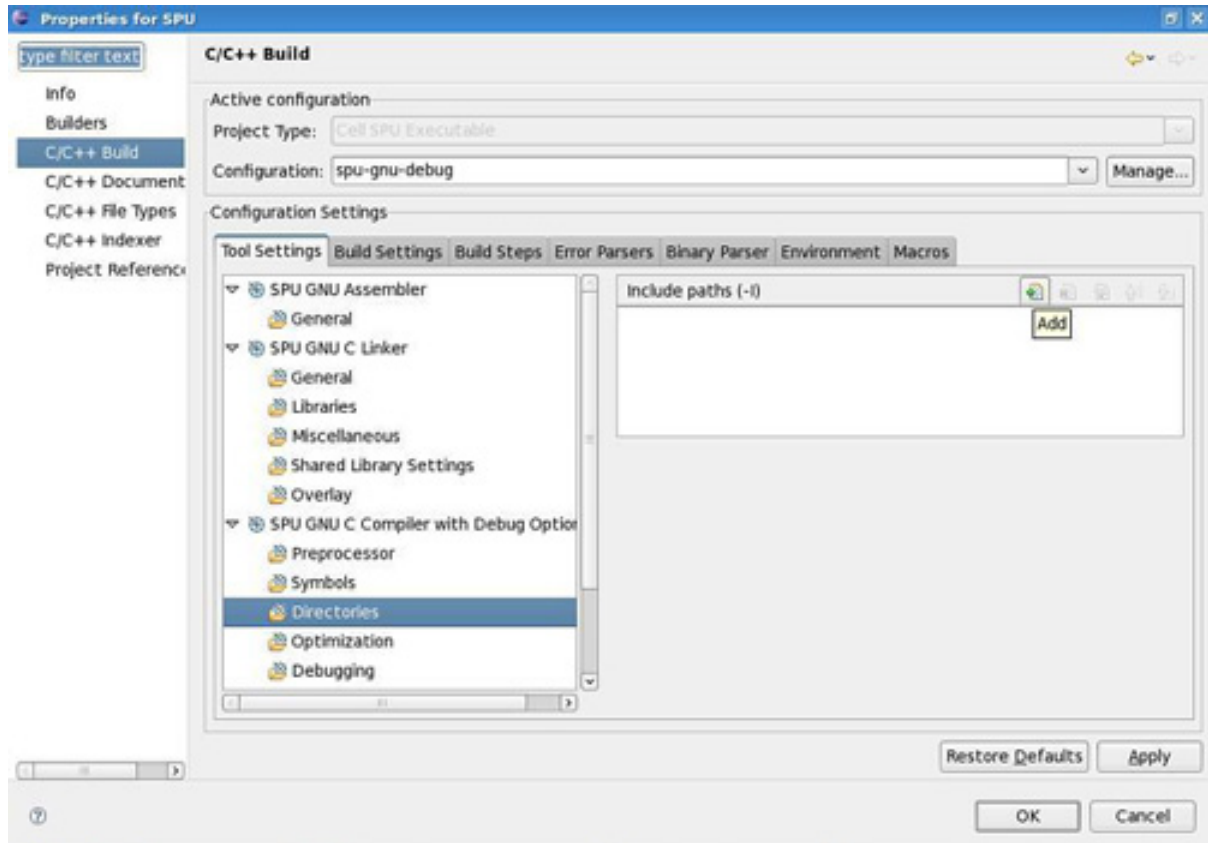


Adding SPU compiler directories

You need to add the directory that contains the profile.h header file to the list of the compiler's include paths so that you can use the dynamic performance-analysis tool.

Click **C/C++ Build** in the left pane. Under SPU GNU C Compiler with Debug Options, select **Directories**. In the Include Paths pane on the right, click the **Add** button.

