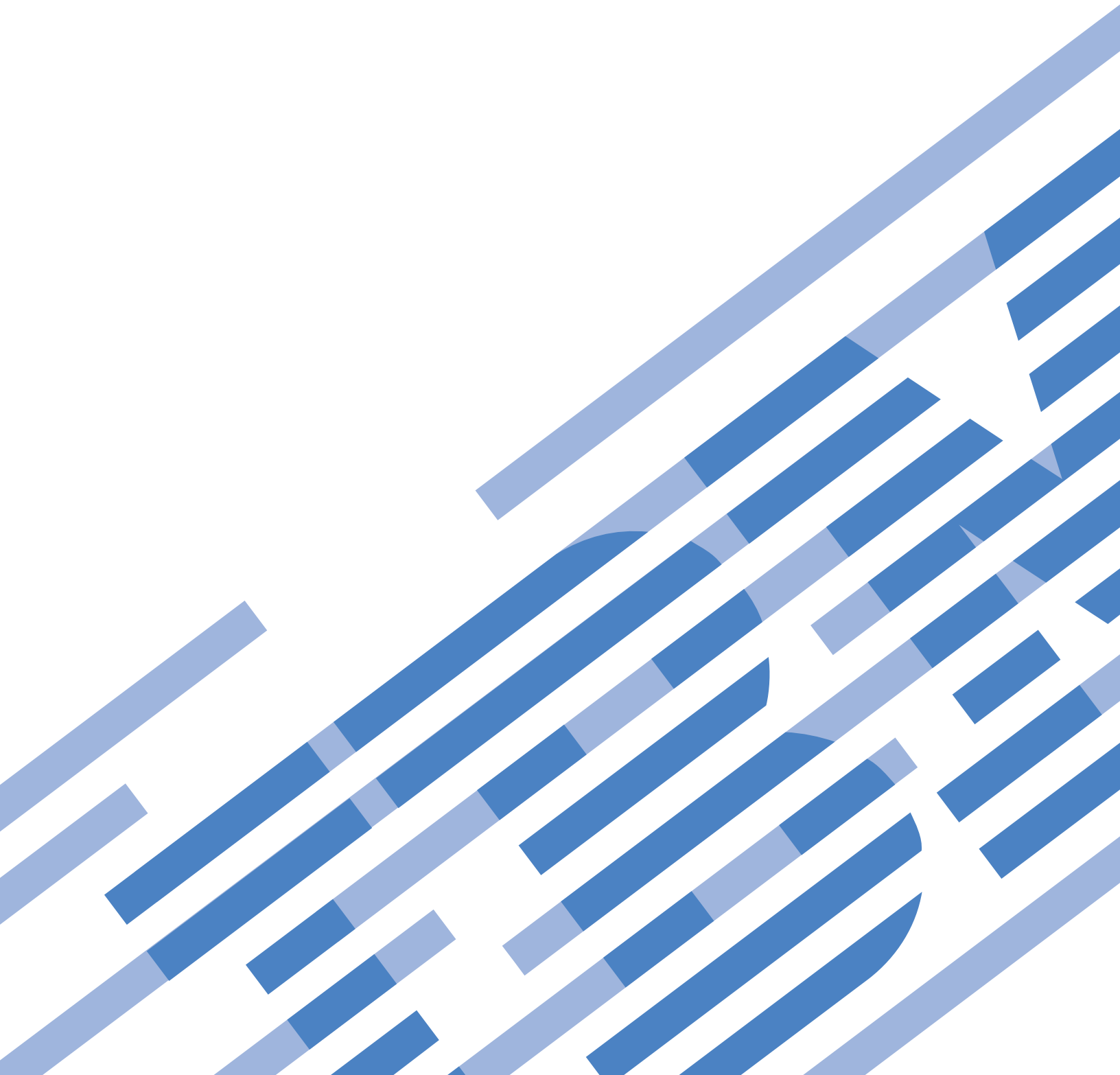




Linux

Developing software using the IBM Software  
Development Kit for Linux on Power







Linux

Developing software using the IBM Software  
Development Kit for Linux on Power

**Note**

Before using this information and the product it supports, read the information in “Notices” on page 97.

**Ninth Edition (April 2016)**

© Copyright IBM Corporation 2012, 2016.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## Developing software using the IBM Software Development Kit for Linux on Power . . . . . v

### 1 Introducing IBM Software Development Kit for Linux on Power . . . . . 1

- 1.1 IBM SDK for Linux on Power requirements . . . . . 2
- 1.2 IBM SDK for Linux on Power supported Linux distributions . . . . . 4
- 1.3 IBM SDK for Linux on Power user interface . . . . . 5

### 2 Setting up the ppc64 or ppc64le version of the IBM SDK for Linux on Power . . . . . 7

- 2.1 Downloading and installing the SDK . . . . . 7
  - 2.1.1 Using the installation script to download and install the IBM Software Development Kit for Linux on Power . . . . . 7
  - 2.1.2 Automatically downloading and installing using the IBM Linux on Power Tools Repository . . . . . 7
    - 2.1.2.1 Downloading and installing the IBM Linux on Power Tools Repository initialization package . . . . . 7
    - 2.1.2.2 Installing the IBM SDK for Linux on Power RPMs using the IBM Linux on Power Tools Repository . . . . . 8
  - 2.1.3 Automatically downloading and installing the IBM SDK for Linux on Power on Ubuntu . . . . . 8
- 2.2 Recommended and optional packages. . . . . 9
  - 2.2.1 Downloading and installing the IBM XL C/C++ Community Edition Compiler . . . . . 9
    - 2.2.1.1 Manually installing the IBM XL C/C++ Compiler . . . . . 9
    - 2.2.1.2 Automatically installing with the Remote Setup Wizard plugin . . . . . 10
  - 2.2.2 Installing the Advance Toolchain for Linux on Power . . . . . 11
- 2.3 Starting the IBM SDK for Linux on Power . . . . . 11
- 2.4 Creating a project . . . . . 12
  - 2.4.1 Creating a C/C++ project directly on a Power Systems server using Advance Toolchain . . . . . 12
  - 2.4.2 Creating a local C/C++ project direct on a Power Systems server using the IBM XL C/C++ Community Edition Compiler . . . . . 13
  - 2.4.3 Importing an existing Makefile project . . . . . 14
  - 2.4.4 Importing an existing Autotools project . . . . . 15
    - 2.4.4.1 Configuring the imported project. . . . . 16
    - 2.4.4.2 Building the imported project . . . . . 16
  - 2.4.5 Creating an empty Autotools project . . . . . 16
  - 2.4.6 Creating a “Hello World” Autotools project . . . . . 17
  - 2.4.7 Changing the Advance Toolchain version. . . . . 17
  - 2.4.8 Running an executable program in a project . . . . . 18

- 2.4.9 Debugging a project with Advance Toolchain for Linux on Power or IBM XL C/C++ Community Edition Compiler . . . . . 18

### 3 Setting up the x86\_64/amd64 version of the IBM SDK for Linux on Power . . . . . 21

- 3.1 Downloading and installing the x86\_64/amd64 client . . . . . 21
- 3.2 Setting up remote Power Systems server . . . . . 22
- 3.3 Creating and using synchronized projects on x86\_64/amd64 clients . . . . . 22
  - 3.3.1 Creating a synchronized C/C++ project . . . . . 23
  - 3.3.2 Creating a synchronized project by importing an existing Makefile project . . . . . 23
  - 3.3.3 Creating a synchronized project by importing an existing Autotools project . . . . . 24
    - 3.3.3.1 Configuring the imported project. . . . . 25
    - 3.3.3.2 Building the imported project . . . . . 25
  - 3.3.4 Creating a synchronized empty Autotools project . . . . . 26
  - 3.3.5 Creating a synchronized “Hello World” Autotools project . . . . . 26
  - 3.3.6 Running an executable program in a synchronized project . . . . . 27
  - 3.3.7 Debugging a synchronized project . . . . . 27
- 3.4 Creating and using cross-compiled projects . . . . . 28
  - 3.4.1 Creating a cross-compiled C/C++ project. . . . . 28
  - 3.4.2 Executing a cross-compiled project . . . . . 29
  - 3.4.3 Debugging a cross-compiled project . . . . . 29
- 3.5 Installing and using QEMU user-mode emulation . . . . . 30
  - 3.5.1 Running a cross-compiled application with QEMU user-mode emulation . . . . . 31
- 3.6 Installing and executing the IBM POWER Functional Simulator . . . . . 31
  - 3.6.1 Installing and setting up the POWER Functional Simulator . . . . . 31
  - 3.6.2 Installing the IBM POWER Functional Simulator . . . . . 32
- 3.7 Running the IBM POWER Functional Simulator . . . . . 32
  - 3.7.1 Running the IBM POWER Functional Simulator from within the SDK. . . . . 32
    - 3.7.1.1 Running the IBM POWER Functional Simulator from within the command line . . . . . 33
- 3.8 Simulator networking. . . . . 33

### 4 Managing projects . . . . . 35

- 4.1 Setting flags . . . . . 35
  - 4.1.1 Recommended debug, compiler, and linker settings for Power processor tuning . . . . . 35
  - 4.1.2 Setting debug flags for Autotools-based projects. . . . . 40
    - 4.1.3 Setting optimization level flags for Autotools-based projects . . . . . 41
- 4.2 Editing a project . . . . . 42

4.2.1	Setting the Linux tools path . . . . .	42
4.2.2	Using the coding assistant. . . . .	42
4.3	Building a project . . . . .	42
4.3.1	Building a project . . . . .	42
4.3.2	Building a project with clean build. . . . .	42
4.3.3	Adding and using a Make target . . . . .	43

## 5 Migrating an application to Power Systems servers using Migration

<b>Advisor . . . . .</b>	<b>45</b>	
5.1	Running Migration Advisor. . . . .	45
5.2	Using the Migration Wizard to configure the Migration Advisor . . . . .	45
5.2.1	Basic Migration Advisor configuration . . . . .	46
5.2.2	Advanced Migration Advisor configuration . . . . .	46
5.3	Migration Advisor checkers. . . . .	47
5.3.1	Linux/x86 to Linux/Power application migration checker . . . . .	51
5.3.2	Linux/Power application optimization checker . . . . .	52
5.3.3	Linux/Power 32bit to Linux/Power 64bit application migration checkers . . . . .	52
5.3.4	Customizing the performance degradation checker. . . . .	54
5.3.5	Customizing the syscall not available for Linux on Power checker . . . . .	55
5.3.6	Customizing the Linux/x86-specific API checker. . . . .	55
5.4	Configuring Migration Advisor checkers manually . . . . .	56
5.5	Using Migration Advisor quick fixes. . . . .	56

## 6 Analyzing application performance on Power Systems servers . . . . . 59

6.1	Analyzing performance with the CPI breakdown plug-in . . . . .	59
6.1.1	CPI analysis overview . . . . .	59
6.1.2	CPI events and metrics for POWER7 . . . . .	59
6.1.3	CPI events and metrics for POWER8 . . . . .	62
6.1.4	Profiling a binary application with CPI from the command line . . . . .	68
6.1.5	Profiling a project with the CPI breakdown plug-in . . . . .	68
6.2	Analyzing performance with OProfile . . . . .	69
6.2.1	Profiling a project with OProfile. . . . .	69
6.2.2	Profiling a synchronized project with OProfile . . . . .	70
6.3	Analyzing performance with Perf . . . . .	71
6.3.1	Profiling a project with Perf . . . . .	71
6.3.2	Profiling a synchronized project with Perf . . . . .	72

6.4	Profiling a project with gprof . . . . .	72
6.5	Analyzing application behavior using Valgrind . . . . .	73
6.5.1	Profiling a project using Valgrind . . . . .	73
6.5.2	Profiling a synchronized project using Valgrind . . . . .	74
6.5.3	Applying quick fixes for Valgrind-reported errors . . . . .	75
6.6	Analyzing POSIX Threads using Trace Analyzer . . . . .	75
6.6.1	Trace Analyzer overview . . . . .	75
6.6.2	Profiling a project with Trace Analyzer . . . . .	75
6.6.3	Collecting a trace by setting variables from the command line . . . . .	76
6.6.4	Record Details view . . . . .	77
6.7	Analyzing coverage with gcov . . . . .	78
6.8	Analyzing performance with IBM Power System Performance Advisor . . . . .	79
6.9	Analyzing projects built with the Build Advisor . . . . .	80
6.9.1	Enabling the Build Advisor . . . . .	80
6.9.2	Enabling extra advice from Build Advisor . . . . .	80
6.9.3	Using the Build Advisor . . . . .	81

## 7 Analyzing performance with Source Code Advisor and FDPR . . . . . 83

7.1	Source Code Advisor and FDPR overview . . . . .	83
7.2	Running the Source Code Advisor on a project . . . . .	84
7.3	Viewing Source Code Advisor on an existing journal . . . . .	85
7.4	Reviewing the Source Code Advisor results . . . . .	85
7.5	Source Code Advisor events . . . . .	85
7.6	Optimizing executable code using FDPR . . . . .	86
7.6.1	Specifying FDPR optimization for a project . . . . .	87

## 8 Automatic updates. . . . . 89

## 9 Support for IBM SDK for Linux on Power . . . . . 91

9.1	Getting customer support . . . . .	91
9.2	Using integrated bug reporting . . . . .	91
9.2.1	Creating a report. . . . .	91
9.2.2	Submitting a report to the Linux on Power Community . . . . .	92
9.3	Setting up SSH credentials . . . . .	93
9.3.1	Logging in from workstation to remote server with DSA key . . . . .	93
9.3.2	Logging in with DSA key but no password . . . . .	93

## Notices . . . . . 97

Trademarks . . . . .	98
----------------------	----

---

## Developing software using the IBM Software Development Kit for Linux on Power

The IBM® Software Development Kit for Linux on Power® is a free, Eclipse-based Integrated Development Environment (IDE). The SDK integrates C/C++ source development with the IBM Advance Toolchain for Linux on Power and classic Linux debugging and performance analysis tools.

The official version of this user guide is located in the Linux Knowledge Center: Developing software using the IBM Software Development Kit for Linux on Power (<http://www.ibm.com/support/knowledgecenter/linuxonibm/liaal/iplsdkmain.htm>).

**Note:** By using the code examples, you agree to the terms of the 10, “Code license and disclaimer information,” on page 95.





---

# 1 Introducing IBM Software Development Kit for Linux on Power

The IBM SDK for Linux on Power provides you with an all-in-one solution for developing software on Linux on Power servers. It integrates the Eclipse integrated development environment (IDE) with the Advance Toolchain and open source tools such as OProfile, Valgrind, and Autotools. In addition, it integrates the Feedback Directed Program Restructuring (FDPR<sup>®</sup>) and Pthread monitoring tool, which are specifically designed to analyze and exploit Power Systems<sup>™</sup> servers and includes powerful porting and analytic tools, such as Migration Advisor, Source Code Advisor and CPI Breakdown. The SDK also includes IBM SDK Java<sup>™</sup> Technology Edition, Version 8.0-3.11 and IBM Runtime Environment Java Technology Edition, Version 8.0-3.11

The IBM Software Development Kit for Linux on Power is available for x86\_64/amd64, ppc64 and ppc64le architectures, providing you with two different development modes:

- Locally on x86\_64/amd64, ppc64 and ppc64le
- Remotely using the x86\_64/amd64 version to connect to a remote Power System server

Using the x86\_64/amd64 version of the SDK, you can:

- Create, debug and profile remote projects using the Advance Toolchain
- Migrate applications using Migration Advisor
- Cross-compile applications using Advance Toolchain cross-compiler
- Take advantage of QEMU user-mode or IBM Power 8 and Power 9 Functional Simulator for development if you don't have a real Power machine

With the ppc64 and ppc64le versions, you can:

- Create applications using the Advance Toolchain and IBM XL C/C++ Community Edition Compiler
- Run, debug, and profile applications
- Migrate applications using Migration Advisor

The SDK includes manuals in the form of help plug-ins for many tools that are included with or integrated with IBM SDK for Linux on Power. To display these manuals, while in the user interface, click Help > Help Contents



**Note:** The IBM Software Development Kit for Linux on Power is provided as is only. Customers are not entitled to IBM Software Support. However, you can get help from the Linux on Power Community. For information, see 9.1, “Getting customer support,” on page 91.

## Related information:

- [IBM Software Development Kit for Linux on Power website](#)
- [IBM Advance Toolchain for Linux on Power documentation](#)

---

## 1.1 IBM SDK for Linux on Power requirements

The IBM Software Development Kit for Linux on Power has software prerequisites. Most of the prerequisites can be automatically installed by the package manager of the Linux distribution.

### Packages required by IBM SDK for Linux on Power

The IBM SDK for Linux on Power includes some required dependent packages to be installed on the Power Systems server while manually installing it. These packages include:

**fdprpro-version**

FDPR post-link optimization tool.

**pthread-mon-version**

Pthread monitoring tool.

**fdpr\_wrap-version**

Wrapper scripts for running FDPR under IBM SDK for Linux on Power.

These required packages are included in the IBM SDK for Linux on Power ISO image. If you configure the SDK's apt repository or the IBM Linux on Power Tools Repository, all the required packages are installed automatically.

**Note:** The Advance Toolchain is not available in the SDK's apt repository.

### Other required packages from the distribution

Depending on the packages you installed with your Linux distribution, you might be required to install additional packages from your distribution provider. If you install the SDK automatically using **yum**, **dnf**, **zypper**, or **apt-get**, these packages might be installed automatically. If you are installing using **rpm** or **dpkg**, you must be aware of these dependencies and install them manually from the distribution.

These packages include:

*Table 1. Other packages required from the distribution.*

Package name	Package description	Required for ppc64 and ppc64le	Required for x86_64/amd64
Autoconf	Produces shell scripts to automatically configure software source code packages.	Yes	Yes
Automake	Automatically generates Makefile.in files that are compliant with the GNU coding standards.	Yes	Yes
gettext	Internationalization and localization system for writing multilingual programs.	Yes	Yes
Git	Distributed revision control system. Required on server for synchronized projects.	Yes	No
GLIBC 2.3 or later	GNU C library.	Yes	Yes
GTK2	Used to create graphical user interfaces.	Yes	Yes

Table 1. Other packages required from the distribution (continued).

Package name	Package description	Required for ppc64 and ppc64le	Required for x86_64/amd64
Libtool	GNU Libtool, script for generic shared library support.	Yes	Yes
Perf	Performance monitoring tool for Linux Kernel. Perf is required if you plan to use Perf with IBM SDK for Linux on Power. In addition, Perf is required for using the CPI Breakdown plug-in.	Yes, for using Perf or CPI breakdown plug-in	No
Python	Support for Python programming language.	Yes	Yes

## Additional IBM tools installed with IBM SDK for Linux on Power

Table 2. Description of additional tools installed with IBM SDK for Linux on Power

Package name	Package Description	Required for ppc64 and ppc64le	Required for x86_64/amd64
advance-toolchain-atversion-runtime	Self contained toolchain that provides preview toolchain functionality in GCC, binutils, GLIBC, GDB, Valgrind, and OProfile	Yes	No
advance-toolchain-atversion-devel	Packages necessary to build applications that use the features provided by the Advance Toolchain	Yes	No
advance-toolchain-atversion-perf	Performance library installation targets for Valgrind and OProfile	Yes	No
advance-toolchain-atversion-mcore-libs	Libraries to build multi-threaded applications using the specialized multi-threaded libraries Animo-CBB, URCU, and Intel TBB	Yes	No

Table 2. Description of additional tools installed with IBM SDK for Linux on Power (continued)

Package name	Package Description	Required for ppc64 and ppc64le	Required for x86_64/amd64
advance-toolchain-atversion-cross-common advance-toolchain-atversion-cross-arch advance-toolchain-atversion-cross-arch-mcore-libs advance-toolchain-atversion-cross-arch-runtime-extras	Advance Toolchain cross-compiler to generate Power-compatible binaries from i686 development environments.	No	Yes
Where atversion is the version of the Advance Toolchain and arch is the architecture, which can be either ppc64 or ppc64le.			
fdprpro-version	IBM Feedback Directed Program Restructuring (FDPR), for post-link optimization	Yes	No
fpdr_wrap-version	Wrapper scripts for integrating FDPR with IBM SDK for Linux on Power	Yes	No
pthead-mon-version	Pthead monitoring tool	Yes	No

## Required authorities for IBM SDK for Linux on Power tasks

Many of the tasks for installing the SDK, such as installing packages, require you to be logged in to the system as root, with super user (administrator) authorities.

## 1.2 IBM SDK for Linux on Power supported Linux distributions

This information lists the Linux distributions that are supported by the SDK.

The SDK was tested to verify that it works with the Linux distributions listed in the table that follows. The usual compatibility with subsequent releases is assumed. The supported distributions vary according to the different development modes or packages.

Table 3. IBM SDK for Linux on Power supported Linux distributions

IBM SDK for Linux on Power version	Supported Linux distributions
ppc64	<ul style="list-style-type: none"> <li>Red Hat Enterprise Linux 7.2, and later</li> </ul>
ppc64le	<ul style="list-style-type: none"> <li>Red Hat Enterprise Linux 7.2, and later</li> <li>SUSE Linux Enterprise Server 12 SP1</li> <li>Ubuntu 14.04 LTS and 16.04 LTS</li> <li>Fedora 22</li> <li>CentOS 7</li> </ul>

Table 3. IBM SDK for Linux on Power supported Linux distributions (continued)

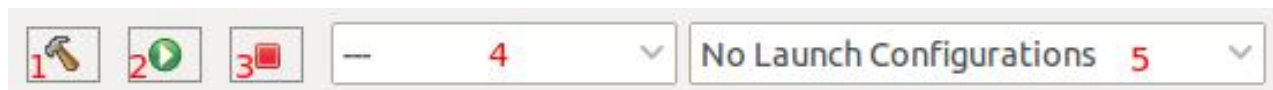
IBM SDK for Linux on Power version	Supported Linux distributions
x86_64/amd64	<ul style="list-style-type: none"><li>• Red Hat Enterprise Linux 7.2, and later</li><li>• SUSE Linux Enterprise Server 12 SP1</li><li>• Fedora 22</li><li>• Ubuntu 14.04 LTS and 16.04 LTS</li><li>• CentOS 7</li></ul>

**Note:** The IBM Software Development Kit for Linux on Power is provided as is only. Customers are not entitled to IBM Software Support. However, you can get help from the Linux on Power Community. For information, see 9.1, “Getting customer support,” on page 91.

### 1.3 IBM SDK for Linux on Power user interface

The SDK includes a launch bar that provides quick access to the plugin launches. It works for common tasks like running, debugging, and profiling an application.

This launch bar is composed for three buttons and three selectors.



1. Build the application before launching
2. Run or profile an application after it was built.
3. Stop the active action.
4. Allow selecting the launch mode: Run, Debug or Profile.
5. List all created launches.

The Launch Bar does not replace the default and known steps to complete these tasks; it aims to ease the task of launching configurations.



---

## 2 Setting up the ppc64 or ppc64le version of the IBM SDK for Linux on Power

You can download and install the packages for the SDK on Power Systems server.

---

### 2.1 Downloading and installing the SDK

These topics describe how to download and install the ppc64 and ppc64le versions of the SDK.

You can automatically download and install these using the IBM Linux on Power Tools Repository or using the installation script. In addition, you can manually download and install the IBM SDK for Linux on Power ISO image.

For any download method you choose, you must accept the license agreements and terms and conditions.