Figure 5. Add compiler directories



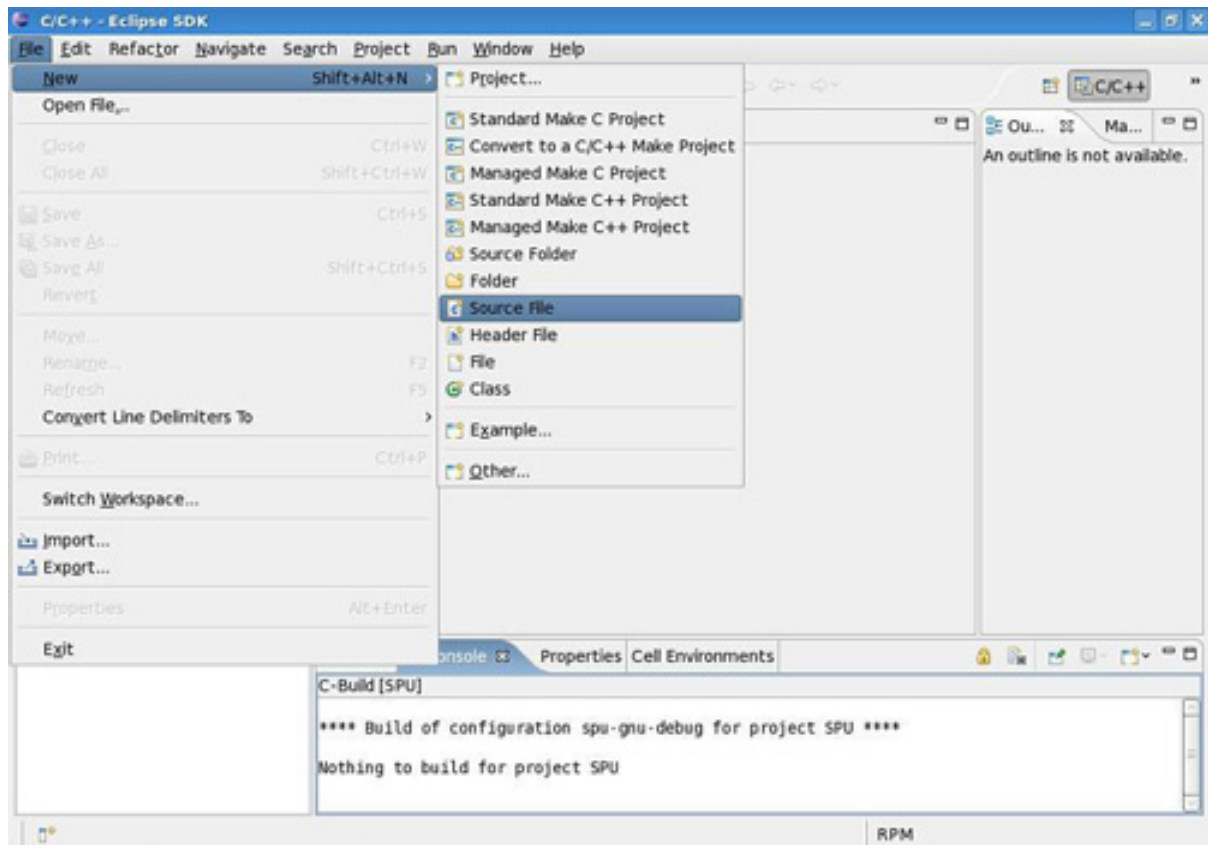
Adding the directory path

Type in `/opt/ibm/systemsim-cell/include/callthru/spu`. Return to the C/C++ Perspective by clicking **OK** twice.

Section 5. Creating a new source file

Create a C source file by clicking **File > New > Source File**.

Figure 6. Create a new source file



For Source File, type `spu.c`. Click **Finish**.

Editing the new source file

A new editor appears so you can enter your source code. Copy and paste the following source code into your editor (you can uncomment the commented lines later):

```
#include <stdio.h>
#include <profile.h>
int main(unsigned long long id)
{
    //prof_clear();
    //prof_start();
    printf("Hello Cell (0x%llx)\n", id);
    //prof_stop();

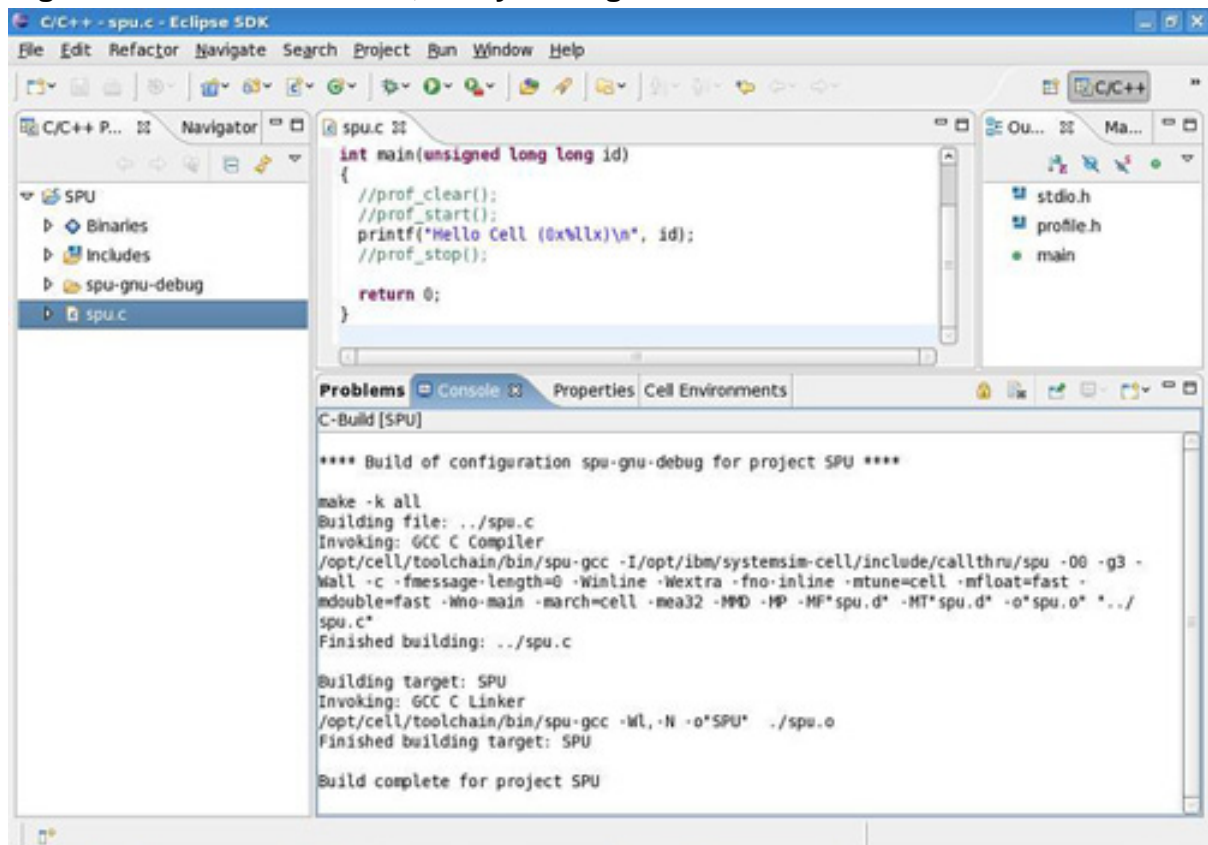
    return 0;
}
```