#### 2.1.1 Using the installation script to download and install the IBM Software Development Kit for Linux on Power

The simplest method for downloading and installing the packages for the SDK on Power Systems server is to download the `install-sdk.sh` script.

You can download the script at Support and Downloads after clicking **I agree**. This method installs the IBM Linux on Power Tools Repository, the Advance Toolchain, and the SDK.

#### 2.1.2 Automatically downloading and installing using the IBM Linux on Power Tools Repository

You can download and install the packages for the SDK on Power Systems server by using the IBM Linux on Power Tools Repository. This method configures yum, dnf, and zypper, allowing for automatic download and installation. Using the repository speeds the installation because all the required packages are installed in the correct order.

##### 2.1.2.1 Downloading and installing the IBM Linux on Power Tools Repository initialization package

The first step for automatic downloading and installation is to download and install the Linux on Power Software Repository initialization package.

#### About this task

To download and install the IBM Linux on Power Tools Repository initialization package, complete the following steps.

#### Procedure

1. Download the Linux on Power Software Repository initialization package to the server in either of the following ways:
  - Download the Linux on Power Software Repository initialization package from the IBM Software Development Kit for Linux on Power website at <http://www-304.ibm.com/webapp/set2/sas/f/lopdiags/sdkdownload.html#1>.
  - Download the configuration RPM file from the IBM Tools Repository for Linux on Power website at <http://www14.software.ibm.com/webapp/set2/sas/f/lopdiags/yum.html>.

**Note:** The code is now downloaded as `ibm-power-repo-latest.noarch.rpm`

2. Install the Linux on Power Software Repository initialization package on the server as follows:

a. Log in as root user.

b. Enter the following command:

For RHEL: `yum install ibm-power-repo-latest.noarch.rpm`

For Fedora: `dnf install ibm-power-repo-latest.noarch.rpm`

For SLES: `zypper install ibm-power-repo-latest.noarch.rpm`

### 2.1.2.2 Installing the IBM SDK for Linux on Power RPMs using the IBM Linux on Power Tools Repository

After the IBM Linux on Power Tools Repository is installed, you can download the IBM SDK for Linux on Power RPMs to the server.

#### About this task

Install the SDK packages with the appropriate command for your distribution:

#### Procedure

- For Red Hat Enterprise Linux:

```
yum install ibm-sdk-lop
```

- For SUSE Linux Enterprise Server:

```
zypper install ibm-sdk-lop
```

- For Fedora:

```
dnf install ibm-sdk-lop
```

**Note:** The IBM Linux on Power Tools Repository is not available for Ubuntu.

#### Results

The `ibm-sdk-lop` package is installed, along with other dependent packages, such as the IBM Advance Toolchain. Depending on the packages you installed with your Linux distribution, additional dependent packages from the distribution repository might also be installed. See “Other required packages from the distribution” on page 2 for a list of possible packages.

#### What to do next

After completing the installation of the SDK, consider installing recommended and optional packages on the Power Systems server. See 2.2, “Recommended and optional packages,” on page 9.

### 2.1.3 Automatically downloading and installing the IBM SDK for Linux on Power on Ubuntu

You can download and install the packages for the SDK on Power Systems server by using the apt repository available for the SDK.

#### About this task

Install the SDK packages on Ubuntu, either on 14.04 (trusty) or 16.04 (xenial).

#### Procedure

1. Download and import the GPG public key using the following commands:

```
$ wget ftp://public.dhe.ibm.com/software/server/iplsdk/v1.10.0/packages/deb/repo/dists/<trusty or xenial>/B346CA20.gpg.key
```



- ```
$ sudo apt-key add B346CA20.gpg.key
```
2. Add the following line to `/etc/apt/sources.list` file:  
On amd64:  

```
deb [arch=amd64] ftp://public.dhe.ibm.com/  
software/server/iplsdk/v1.10.0/packages/deb/repo <trusty or xenial> sdk
```

  
On ppc64el:  

```
deb ftp://public.dhe.ibm.com/  
software/server/iplsdk/v1.10.0/packages/deb/repo <trusty or xenial> sdk
```
  3. Run update: `$ sudo apt-get update`
  4. Install the IBM SDK for Linux on Power: `$ sudo apt-get install ibm-sdk-lop`

## Results

The IBM SDK for Linux on Power is installed on your system.

**Note:** After installing the SDK, be sure to log off to ensure that the group rights are applied correctly.

## What to do next

After you complete the installation of the SDK, consider installing recommended and optional packages on the Power Systems server. See 2.2, “Recommended and optional packages.”

---

## 2.2 Recommended and optional packages

Depending on your intended use of the SDK, you might want to install recommended and optional packages.

### 2.2.1 Downloading and installing the IBM XL C/C++ Community Edition Compiler

XL C/C++ Community Edition is a free-of-charge product (unlimited license for production use) and does not include official IBM support.

You can automatically download and install the XL C/C++ Community Edition using the Remote Setup Wizard plugin or you can download and install it manually.

#### 2.2.1.1 Manually installing the IBM XL C/C++ Compiler

Use these instructions to manually download and install the IBM XL C/C++ compiler on your remote system.

#### About this task

Install the IBM XL C/C++ Community Edition Compiler on Power Systems Servers.

#### Procedure

Select your distribution:

- For Ubuntu:

Configure the correct repository by using the following commands:

```
wget -q http://public.dhe.ibm.com/software/server/POWER/Linux/xl-compiler/eval/ppc64le/ubuntu/public.gpg  
-O- | sudo apt-key add -  
echo "deb http://public.dhe.ibm.com/software/server/POWER/Linux/xl-compiler/eval/ppc64le/ubuntu/  
trusty main" | sudo tee /etc/apt/sources.list.d/ibm-xl-compiler-eval.list  
sudo apt-get update
```

Install IBM XL C/C++ Community Edition for Linux, V13.1.4 by using the following commands:

```
sudo apt-get install xlc.13.1.4 xlc-license-community.13.1.4
sudo /opt/ibm/xlC/13.1.4/bin/xlc_configure
```

- For Fedora and Red Hat Enterprise Linux and CentOS:

Configure the correct repository by using the following commands:

```
wget http://public.dhe.ibm.com/software/server/POWER/Linux/xl-compiler/eval/ppc64le/rhel7/repdata/
repomd.xml.key
```

```
sudo rpm --import repomd.xml.key
```

```
wget http://public.dhe.ibm.com/software/server/POWER/Linux/xl-compiler/eval/ppc64le/rhel7/ibm-xl-compiler
-eval.repo
```

```
sudo cp ibm-xl-compiler-eval.repo /etc/yum.repos.d/
```

Install IBM XL C/C++ Community Edition for Linux, V13.1.4, by using the following commands:

```
sudo yum install xlc.13.1.4 xlc-license-community.13.1.4 --exclude xlc-license-eval.13.1.4
sudo /opt/ibm/xlC/13.1.4/bin/xlc_configure
```

- For SLES:

Configure the correct repository by using the following commands:

```
sudo zypper addrepo -c http://public.dhe.ibm.com/software/server/POWER/Linux/xl-compiler/eval/ppc64le/sles12/
ibm-xl-compiler-eval
sudo zypper refresh
```

Install IBM XL C/C++ Community Edition for Linux, V13.1.4, by using the following commands:

```
sudo zypper install xlc.13.1.4 xlc-license-community.13.1.4
sudo /opt/ibm/xlC/13.1.4/bin/xlc_configure
```

## Results

IBM XL C/C++ Community Edition Compiler is installed in the remote system.

### 2.2.1.2 Automatically installing with the Remote Setup Wizard plugin

Use these instructions to automatically download and install the IBM XL C/C++ on your remote system with the Remote Setup Wizard plugin.

#### About this task

Complete the following steps:

#### Procedure

1. After installing the IBM SDK for Linux on Power on your workstation, go to **Help > Setup Remote Machine**.
2. Click **Setup Machine** to manage an existing remote connection (see (reference) to add a new one);
3. In the dialog box, select **IBM XL C/C++ Community Edition** from the **Available IBM XL Compiler versions** field.
  - a. Select the SDK packages to install
  - b. Select at least one Advance Toolchain version (even if already installed).
4. Click **Install** and accept the licenses.

## Results

IBM XL C/C++ Community Edition Compiler is installed in the remote system along with the other packages you selected in the Remote Setup Wizard dialog box.

## 2.2.2 Installing the Advance Toolchain for Linux on Power

### About this task

The Advance Toolchain for Linux on Power packages are stored in an ftp repository of the University of Campinas, a public university in the state of São Paulo, Brazil. For information about installing Advance Toolchain for Linux on Power, see the IBM developerWorks® page(<http://ibm.co/AdvanceToolchain>).

### Procedure

Install the Advance Toolchain for Linux on Power on the Power Systems server. This table lists the required and recommended prerequisite packages for Advance Toolchain for Linux on Power.

Table 4. Advance Toolchain for Linux on Power required and recommended prerequisites

| Advance Toolchain for Linux on Power required prerequisites                                                                                                                                                                                                                       | Advance Toolchain for Linux on Power recommended prerequisites                                                                                                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• advance-toolchain-at9.0-runtime-9.0-5 or greater</li><li>• advance-toolchain-at9.0-devel-9.0-5 or greater</li><li>• advance-toolchain-at9.0-perf-9.0-5 or greater</li><li>• advance-toolchain-at9.0-mcore-libs-9.0-5 or greater</li></ul> | <ul style="list-style-type: none"><li>• advance-toolchain-at10.0-runtime-10.0-0 or greater</li><li>• advance-toolchain-at10.0-devel-10.0-0 or greater</li><li>• advance-toolchain-at10.0-perf-10.0-0 or greater</li><li>• advance-toolchain-at10.0-mcore-libs-10.0-0 or greater</li></ul> |

---

## 2.3 Starting the IBM SDK for Linux on Power

You can start the SDK on the Power Systems server, and access it from your workstation.

### About this task

To start the SDK, complete the following steps.

### Procedure

1. On your workstation, begin either X11 forwarding or Virtual Network Computing (VNC) to connect to the remote Power Systems server.

X11 forwarding provides the most seamless integration between the workstation and the server. However, if your connection is not fast, you might experience network latency. In that case, VNC might be a better option.

VNC works with either Linux or Microsoft Windows clients. In addition, when using VNC, you can disconnect from the session while the IBM SDK for Linux on Power continues to run on the server.

- To use X11 forwarding, ensure that you have a running X Window System server as well as a terminal emulator for the X Window System, such as GNOME Terminal, on your workstation, and the xauth utility and X server font package installed on the server. To connect to the server, open a terminal session and enter the **ssh** command with the **-X** option.
- To use VNC, ensure that you have a VNC server package installed on the Power Systems server, as well as a terminal emulator for the X Window System, such as GNOME Terminal. You also need a VNC client on your workstation. To start VNC, complete the following steps:
  - a. Log in to the server console and start a new VNC server by entering the **vncserver** command.
  - b. When prompted, choose and type a password. You will use this password later to connect to the session.

When the command completes, it displays an address you will use to connect.

- c. On the workstation, use the VNC client to connect. Specify the address that was displayed and the password that you chose.

2. Once connected to the Power Systems server, choose one of the following options to start IBM SDK for Linux on Power.
  - Open a console and enter the following command:

```
ibm-sdk-top
```
  - If your VNC client provides you access to the System menus, select **Applications > Programming > IBM SDK for Linux on Power**.

**Note:** Depending on the desktop environment you are using, the hierarchy of the System menus might differ.

## Results

**Note:** The first time it is started, the IBM SDK for Linux on Power might take several minutes to launch. The operating system might consider the application unresponsive due to the intensive input/output operations necessary to create the initial workspace. If you are prompted to wait or to end the application, select **Wait**. The user interface loads within a few seconds. If prompted, select the location on the Power Systems server that you will use to store the project files.

---

## 2.4 Creating a project

With the SDK, you can create projects or import existing ones. You can leverage the integration with the Advance Toolchain to set specific flags for Power Systems.

For more information about remote projects, see 3.3, “Creating and using synchronized projects on x86\_64/amd64 clients,” on page 22

### 2.4.1 Creating a C/C++ project directly on a Power Systems server using Advance Toolchain

You can create a C/C++ project that uses Advance Toolchain for Linux on Power.

#### About this task

Complete the following steps in the SDK user interface.

#### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Advance Toolchain C/C++**. Select either **Advance Toolchain C Project** or **Advance Toolchain C++ Project**, as appropriate. Click **Next**.
3. In the Project window, type a name for the project in the **Project name** field.

**Tip:** At the bottom of the window, verify that the **Show projects type and toolchains only if they are supported on the platform** check box is selected.

4. Optional: The project will be created in a directory structure in the file system. The default file system location is displayed in the **Location** field. If you do not want to save the project in the default location, clear the **Use default location** check box, and specify or browse for a new location.
5. In the Project type pane, expand one of the following, as appropriate:
  - **Executable**
  - **Shared Library**
  - **Static Library**

Then, select **Advance Toolchain C Project** or **Advance Toolchain C++ Project**, as appropriate. If you do not see Advance Toolchain for Linux on Power in the Toolchains list, ensure that you have installed a supported version of Advance Toolchain. For more information, see 1.1, “IBM SDK for Linux on Power requirements,” on page 2.

6. In the Toolchains pane, click the **Advance Toolchain *version*** option corresponding to your Advance Toolchain for Linux on Power version. Click **Next**.
7. In the Compiler tuning window, select the appropriate options for tuning your application or library.
  - a. **Environment:** Select from - **use default -**, **64-bit**, or **32-bit**.
  - b. **Generate POWER-series code that is compatible with:** Select from - **use default -** or the listed **POWER $n$**  technology options.
  - c. **Tune the instruction scheduling for:** Select from - **use default -** or the listed **POWER $n$**  technology options. Click **Next**.
8. Optionally in the Templates window, you can select a project template for a specific version. In the **Library** list, select from the following options:
  - **Do not use any library template.** This is the default option.
  - **SPHDE (Shared Persistent Heap Data Environment).** SPHDE is composed of two major software layers: The Shared Address Space (SAS) layer provides the basic services for a shared address space and transparent, persistent storage. The Shared Persistent Heap (SPH) layer organizes blocks of SAS storage into useful functions for storing and retrieving data.
  - **AUXV (Auxiliary Vector).** The AUX vector contains information about the system's platform and hardware capabilities.
9. Optional: To allow optimizations for Power Systems servers when you are building the application or library, complete these steps:
  - a. In the Select configurations window, click **Advanced settings**.
  - b. In the next window, expand **C/C++ Build** and click **Settings**.
  - c. On the **Tool Settings** tab, click **POWER-specific optimizations**. Then, select the appropriate optimization options for your workload and application or library characteristics, and target processor. You can change the default build settings to tune for POWER<sup>®</sup> processor capabilities and the IBM SDK for Linux on Power analysis tools. See 4.1, “Setting flags,” on page 35 for information about the recommended build settings, and instructions for setting them. When finished, click **OK**.
10. Click **Finish**.

## Results

The new project is shown in the Project Explorer pane and will be built using the Advance Toolchain for Linux on Power, with any Power-specific optimizations that you have chosen.

### 2.4.2 Creating a local C/C++ project direct on a Power Systems server using the IBM XL C/C++ Community Edition Compiler

You can create a C/C++ project that uses the IBM XL C/C++ Community Edition Compiler.

#### About this task

Complete the following steps in the SDK user interface.

#### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **IBM XL C/C++ Community Edition**. Select either **IBM XL Community Edition C Project** or **IBM XL Community Edition C++ Project**, as appropriate. Click **Next**.
3. In the Project window, type a name for the project in the **Project name** field.

**Tip:** At the bottom of the window, verify that the **Show projects type and toolchains only if they are supported on the platform** check box is selected.

- Optional: The project will be created in a directory structure in the file system. The default file system location is displayed in the **Location** field. If you do not want to save the project in the default location, clear the **Use default location** check box, and specify or browse for a new location.
- In the Project type pane, expand one of the following, as appropriate:
  - **Executable**
  - **Shared Library**
  - **Static Library**

Then, select **IBM XL Community Edition C Project** or **IBM XL Community Edition C++ Project**, as appropriate. If you do not see IBM XL C/C++ Community Edition Compiler in the Toolchains list, ensure that you have installed it. For more information, see 2.2.1, “Downloading and installing the IBM XL C/C++ Community Edition Compiler,” on page 9.

### 2.4.3 Importing an existing Makefile project

You can import an existing Makefile project into the IBM Software Development Kit for Linux on Power to be used with Advance Toolchain for Linux on Power.

#### Before you begin

Ensure that the Makefile project already exists on or has been saved to the Power Systems server.

#### About this task

To import a Makefile project, complete the following steps in the SDK user interface.

#### Procedure

- Before importing the project, you should disable the option to build projects automatically. This prevents the SDK from building a project that might not be ready after importing. Click **Project** and clear the **Build Automatically** check box if it is selected.
- Import the project by clicking **File > Import**.
- In the Import window, expand **C/C++**, then click **Existing Code as Makefile Project**. Click **Next**.
- In the Import Existing Code window, do the following.
  - Click **Browse** next to the **Existing Code Location** field. The project must exist on the Power Systems server.
  - Optional: Type a name for the project in the **Project Name** field.
  - Successively click the displayed directories to locate the directory that contains the project to be imported.
  - When you locate the directory, click **OK**.
  - Back in the Import Existing Code window, under Toolchain for Indexer Settings, click the **Advance Toolchain version** option corresponding to your Advance Toolchain for Linux on Power version. *version* is the version number of Advance Toolchain for Linux on Power.
  - Click **Finish**.

The project is imported as an Makefile project and is shown in the Project Explorer pane.

- You must export the environment variables **CC** or **GCC** to ensure that the **make** command builds using the correct Advance Toolchain for Linux on Power version. Choose one of the following methods.
  - Use the user interface options to change the settings of the environment variables.
    - View the project by clicking the project name in the Project Explorer pane.
    - Right-click the project name and click **Properties**.

- c. Expand **C/C++ Build** and click **Environment**.
  - d. Click **Export** and type `CC="/opt/at $version$ /bin/gcc".` where *version* is the version number of Advance Toolchain for Linux on Power.
  - e. Click **OK** when finished.
- Change the environment variables manually by editing the Makefile. You edit the Makefile by double-clicking on the project in the Project Explorer pane.
6. Optional: You can change the default build settings to tune for POWER processor capabilities and the IBM SDK for Linux on Power analysis tools. See 4.1, “Setting flags,” on page 35 for information about the recommended build settings, and instructions for setting them.

To change build settings such as CFLAGS, CPPFLAGS, and LDFLAGS, choose one of the following methods.

- Use the user interface options to change the settings of the environment variables.
  - a. View the project by clicking the project name in the Project Explorer pane.
  - b. Right-click the project name and click **Properties**.
  - c. Expand **C/C++ Build** and click **Environment**.
  - d. Click **Add** to add environment variables, or click **Select** to select and change the value of an existing environment variable. For example, a typical set of parameters for POWER8<sup>®</sup> targets is
 

```
CC=/opt/at10.0/bin/gcc CFLAGS='-m64 -g -O3 -mcpu=power8' CXXFLAGS='-m64 -g -O3 -mcpu=power8' LDFLAGS='-m64 -Wl,-q'.
```
  - e. Click **OK** when finished.
- Change the build settings manually by editing the Makefile. You edit the Makefile by double-clicking on the project in the Project Explorer pane.

## 2.4.4 Importing an existing Autotools project

You can import an existing project that uses Autotools into the IBM Software Development Kit for Linux on Power to be used with Advance Toolchain for Linux on Power.

### Before you begin

Ensure that the Autotools project already exists on or has been saved to the Power Systems server.

### About this task

To import an Autotools project, complete the following steps in the SDK user interface.

### Procedure

1. Before importing the project, you should disable the option to build projects automatically. This prevents the SDK from building a project that might not be ready after importing. Click **Project** and clear the **Build Automatically** check box if it is selected.
2. Import the project by clicking **File > Import**.
3. In the Import window, expand **C/C++**, then click **Existing Code as Autotools Project**. Click **Next**.
4. In the Import Existing Code window, do the following.
  - a. Click **Browse** next to the **Existing Code Location** field. The project must exist on the Power Systems server.
  - b. Optional: Type a name for the project in the **Project Name** field.
  - c. Successively click the displayed directories to locate the directory that contains the project to be imported.
  - d. When you locate the directory, click **OK**.
  - e. Click **Finish**.

The project is imported as an Autotools project and is shown in the Project Explorer pane.

## 2.4.4.1 Configuring the imported project

### About this task

After the project is imported, the SDK automatically indexes the source code for the project. The source files then appear with the project in the Project Explorer pane.

### Procedure

1. To view the project, click the project name in the Project Explorer pane. You can explore it and make configuration updates.
  - Use the Project Explorer pane to expand and explore the project.
  - Double-click a project file to open it in the editor. The outline pane shows macro references and program control statements.
  - Open the autoconf configuration file (`configure.ac`) or the makefile configuration file (`makefile.am`). This allows the SDK to index the file and provide proper syntax highlighting.
  - You can set autoconf configuration script options in the project properties.
    - Click **Project > Properties**.
    - In the Properties window, expand **Autotools** and click **Configure Settings**.
    - Under Configure, click **Advanced**.
    - You can specify additional options in the **Additional command-line options** field. To use Advance Toolchain for Linux on Power with Autotools projects, type `CC="/opt/atversion/bin/gcc"`, where *version* is the version number of Advance Toolchain for Linux on Power. For example: `/opt/at10.0/bin/gcc`. On this field, you can also set other flags for your project. For example, a typical set of parameters for POWER8 targets is

```
CC=/opt/at10.0/bin/gcc CFLAGS='-m64 -g -O3 -mcpu=power8' CXXFLAGS='-m64 -g -O3 -mcpu=power8' LDFLAGS='-m64 -Wl,-q'
```
    - Click **OK**.
2. After you have made the configuration updates, reconfigure the project.
3. Right-click the project name, and select **Reconfigure Project**.
4. While the project is being configured, you can click the **Console** tab to monitor progress.

## 2.4.4.2 Building the imported project

### About this task

After the project reconfiguration is complete, you can build the project.

### Procedure

1. Return to the project view by clicking the project name in the Project Explorer pane.
2. Click **Project > Build Project** to build the project.
3. While the project is building, you can click the **Console** tab to monitor progress.

### Related information:

 [An Introduction to the Autotools](#)

## 2.4.5 Creating an empty Autotools project

You can use the SDK to create an empty Autotools project.

### About this task

Complete the following steps in the SDK user interface.



## Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Advance Toolchain C/C++**. Select either **Advance Toolchain C Project** or **Advance Toolchain C++ Project**, as appropriate. Click **Next**.
3. In the Project type pane, expand **GNU Autotools** and select **Empty Project**.
4. In the Project window, type a name for the project in the **Project name** field.
5. The project will be created in a directory structure in the file system. The default file system location is displayed in the **Location** field. If you do not want to save the project in the default location, clear the **Use default location** check box, and specify or browse for a new location.
6. Click **Finish**.

## Results

The new project is shown in the Project Explorer pane.

### 2.4.6 Creating a “Hello World” Autotools project

You can use the SDK to create a “Hello World” Autotools project.

#### About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

## Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Advance Toolchain C/C++**. Select either **Advance Toolchain C Project** or **Advance Toolchain C++ Project**, as appropriate. Click **Next**.
3. In the Project type pane, expand **GNU Autotools** and select **Hello World ANSI C Autotools Project**.
4. In the Project window, type a name for the project in the **Project name** field.
5. The project will be created in a directory structure in the file system. The default file system location is displayed in the **Location** field. If you do not want to save the project in the default location, clear the **Use default location** check box, and specify or browse for a new location.
6. Click **Finish**.