Section 6. Building the project

Save the source file (**Ctrl + S**). The project is built automatically.

The build output is displayed in the Console view. New resources, such as binaries and includes, are shown in the C/C++ Projects view.

Figure 7. Push the buttons, and you've got an automatic build



You're finished. You've built an SPU project.

Section 7. Going to other parts of this tutorial series

- [Cell/B.E. SDK 3.0, Part 2: Create a PPU project](#)
- [Cell/B.E. SDK 3.0, Part 3: Create the Cell/B.E. simulator environment](#)

- [Cell/B.E. SDK 3.0, Part 4: Configure the application launcher](#)
- [Cell/B.E. SDK 3.0, Part 5: Debug and complete dynamic or static performance analysis](#)
- [Cell/B.E. SDK 3.0, Part 6: Use simulator consoles, use the ALF wizard, and set IDE preferences](#)

Resources

Learn

- Use an [RSS feed](#) to request notification for the upcoming articles in this series. (Find out more about [RSS feeds of developerWorks content](#).)
- Check out the original installation document, "[Installation Guide for the SDK for Multicore Acceleration v3.0](#)" if you get stuck with a point in this series of installation tutorials.
- See "[Introduction to the Cell Multiprocessor](#)" (*IBM Journal of Research and Development*, 2005) for an introductory overview of the Cell/B.E. multiprocessor's history, the program objectives and challenges, the design concept, the architecture and programming models, and the implementation.
- To learn more on Cell/B.E. programming, try the developerWorks series:
 - "[Programming high-performance applications on the Cell/B.E. processor](#)"
 - "[PS3 fab-to-lab](#)"
 - "[The little broadband engine that could](#)"
- Refer to the [Cell Broadband Engine documentation](#) section of the IBM Semiconductor Solutions Technical Library for a wealth of downloadable manuals, specifications, and more.
- Refer to the [Cell Broadband Engine documentation](#) section of the IBM Semiconductor Solutions Technical Library for a wealth of downloadable manuals, specifications, and more.
- Sign up for the [developerWorks newsletter](#) and get the latest developer news and Cell/B.E. happenings delivered to your inbox each week. Check *Power Architecture* when you sign up to receive Cell/B.E. news in your newsletter.

Get products and technologies

- Go to the [developerWorks Cell Broadband Engine Resource Center](#) for Cell/B.E.-related resources, [SDK libraries](#), [SDK and third-party-product downloads](#), and news.
- Find [Cell/B.E. SDK version 2.1](#) and its tutorial "[An introduction to the IDE for the Cell Broadband Engine SDK](#)" (developerWorks, March 2007).
- Access the [IBM XLC compiler](#) for porting efforts; it is optimized for the Cell/B.E. processor.
- Find all Cell/B.E.-related articles, discussion forums, downloads, and more at the IBM developerWorks [Cell Broadband Engine resource center](#): your definitive resource for all things Cell/B.E.

- Contact IBM about [custom Cell/B.E.-based or custom-processor based solutions](#).

Discuss

- [Participate in the discussion forum for this content](#).
- Check out the [Cell Broadband Engine Architecture forum](#) to get your technical questions about the processor answered. Juicy problems and answers from the forums are rounded up periodically and highlighted in the ["Forum watch" blog series](#).
- Go to the [Power Architecture blog](#) for news, downloads, instructional resources, and event notifications for Cell/B.E. and other Power Architecture-related technologies. You can find the popular "Forum watch" blog series (Q&A roundup) and the "FixIt" technology updates.

About the author

Sean Curry

Sean is an undergraduate student at The University of Texas at Austin, working toward a degree in Computer Sciences. As an intern at the IBM Linux Technology Center, Sean works as part of the Linux on Cell/B.E. team, specifically contributing to the IDE for Cell Broadband Engine SDK project. Sean develops source code and conducts integration testing and product release management.

Trademarks

IBM, BladeCenter, and POWER are trademarks of IBM Corporation in the United States, other countries, or both.

Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.