## Results

The new project is shown in the Project Explorer pane.

### 2.4.7 Changing the Advance Toolchain version

After you have created a C/C++ Development Toolkit managed C/C++ project that uses the Advance Toolchain for Linux on Power, you can switch between installed Advance Toolchain for Linux on Power versions as needed.

#### About this task

Complete the following steps in the SDK user interface.

## Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Right-click the project name and click **Properties**.
3. Expand **C/C++ Build** and click **Tool Chain Editor**.

4. In the Tool Chain Editor window, in the **Current toolchain** field, click the arrows to select the Advance Toolchain for Linux on Power version you want to use.
5. Optional: You can verify the version of Advance Toolchain for Linux on Power being used by doing the following:
  - a. Right-click the project name and click **Properties**.
  - b. Expand **C/C++ Build** and click **Settings**.
6. Switch the Linux tools path according to the Advance Toolchain for Linux on Power version, to ensure that the analysis and profiling tools use the correct Advance Toolchain for Linux on Power binary files. Complete the following steps:
  - a. Right-click the project name and click **Properties**.
  - b. Click **Linux Tools Path**.
  - c. Ensure that **Prepend string to PATH** is selected.
  - d. Select the appropriate Advance Toolchain for Linux on Power version.

## 2.4.8 Running an executable program in a project

This topic provides instructions for running executable programs in local projects.

### About this task

To run an executable program, complete the following steps in the SDK user interface.

#### Procedure

1. In the Project Explorer pane, right-click the project name. Click **Run as > Run configurations**.
2. In the open space under **C/C++ Application**, double-click to create a new configuration.
3. In the New Run Configuration window, beside the **C/C++ Application** field, click **Search Project** to select the executable program that you want to run.
4. On the Arguments tab, specify any arguments.
5. Click **Run**.

## 2.4.9 Debugging a project with Advance Toolchain for Linux on Power or IBM XL C/C++ Community Edition Compiler

This topic provides instructions for debugging a local project using the debugger provided by Advance Toolchain for Linux on Power or IBM XL C/C++ Community Edition Compiler.

### About this task

The C/C++ Development User Guide ([http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt\\_o\\_home.htm](http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm)) in the Eclipse platform help information contains additional general and advanced information about debugging projects.

To begin debugging a project, complete the following steps in the SDK user interface.

#### Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Click **Debug as > Debug Configurations**.
3. In the Debug Configurations window, click **C/C++ Application** in the left pane, and then click the new launch configuration icon near the upper left of the window.
4. On the Main tab, beside the **C/C++ Application** field, click **Search Project** to select the binary file that you want to debug.
5. If necessary, click the arrows in the **Build configuration** field and select **Debug**.

6. On the Arguments tab, specify any arguments.
7. Optional: The GDB debugger path was set when the project was created. However, you can check the path as follows. On the Debugger tab, on the Main tab within the Debugger Options area, in the **GDB debugger** field, verify the path for the location of the debugger to be used, `/opt/at $version$ /bin/gdb`, where *version* is the version number of Advance Toolchain for Linux on Power. For example: `/opt/at10.0/bin/gdb`. If you are debugging a project built with IBM XL C/C++ Community Edition Compiler ensure that the path of the debugger contains only `gdb`.
8. Click **Apply**, then click **Debug** to begin debugging.
9. You might be prompted to open the Debug perspective. If so, click **Yes** to open the Debug perspective.
10. In the Debug perspective, you can use the displayed icons to step through the execution, inspect variables, and set breakpoints.



---

## 3 Setting up the x86\_64/amd64 version of the IBM SDK for Linux on Power

You can start the IBM SDK for Linux on Power x86\_64/amd64 client on a workstation and then connect to a remote Power System to build, execute, debug, and analyze programs. You can also develop and build your application locally using the Advance Toolchain for Linux on Power cross-compiler.

For remote development, you must install the required dependencies on the remote Power System. The `ibm-sdk-lop-remote-dependencies` package ensures that all required dependencies are installed on the remote Power Systems.

---

### 3.1 Downloading and installing the x86\_64/amd64 client

This topic gives instructions for downloading the and installing the x86\_64 version of the SDK.

#### About this task

Install the x86\_64/amd64 version of the SDK.

#### Procedure

1. Install the Advance Toolchain cross-compiler on the x86\_64/amd64 system, if needed

*Table 5. Advance Toolchain for Linux on Power required and recommended prerequisites*

| Advance Toolchain for Linux on Power required prerequisites                                                                                                                                                                                                                                                                                                                                                         | Advance Toolchain for Linux on Power recommended prerequisites                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• <code>advance-toolchain-at9.0-cross-common-9.0-5</code></li><li>• <code>advance-toolchain-at9.0-cross-arch-9.0-5</code><br/>Where <i>arch</i> can be either <code>ppc64</code> or <code>ppc64le</code>.</li><li>• <code>advance-toolchain-at9.0-cross-arch-mcore-libs-9.0-5</code></li><li>• <code>advance-toolchain-at9.0-cross-arch-runtime-extras-9.0-5</code></li></ul> | <ul style="list-style-type: none"><li>• <code>advance-toolchain-at10.0-cross-common-10.0-0</code></li><li>• <code>advance-toolchain-at10.0-cross-arch-10.0-0</code></li><li>• <code>advance-toolchain-at10.0-cross-arch-mcore-libs-10.0-0</code></li><li>• <code>advance-toolchain-at10.0-cross-arch-runtime-extras-10.0-0</code></li></ul> <p>Where <i>arch</i> can be either <code>ppc64</code> or <code>ppc64le</code>.</p> |

For more details about installing the Advance Toolchain for Linux on Power, see 2.2.2, “Installing the Advance Toolchain for Linux on Power,” on page 11.

2. For automatic installation, refer to 2.1, “Downloading and installing the SDK,” on page 7 or go to the IBM Software Development Kit for Linux on Power website at <http://www-304.ibm.com/webapp/set2/sas/f/lopdiags/sdkdownload.html#2..>
3. Select from the following choices to download the x86\_64/amd64 version of the SDK.
  - For Red Hat Enterprise Linux, SUSE Linux Enterprise, Fedora, and CentOS in the Download individual packages for IBM SDK section of the page, download `ibm-sdk-lop-version.x86_64.rpm`.
  - For Ubuntu, in the Download the ISO image or the DEB image section of the page, download the DEB image for the x86\_64/amd64 client, `ibm-sdk-lop_version_amd64.deb`.

*version* is the version number of the package.

4. Install the IBM SDK for Linux on Power and its dependencies on the x86\_64/amd64 host:
  - For Red Hat Enterprise Linux and CentOS:  

```
# yum install ibm-sdk-lop-version.x86_64.rpm
```
  - For SUSE Linux Enterprise Server:  

```
# zypper install ibm-sdk-lop-version.x86_64.rpm
```

- For Ubuntu:  

```
sudo dpkg -i ibm-sdk-lop_version_amd64.deb
```
- For Fedora:  

```
# dnf install ibm-sdk-lop-version.x86_64.rpm
```

If the installer warns you about missing dependencies on Ubuntu, run the following command to automatically download and install them:

```
sudo apt-get -f install
```

## What to do next

**Note:** After installing the SDK, be sure to log off to ensure that the group rights are applied correctly.

---

## 3.2 Setting up remote Power Systems server

The Remote Setup Wizard available on the x86\_64/amd64 of the SDK allows users to install the SDK packages and their dependencies on a remote Power Systems server.

### About this task

Complete the following steps to install the SDK with the Remote Setup Wizard:

#### Procedure

1. Click **Help** in the toolbar and select **Setup Remote Machine**.
2. If no systems are listed in table, complete the following steps:
  - a. Click **New Machine**.
  - b. In the **Remote Services** field, select **SSH** and click **Add**.
  - c. In the New Connection window, complete the fields: New Connection Name, Host, User, and Password. Then, select **Finish**.
  - d. In the Preferences window, click **OK**.
3. Select one of the systems that are listed in the table and click **Setup Machine**.
4. In the new window, select the SDK packages and the Advance Toolchain version to be installed. If you want to install IBM XL C/C++ Community Edition Compiler, select that option. Click **Install**.

**Note:** To proceed with installation, you need to agree with the license terms.

5. All the necessary packages will be downloaded. When the download process is completed, a confirmation message is displayed, informing which packages will be installed. Select **Yes** to install those packages.
6. During the installation process, you are prompted to enter your password. Type your user password and select **OK**.

**Note:** You need administrative privileges to install the packages.

7. When the installation is completed, a message is displayed, informing you that all the packages were successfully installed. Select **OK** to complete the process.

---

## 3.3 Creating and using synchronized projects on x86\_64/amd64 clients

If you have installed the x86\_64/amd64 version of the SDK on you workstation and the dependencies package on a Power Systems server, you can create synchronized projects on the remote server from the client.

See 3.2, “Setting up remote Power Systems server,” on page 22 for instructions if you have not already installed the packages.

### 3.3.1 Creating a synchronized C/C++ project

Synchronized projects are mirrored on the local system and the remote system. You can edit locally, and your changes are synchronized with the remote system. This topic details how to create a synchronized project that uses Advance Toolchain for Linux on Power.

#### Before you begin

If you plan to use synchronized projects, you must first install the git package on the remote server. See “Packages required by IBM SDK for Linux on Power ” on page 2.

#### About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

#### Procedure

1. Click **File** > **New** > **Other**.
2. In the New window, expand **Remote Advance Toolchain** and click to select **Synchronized Advance Toolchain C/C++ Project**. Click **Next**.
3. In the New Advance Toolchain Synchronized Project window, type a name for the project in the **Project name** field.
4. In the **Connection Name** field, click the arrows to select a connection to the remote server, or click **New** to create a connection.

**Tip:** If you are creating a connection, use your ssh credentials instead of public key. See 9.3, “Setting up SSH credentials,” on page 93.

5. For the location, click **Browse** to select the remote location where the project source is stored.
6. In the Project type pane, expand and select a project type for one of the following categories:
  - Remote Advance Toolchain Executable
  - Remote Advance Toolchain Shared Library
  - Remote Advance Toolchain Static Library

In the project type, select **Empty Project**.

7. In the Toolchain pane, select an available Remote Advance Toolchain. Do not select “Local Toolchain”.
8. Click **Next** or **Finish**.

#### Results

The new project is shown in the Project Explorer pane and will be built using the Advance Toolchain for Linux on Power, with any Power-specific optimizations that you have chosen. The project folder is created on the remote Power Systems server, and a local cached copy is maintained. IBM SDK for Linux on Power operations such as building, running, and profiling can be performed remotely.

### 3.3.2 Creating a synchronized project by importing an existing Makefile project

You can import an existing Makefile project into the SDK as a synchronized project to be used with Advance Toolchain for Linux on Power.

## Before you begin

If you plan to use synchronized projects, ensure that the Makefile project already exists on or has been saved to the Power Systems server.

## About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Remote Advance Toolchain** and click to select **Advance Toolchain Synchronized C/C++ Project**. Click **Next**.
3. In the New Advance Toolchain Synchronized Project window, type a name for the project in the **Project name** field.
4. In the **Connection Name** field, click the arrows to select a connection to the remote server, or click **New** to create a connection.

**Tip:** If you are creating a connection, use your ssh credentials instead of public key. See 9.3, “Setting up SSH credentials,” on page 93.

5. For the location, click **Browse** to select the remote location where the project source is stored.
6. In the Project type pane, expand **Makefile Project** and select **Empty Project**.
7. In the Remote Toolchain pane, select the desired **Remote Linux Advance Toolchain**. Do not select “Local Toolchain”.
8. Click **Finish**.

## 3.3.3 Creating a synchronized project by importing an existing Autotools project

You can import an existing project that uses Autotools into the SDK to be used remotely with Advance Toolchain for Linux on Power.

## Before you begin

Ensure that the Autotools project already exists on or has been saved to the Power Systems server.

## About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

### Procedure

1. Before importing the project, you should disable the option to build projects automatically. This prevents the SDK from building a project that might not be ready after importing. Click **Project** and clear the **Build Automatically** check box if it is selected.
2. Click **File > New > Other**.
3. In the New window, expand **Remote Advance Toolchain** and click to select **Synchronized Advance Toolchain C/C++ Project**. Click **Next**.
4. In the New Advance Toolchain Synchronized Project window, type a name for the project in the **Project name** field.
5. In the **Connection Name** field, click the arrows to select a connection to the remote server, or click **New** to create a connection.



**Tip:** If you are creating a connection, use your ssh credentials instead of public key. See 9.3, “Setting up SSH credentials,” on page 93.

6. For the location, click **Browse** to select the remote location where the project source is stored.
7. In the Project type pane, expand **GNU Autotools**, and select **Empty project**.
8. In the Remote Toolchain pane, select **GNU Autotools Toolchain**.
9. Click **Finish**.

## Results

The project is imported as an Autotools project and is shown in the Project Explorer pane.

## What to do next

Configuring the imported project.

### 3.3.3.1 Configuring the imported project

#### About this task

After the project is imported, the SDK automatically indexes the source code for the project. The source files then appear with the project in the Project Explorer pane.

#### Procedure

1. To view the project, click the project name in the Project Explorer pane. You can explore it and make configuration updates.
  - Use the Project Explorer pane to expand and explore the project.
  - Double-click a project file to open it in the editor. The outline pane shows macro references and program control statements.
  - Open the autoconf configuration file (`configure.ac`) or the makefile configuration file (`makefile.am`). This allows the SDK to index the file and provide proper syntax highlighting.
  - You can set autoconf configuration script options in the project properties.
    - Click **Project > Properties**.
    - In the Properties window, expand **Autotools** and click **Configure Settings**.
    - Under **Configure**, click **Advanced**.
    - You can specify additional options in the **Additional command-line options** field. To use Advance Toolchain for Linux on Power with Autotools projects, type `CC="/opt/atversion/bin/gcc"`, where *version* is the version number of Advance Toolchain for Linux on Power. For example: `/opt/at10.0/bin/gcc`. On this field, you can also set other flags for your project. For example, a typical set of parameters for POWER8 targets is

```
CC=/opt/at10.0/bin/gcc CFLAGS='-m64 -g -O3 -mcpu=power8' CXXFLAGS='-m64 -g -O3 -mcpu=power8' LDFLAGS='-m64 -Wl,-q'
```
    - Click **OK**.
2. After you have made the configuration updates, reconfigure the project.
3. Right-click the project name, and select **Reconfigure Project**.
4. While the project is being configured, you can click the **Console** tab to monitor progress.

### 3.3.3.2 Building the imported project


#### About this task

After the project reconfiguration is complete, you can build the project.

## Procedure

1. Return to the project view by clicking the project name in the Project Explorer pane.
2. Click **Project > Build Project** to build the project.
3. While the project is building, you can click the **Console** tab to monitor progress.

### Related information:

 [An Introduction to the Autotools](#)

## 3.3.4 Creating a synchronized empty Autotools project

You can use the IBM SDK for Linux on Power to create an empty synchronized Autotools project.

### About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

#### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Remote Advance Toolchain** and click to select **Synchronized Advance Toolchain C/C++ Project**. Click **Next**.
3. In the **Connection Name** field, click the arrows to select a connection to the remote server, or click **New** to create a connection.

**Tip:** If you are creating a connection, use your ssh credentials instead of public key. See 9.3, “Setting up SSH credentials,” on page 93.

4. Click **Browse** to find the location where the project is to be stored on the remote server.
5. In the Project type pane, expand **GNU Autotools** and select **Empty Project**.
6. In the Project window, type a name for the project in the **Project name** field.
7. Click **Finish**.

### Results

The new project is shown in the Project Explorer pane.

**Note:** Running Autotools in a remote project will fail if there are environment variables with no value assigned in the local system (for example, `VARIABLE=`).

## 3.3.5 Creating a synchronized “Hello World” Autotools project

You can use IBM SDK for Linux on Power to create a “Hello World” synchronized Autotools project.

### About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

#### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Remote Advance Toolchain** and click to select **Synchronized Advance Toolchain C/C++ Project**. Click **Next**.
3. In the **Connection Name** field, click the arrows to select a connection to the remote server, or click **New** to create a connection.

**Tip:** If you are creating a connection, use your ssh credentials instead of public key. See 9.3, “Setting up SSH credentials,” on page 93.

4. In the Project type pane, expand **GNU Autotools** and select **Hello World ANSI C Autotools Project**.
5. In the Project window, type a name for the project in the **Project name** field.
6. Click **Finish**.

## Results

The new project is shown in the Project Explorer pane.

**Note:** Running Autotools in a remote project will fail if there are environment variables with no value assigned in the local system (for example, `VARIABLE=` ).

### 3.3.6 Running an executable program in a synchronized project

This topic provides instructions for running an executable program on a remote server.

#### About this task

To run a remote executable program, complete the following steps in the IBM SDK for Linux on Power user interface.

#### Procedure

1. In the Project Explorer pane, right-click the remote project name. Click **Run as > Run configurations**. Double-click **Parallel Application**.
2. On the Resources tab, complete the following steps.
  - a. In the **Target System Configuration** field, click the arrows to select **IBM SDK Remote Connection**.
  - b. In the **Remote service provider** field, select **Remote tools**.
  - c. In the **Please select a connection** box, click the arrows to select the connection to the remote server, or click **New** to create a connection.
3. On the Application tab, in the **Application program** field, click **Browse** to select the executable program that you want to run.
4. Click **Run**.

### 3.3.7 Debugging a synchronized project

This topic provides instructions for debugging a synchronized project using the IBM SDK for Linux on Power.

#### Before you begin

To debug a project on a remote Power Systems server, ensure that all required dependencies are installed by installing the `ibm-sdk-lop-remote-dependencies`.

#### About this task

To debug a synchronized project, complete the following steps.

#### Procedure

1. In the Project Explorer pane, right-click the synchronized project name. Click **Debug as > Debug configurations**. Double-click **Parallel Application**.
2. On the **Resources** tab, complete the following steps.
  - a. In the **Target System Configuration** field, click the arrows to select **IBM SDK Remote Connection**.

- b. In the **Please select a connection** box, click the arrows to select the connection to the remote server, or click **New** to create a connection.
3. On the Application tab, in the **Application program** field, click **Browse** to select the binary file that you want to debug.
4. Click **Debug**. The Parallel Debug perspective opens automatically. You can then debug the application.
5. In the Parallel Debug perspective, you can use the displayed icons to step through the execution, inspect variables, and set breakpoints.

---

## 3.4 Creating and using cross-compiled projects

### 3.4.1 Creating a cross-compiled C/C++ project

You can create a C/C++ project that uses the Advance Toolchain for Linux on Power.

#### About this task

Complete the following steps in the SDK for Linux on Power user interface.

#### Procedure

1. Click **File > New > Other**.
2. In the New window, expand **Advance Toolchain C/C++**. Select either **Advance Toolchain C Project** or **Advance Toolchain C++ Project**, as appropriate. Click **Next**.
3. In the Project window, type a name for the project in the **Project name** field.

**Tip:** At the bottom of the window, verify that the **Show projects type and toolchains only if they are supported on the platform** check box is selected.

4. Optional: The project will be created in a directory structure in the file system. The default file system location is displayed in the **Location** field. If you do not want to save the project in the default location, clear the **Use default location** check box, and specify or browse for a new location.
5. In the Project type pane, expand one of the following, as appropriate:
  - **Executable**
  - **Shared Library**
  - **Static Library**

Then select **Advance Toolchain cross-compiler C Project** or **Advance Toolchain cross-compiler C++ Project**, as appropriate.

6. In the Toolchains pane, select the option corresponding to your Advance Toolchain for Linux on Power cross-compiler. Click **Next**.
7. In the Compiler tuning window, select the appropriate options for tuning your application or library.
  - a. **Environment:** Select from - **use default -**, **64-bit**, or **32-bit**.
  - b. **Generate POWER-series code that is compatible with:** Select from - **use default -** or the listed **POWER $n$**  technology options.
  - c. **Tune the instruction scheduling for:** Select from - **use default -** or the listed **POWER $n$**  technology options. Click **Next**.
8. Optional: To allow optimizations for Power Systems servers when you are building the application or library, complete these steps:
  - a. In the Select configurations window, click **Advanced settings**.
  - b. In the next window, expand **C/C++ Build** and click **Settings**.
  - c. On the **Tool Settings** tab, click **POWER-specific optimizations**. Then, select the appropriate optimization options for your workload and application or library characteristics, and target

processor. You can change the default build settings to tune for POWER processor capabilities and the IBM SDK for Linux on Power analysis tools. See 4.1, “Setting flags,” on page 35 for information about the recommended build settings, and instructions for setting them. When finished, click **OK**.

9. Click **Finish**.
10. If prompted to open the C/C++ perspective, select **Yes**.

## Results

The new project is shown in the Project Explorer pane and will be built using the Advance Toolchain for Linux on Power, with any Power-specific optimizations that you have chosen.

### 3.4.2 Executing a cross-compiled project

This topic provides instructions for running an executable program remotely using the cross-compiler development mode.

#### About this task

To run a cross-compiled executable program, complete the following steps in the IBM SDK for Linux on Power user interface.

#### Procedure

1. Switch to the C/C++ perspective by clicking **Window > Open Perspective > C/C++**. Click **OK**. The cross-compiled project should be displayed under the Project Explorer view.
2. In the Project Explorer pane, right-click the cross-compiled project name. Click **Run as > Run configurations**. Double-click **Parallel Application**.
3. On the Resources tab, complete the following steps.
  - a. In the **Target System Configuration** field, click the arrows to select **IBM SDK Remote Connection**.
  - b. In the **Please select a connection** box, click the arrows to select the connection to the remote server, or click **New** to create a connection.
4. On the Application tab, complete the following steps.
  - a. In the **Application program** field, type a name for the application to be created on the remote server.
  - b. Select the **Copy executable from local filesystem** check box.
  - c. In the **Path to local executable** field, click **Browse** to select the executable program that was generated when the project was built.
  - d. Optional: If you want to see the output from the running application, ensure that the **Display output from all processes in a console view** check box is selected.
5. Click **Run**.

### 3.4.3 Debugging a cross-compiled project

This topic provides instructions for debugging a cross-compiled project using the SDK.

#### Before you begin

To debug a project on a remote Power Systems server, the `ibm-sdk-lop-server-version` package must be installed on the remote server.

#### About this task

To debug a cross-compiled project, complete the following steps.

## Procedure

1. Switch to the C/C++ perspective by clicking **Window > Open Perspective > C/C++**. Click **OK**. The cross-compiled project should be displayed under the Project Explorer view.
2. In the Project Explorer pane, right-click the cross-compiled project name. Click **Debug as > Debug configurations**. Select the parallel application that was created previously when you ran the compiled program. The application has the same name as your project.
3. On the Resources tab, complete the following steps.
  - a. In the **Target System Configuration** field, click the arrows to select **IBM SDK Remote Connection**.
  - b. In the **Please select a connection** box, click the arrows to select the connection to the remote server, or click **New** to create a connection.
4. On the Application tab, in the **Application program** field, click **Browse** to select the binary file that you want to debug.
5. Click **Debug**. The Parallel Debug perspective opens automatically. You can then debug the application.
6. In the Parallel Debug perspective, you can use the displayed icons to step through the execution, inspect variables, and set breakpoints.

---

## 3.5 Installing and using QEMU user-mode emulation

The SDK provides integration with QEMU user-mode emulation. In this mode, QEMU can launch processes compiled for one CPU on another CPU, allowing easy cross-compilation and cross-debugging.

### Before you begin

Before you begin using QEMU user-mode emulation, ensure that you have the following prerequisites installed on your system:

- The Advance Toolchain for Linux on Power cross-compiler package, as detailed in 2.2.2, “Installing the Advance Toolchain for Linux on Power,” on page 11.
- IBM Software Development Kit for Linux on Power

### About this task

At the CPU level, user-mode emulation is a subset of the full system emulation. No memory management unit (MMU) simulation is done because QEMU supposes the user memory mappings are handled by the host operating system. QEMU includes a generic Linux system call converter to handle endianness issues and 32/64 bit conversions. Because QEMU supports exceptions, it emulates the target signals exactly. Each target thread is run in one host thread.

## Procedure

1. If you have installed the IBM Power Repository or configured the apt repository as listed in 2.1, “Downloading and installing the SDK,” on page 7, you can install the package as following:
  - For RHEL and CentOS: `yum install QEMU-user-space-emulator`
  - For SLES: `zypper install QEMU-user-space-emulator`
  - For Ubuntu: `apt-get install QEMU-user-space-emulator`

If you prefer, you can download and install the latest packages for your operating system from the Unicamp FTP site (<ftp://ftp.unicamp.br/pub/linuxpatch/sdk/qemu/>).

2. Install the packages using the command appropriate for your Linux distribution.
  - For Red Hat Enterprise Linux and CentOS:  

```
# yum install -y QEMU-user-space-emulator-<version>.x86_64.rpm
```
  - For SUSE Linux Enterprise Server:

- ```
# zypper install -y QEMU-user-space-emulator-<version>.x86_64.rpm
```
- For Ubuntu:

```
sudo dpkg -i QEMU-user-space-emulator_<version>_amd64.deb
```
  - For Fedora:

```
# dnf install -y QEMU-user-space-emulator-<version>.x86_64.rpm
```

### 3.5.1 Running a cross-compiled application with QEMU user-mode emulation

Follow these steps to run a cross-compiled application with QEMU user-mode emulation.

#### Before you begin

Before using QEMU to run your application, you must create (or import) and build a cross-compiled project. See 3.4.1, “Creating a cross-compiled C/C++ project,” on page 28 for details.

#### About this task

In order to run your cross-compiled application using QEMU user-mode, follow these steps.

#### Procedure

1. Select the target project, then **Run > Run As > QEMU**. If this is the first time you are using QEMU, the launcher configuration opens.
2. From the Application options menu, select **Browse**. A new window appears.
3. Select the application binary. If the Advance Toolchain version cannot be detected, you are prompted to select the toolchain used to build the project. After you select the binary and identify the toolchain version, the remaining fields are automatically populated.
4. Optional: Set arguments for your application from **Applications > Parameters**.
5. Select from the QEMU options. Descriptions for each options display as you move your cursor over the option name.
6. By default, the **Libraries Paths** fields contains the paths for the Advance Toolchain libraries. If you need to use any other library available on your system, select **Browse** and enter the location of the library.
7. Press **Run**.

---

## 3.6 Installing and executing the IBM POWER Functional Simulator

The IBM Software Development Kit for Linux on Power provides integration with the IBM POWER Functional Simulator, a POWER8 and POWER9 simulator that can be installed in any x86\_64/amd64 system. The simulator instantiates a Power virtual machine to which the x86\_64/amd64 version of SDK can connect. Once connected, you can compile and run ppc64le programs, all in the x86\_64/amd64 machine.

### 3.6.1 Installing and setting up the POWER Functional Simulator

Follow these steps to install, set up, and run the POWER Functional Simulator on your x86\_64/amd64 client.

#### About this task

During the installation process, you will install the simulator and a disk image to boot the virtual machine. You must install the simulator on the same physical system where the x86\_64/amd64 version of the SDK is installed.

The SDK supports a Debian ppc64le image. The image contains the root file system of the standard version of Debian, including system libraries and basic system binaries.

### 3.6.2 Installing the IBM POWER Functional Simulator

If your x86\_64/amd64 machine has Internet access, you can install the IBM POWER Functional Simulator automatically.

#### About this task

You can install and configure the IBM POWER Functional Simulator.

#### Procedure

1. In the SDK UI, select **Power Simulator** > **Start**. The required files are downloaded and extracted in a directory `sdk-systemsim`.
2. When the download and package extraction is complete, go to `/home/<USER>/sdk-systemsim` and run the following command.

```
setupsimulator -i
```

The simulator and its dependencies are downloaded and configured automatically.

#### Results

After the installation completes, you can boot the simulator either through the SDK or through the command line.

---

## 3.7 Running the IBM POWER Functional Simulator

You can run the simulator either in the IBM Software Development Kit for Linux on Power or in the command line.

### 3.7.1 Running the IBM POWER Functional Simulator from within the SDK

You can run the IBM POWER Functional Simulator from within the IBM Software Development Kit for Linux on Power (SDK).

#### About this task

You can run the IBM POWER Functional Simulator by using the SDK.

#### Procedure

1. Select **POWER Simulator** > **Start** in the SDK user interface.
2. Select the simulator version that you would like to start, and then set your user name and password. The password is required to set the iptables rules that allow the simulator to connect to the internet.

**Note:** Your password is encrypted and securely stored within the SDK.

#### Results

After these steps are complete, the simulator starts. An SSH connection to the running instance of the simulator is created and you are automatically connected to its terminal.



### 3.7.1.1 Running the IBM POWER Functional Simulator from within the command line

You can run the IBM POWER Functional Simulator from within the command line.

#### About this task

You can run the IBM POWER Functional Simulator by using the command line.

#### Procedure

1. Locate the `system_execution` directory. The default location for this directory is the user's home location.
2. In this directory, run `startsimulator` using `-p8` or `-p9`, according to the required version of the simulator you want to start.

**Note:** You must have administrator rights and then run this command. You will be prompted to enter the password throughout the process.

#### Results

When the simulator starts, an Xterm is opened in which you can interact with the simulator. The default credentials to access the simulator are: `user=root`; `password=mambo`. The default IP address of the simulator is `172.19.0.109`

---

## 3.8 Simulator networking

The communication between the host and the simulator is carried out through a network interface `tap0`.

#### About this task

Check that the network interface is working.

#### Procedure

1. Run the following command to check that the interface exists and is working:  
`ifconfig tap0`
2. To establish communication between the host and the simulator, rules are added to iptables. Run the following command to check whether those rules are added:

```
sudo iptables --list
```

The output of the previous command must contain the following:

```
Chain FORWARD (policy ACCEPT)
target prot opt source destination
ACCEPT all -- anywhere 172.19.0.0/16
ACCEPT all -- 172.19.0.0/16 anywhere
ACCEPT all -- anywhere 172.19.0.0/16
ACCEPT all -- 172.19.0.0/16 anywhere
```

3. If the iptables rules do not exist, run the following commands to create them:

```
sudo tunc1l -u <YOUR USER> -t tap0
sudo ifconfig tap0 172.19.98.108 netmask 255.255.255.254
sudo iptables -t nat -A POSTROUTING -j MASQUERADE
sudo iptables -I FORWARD -s 172.19.0.0/16 -i tap0 -j ACCEPT
sudo iptables -I FORWARD -d 172.19.0.0/16 -o tap0 -j ACCEPT
sudo sysctl -w net.ipv4.ip_forward=1 $2>&1
```



---

## 4 Managing projects

You can use the IBM Software Development Kit for Linux on Power to create C/C++ projects, or to import existing Makefile or Autotools projects. With the SDK, you can edit the application source code, as well as run and debug the executable program.

The C/C++ Development User Guide ([http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt\\_o\\_home.htm](http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm)) in the Eclipse platform help information contains information about creating Eclipse projects, importing Makefile or Autotools projects, and editing, running, and debugging. Much of this information applies to IBM SDK for Linux on Power projects.

---

### 4.1 Setting flags

Some of the tools included with the SDK require that certain flags be set. This can be done when you create the project, or by editing the project before you run the tools.

#### 4.1.1 Recommended debug, compiler, and linker settings for Power processor tuning

This topic provides recommendations for debug, compiler, and linker settings to use POWER processor capabilities and the IBM Software Development Kit for Linux on Power analysis tools. These recommendations should be considered along with the workload and characteristics of applications that are being built.

Compiler and linker flags are normally passed along all build stages to avoid failures. The flags are passed by setting environment variables such as CFLAGS, CXXFLAGS, CPPFLAGS, and LDFlags for Makefile or Autotools projects in the IBM SDK for Linux on Power. See the following websites for details about the environment variables.

- GCC, the GNU Compiler Collection website (<http://gcc.gnu.org>)
- GNU Operating System - Make website (<http://www.gnu.org/software/make/>)
- GNU Operating System - Autoconf website (<http://www.gnu.org/software/autoconf/>)

#### Recommendations for debug flags

Many analysis and debug tools in the SDK rely on debugging information that is produced when the `-g` flag is enabled. Therefore, it is highly recommended that you set that option by using the CFLAGS or CXXFLAGS environment variables.

- Source Code Analyzer and FDPR

Source Code Analyzer and FDPR tools use information that is produced by the `-Wl,-q` option, which leaves relocation sections and contents in fully linked executable programs. These tools are also able to analyze the post-linked executable program (or library) and to make modifications to improve application performance.

- Gprof

Gprof requires the `-pg` compiler option. This option instruments the resulting binary file to produce profiling information, in the `gmon.out` file, when the executable file or shared library runs.

- Gcov

Gcov requires the `-ftest-coverage` and `-fprofile-arcs` compiler options. These options produce a text file that the `gcov` utility uses to show program coverage, and create a program flow graph for each function of your program and a spanning tree for the graph.

See 4.1.2, “Setting debug flags for Autotools-based projects,” on page 40 for instructions for setting debug flags.

## Recommendations for optimization level options

Optimization level **-O0** is appropriate only for low-level debugging, where every source line must appear to execute sequentially. Other debugging can use optimization levels of **-O1** or **-O2**. Production builds should use a **-O3** optimization level.

If you have installed Advance Toolchain for Linux on Power 9.0 or later, you can use optimization level **-Ofast**. **-Ofast** sets optimization level **-O3** and compiler flag `-ffast-math`.

See Options That Control Optimization(<https://gcc.gnu.org/onlinedocs/gcc-5.3.0/gcc/Optimize-Options.html>) in the GNU GCC documentation for details about the optimization level options.

See 4.1.3, “Setting optimization level flags for Autotools-based projects,” on page 41 for instructions for setting optimization levels.

## Recommendations for compiler flags

Several different compiler flags are recommended for improving performance in applications that are running on Power Systems servers. These compiler flags can be set using the `CFLAGS` or `CXXFLAGS` environment variables.

See IBM RS/6000 and PowerPC Options ([https://gcc.gnu.org/onlinedocs/gcc-5.3.0/gcc/RS\\_002f6000-and-PowerPC-Options.html#RS\\_002f6000-and-PowerPC-Options](https://gcc.gnu.org/onlinedocs/gcc-5.3.0/gcc/RS_002f6000-and-PowerPC-Options.html#RS_002f6000-and-PowerPC-Options)) in the GNU GCC documentation for details about these and other Power specific compiler flags.

Table 6. Recommended compiler flags

Flag	Short description	Extended description	Values
<code>-ffast-math</code>	Enables faster, but non-IEEE 754 compliant math results	Sets <code>-fno-math-errno</code> , <code>-funsafe-math-optimizations</code> , <code>-ffinite-math-only</code> , <code>-fno-rounding-math</code> , <code>-fno-signaling-nans</code> , and <code>-fcx-limited-range</code> options. This option causes the preprocessor macro <code>__FAST_MATH__</code> to be defined.  This option is not enabled by any <code>-O</code> option, because it might result in incorrect output for programs that depend on an exact implementation of IEEE or ISO specifications for math functions. It might, however, yield faster code for programs that do not require the guarantees of these specifications.	
<code>-ffp-contract</code>	Generates code that does or does not use floating point multiply and accumulate instructions	Available with IBM Advance Toolchain for Linux on Power 8.0 and later.  fast: Generates code that uses floating point multiply and accumulate instructions. Equivalent to <code>-mfused-madd</code> .  off: Generates code that does not use floating point multiply and accumulate instructions. Equivalent to <code>-mno-fused-madd</code> .	fast, off, or on
<code>-fpeel-loops</code>	Simplifies loops or splits them into multiple loops to eliminate dependencies	Loop peeling is a special case of loop splitting, which splits any problematic first (or last) few iterations from the loop and performs them outside of the loop body.	

Table 6. Recommended compiler flags (continued)

Flag	Short description	Extended description	Values
-fPIC	Specifies to emit position-independent code (PIC) suitable for use in a shared library	Specifies to emit position-independent code (PIC) suitable for use in a shared library. Also, this option avoids any limit on the size of the global offset table. <b>Note:</b> At present, -fPIC applies only for 32-bit shared libraries. It does not apply for 64-bit.	
-funroll-loops	Unrolls loops, and replicates the body of the loop N times to reduce loop system use and improve scheduling opportunities.	Unrolls loops for which the number of iterations can be determined at compile time or upon entry to the loop. This option makes code larger, and might or might not make it run faster.	
-m32	Generates code for a 32-bit environment.	Generates code for a 32-bit environment.	
-m64	Generates code for a 64-bit environment.	Generates code for a 64-bit environment.	
-maltivec	Generates code that uses AltiVec instructions.	Generates code that uses AltiVec instructions, and also enables the use of built-in functions that allow more direct access to the AltiVec instruction set. You might also need to set -mabi=altivec (using <b>Other POWER-specific flags</b> ) to adjust the current ABI with AltiVec ABI enhancements.  Setting -mcpu=power8 is preferred because it sets -mvsx, -maltivec, and -mabi=altivec.	
-mno-altivec	Generates code that does not use AltiVec instructions.	Generates code that does not use AltiVec instructions.	
-mavoid-indexed-addresses	Generates code that avoids indexed load/store instructions.	Generates code that tries to avoid the use of indexed load or store instructions. These instructions can incur a performance penalty on POWER6 <sup>®</sup> processors in certain situations, such as when a program is stepping through large arrays that cross a 16M boundary. This option is enabled by default when a program is targeting POWER6, and disabled otherwise.	
-mno-avoid-indexed-addresses	Generates code that does not avoid indexed load/store instructions	Generates code that does not try to avoid the use of indexed load or store instructions. These instructions can incur a performance penalty on POWER6 processors in certain situations, such as when a program is stepping through large arrays that cross a 16M boundary. This option is enabled by default when a program is targeting POWER6, and disabled otherwise.	

Table 6. Recommended compiler flags (continued)

Flag	Short description	Extended description	Values
<code>-mmodel=model</code>	Sets code model	<p>small: Generates PowerPC® 64-bit code for the small model. The TOC is limited to 64 K in size.</p> <p>medium: Generates PowerPC 64-bit code for the medium model. The TOC and other static data is limited to 4G in size.</p> <p>large: Generates PowerPC 64-bit code for the large model. The TOC is limited to 4G in size. Other data and code is limited only by the 64-bit address space.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• Most applications should use <code>-mmodel=medium</code>.</li> <li>• If you are using <code>-mmodel=medium</code> or <code>-mmodel=large</code>, you should also remove the <code>-mminimal-toc</code> flag, if it is set.</li> </ul>	small, medium, or large
<code>-mcpu=cpu_type</code>	Sets machine type parameters	Sets architecture type, register usage, choice of mnemonics, and instruction scheduling parameters for machine type <code>cpu_type</code> .	power8 or any Power CPU type
<code>-mrecip=option</code>	Controls which type of reciprocal estimate instructions can be used	<p>Controls which reciprocal estimate instructions can be used. <i>option</i> is a comma-separated list of options, which can be preceded by a “!” to invert the option.</p> <p><b>all</b> enable all estimate instructions</p> <p><b>default</b> enable the default instructions, equivalent to <code>-m-recip</code></p> <p><b>none</b> disable all estimate instructions, equivalent to <code>-mno-recip</code></p> <p><b>div</b> enable the reciprocal approximation instructions for both single and double precision</p> <p><b>divf</b> enable the single precision reciprocal approximation instructions</p> <p><b>divd</b> enable the double precision reciprocal approximation instructions</p> <p><b>rsqrt</b> enable the reciprocal square root approximation instructions for both single and double precision</p> <p><b>rsqrtf</b> enable the single precision reciprocal square root approximation instructions</p> <p><b>rsqrtd</b> enable the double precision reciprocal square root approximation instructions</p> <p>For example, <code>-mrecip=all,!rsqrtd</code> enables all reciprocal estimate instructions except for <code>FRSQRTE</code>, <code>XRSQRTEDP</code>, and <code>XVRSQRTEDP</code> instructions, which handle the double precision reciprocal square root calculations. <code>-ffast-math</code> must be enabled to use this option.</p>	all, default, none, div, divf, divd, rsqrt, rsqrtf, or rsqrtd

Table 6. Recommended compiler flags (continued)

Flag	Short description	Extended description	Values
-mrecip-precision	Specifies to assume that the reciprocal estimate instructions have higher precision than needed by the ABI	Specifies to assume that the reciprocal estimate instructions provide higher precision estimates than is mandated by the POWERPC ABI. Selecting <code>-mcpu=power8</code> automatically selects <code>-mrecip-precision</code> . The double precision square root estimate instructions are not generated by default on low precision machines, because they do not provide an estimate that converges after three steps. <code>-ffast-math</code> must be enabled to use this option.	
-mno-recip-precision	Specifies to not assume that the reciprocal estimate instructions have higher precision than needed by the ABI	Specifies to not assume that the reciprocal estimate instructions provide higher precision estimates than is mandated by the POWERPC ABI. <code>-ffast-math</code> must be enabled to use this option.	
-mtune= <i>cpu_type</i>	Sets machine tuning parameters	Sets the instruction scheduling parameters for machine type <i>cpu_type</i> , but does not set the architecture type, register usage, or choice of mnemonics as <code>-mcpu=<i>cpu_type</i></code> would. The same values for <i>cpu_type</i> are used for <code>-mtune</code> as for <code>-mcpu</code> . If both are specified, the code that is generated uses the architecture, registers, and mnemonics that are set by <code>-mcpu</code> , but the scheduling parameters that are set by <code>-mtune</code> .	power8 or any Power CPU type
-mupdate	Generates code that updates the base register	Generates code that uses the load or store instructions that update the base register to the address of the calculated memory location. These instructions are generated by default.	
-mno-update	Generates code that does not update the base register	Generates code that does not use the load or store instructions that update the base register to the address of the calculated memory location. These instructions are generated by default.	
-mveclibabi=MASS	Specifies type of ABI to use for the vectorizing intrinsics	Specifies the ABI type to use for vectorizing intrinsics by using an external library. The only type supported currently is MASS, which specifies to use IBM Mathematical Acceleration Subsystem (MASS) libraries for vectorizing intrinsics by using external libraries when generating code for POWER7®.  Flags <code>-ftree-vectorize</code> , <code>-funsafe-math-optimizations</code> , and <code>-ffast-math</code> must be enabled. The MASS libraries must be specified at link time.	
-mvsx	Generates code that uses the vector/scalar instructions	Generates code that uses vector/scalar (VSX) instructions, and also enables the use of built-in functions that allow more direct access to the VSX instruction set.  Setting <code>-mcpu=power8</code> is preferred because it sets <code>-mvsx</code> , <code>-maltivec</code> , and <code>-mabi=altivec</code> .	
-mno-vsx	Generates code that does not use the vector/scalar instructions	Generates code that does not use the vector/scalar (VSX) instructions.	

Table 6. Recommended compiler flags (continued)

Flag	Short description	Extended description	Values
-mpower8-fusion	Specifies to fuse an integer load with a preceding <b>addis</b> instruction and to fuse a vector load with a preceding <b>addi</b> instruction.	Specifies to fuse an integer load with a preceding <b>addis</b> instruction and to fuse a vector load with a preceding <b>addi</b> instruction. Setting <code>-mtune=power8</code> specifies this also.	
-mpower8-vector	Enables POWER8 vector instructions.	Enables POWER8 vector instructions. This option requires that <code>-mvsx</code> also are set.	
-mcrypto	Enables POWER8 cryptographic built-in functions.	Enables POWER8 cryptographic built-in functions. This option requires that <code>-maltivec</code> also are set.	
-mdirect-move	Enables POWER8 moves between general purpose registers and vector registers.	Enables POWER8 moves between general purpose registers and vector registers. This option requires that <code>-mvsx</code> also are set.	
-mqquad-memory	Enables quadword memory instructions, including quadword atomic instructions.	Enables quadword memory instructions, including quadword atomic instructions. This option requires that <code>-m64</code> also are set.	

## Recommendations for linker flags

The `-Bsymbolic` flag enables the linker to bind global symbol references to local definitions within the shared library. Normally this binding is deferred until program load time. At load time, global symbol references are bound to the definitions that are loaded first, starting with the main program. Early binding to local definitions reduces processor usage during calls, but also disables symbol overrides using preloaded libraries.

The `-m32` and `-m64` flags enable the linker to generate code for 32-bit and 64-bit environments. In most Makefile and Autotools projects, these flags can be passed along all build stages by setting them using the `LDFLAGS` environment variable.

### 4.1.2 Setting debug flags for Autotools-based projects

This topic details the steps for setting debug flags to use POWER processor capabilities and the IBM Software Development Kit for Linux on Power analysis tools, Source Code Analyzer and FDPR, `gprof`, and `gcov` in Autotools-based projects.

#### Before you begin

**Note:** If your project was created by importing a Makefile project, you must manually edit the Makefile file to export build control variables such as `CFLAGS` and `LDFLAGS` with debug flags.

See “Recommendations for debug flags” on page 35 for details on the recommended settings.

#### About this task

To set debug flags for projects that were created by importing Autotools projects, complete the following steps in the IBM SDK for Linux on Power user interface.



## Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Right-click the project name and click **Properties**.
3. Expand **Autotools** and click **Configure Settings**.
4. Click **Advanced**.
5. Choose from the following steps to set debug flags for particular tools.
  - For gcov and gprof, complete the following steps:
    - a. In the right pane, select the **Debug (-g)** check box.
    - b. In the right pane, select the **Gprof support (-pg)** or **Gcov support (-fprofile-arcs-ftest-coverage)** check box, as appropriate.
  - For Source Code Analyzer and FDP, complete the following steps:
    - a. In the right pane, ensure that the **Debug (-g)**, **Gprof support (-pg)**, and **Gcov support (-fprofile-arcs-ftest-coverage)** check boxes are not selected. If the source code is compiled with these options, Source Code Analyzer and FDP cannot work properly.
    - b. In the **Additional command-line options** field, set the value of CFLAGS (C) or CXXFLAGS (C++) to -g -Wl,-q. For example, type the following in the field:  
`CFLAGS="-g -Wl,-q"`
6. Click **Apply**.

### 4.1.3 Setting optimization level flags for Autotools-based projects

This topic details the steps for setting optimization levels for debugging with IBM Software Development Kit for Linux on Power in Autotools-based projects.

#### Before you begin

**Note:** If your project was created by importing a Makefile project, you must manually edit the Makefile file to export build control variables such as CFLAGS and LDFLAGS with debug flags.

See “Recommendations for optimization level options” on page 36 for details on the recommended settings.

#### About this task

Complete the following steps in the IBM SDK for Linux on Power user interface.

## Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Right-click the project name and click **Properties**.
3. Expand **Autotools** and click **Configure Settings**.
4. Click **Advanced**.
5. In the right pane, ensure that the **Debug (-g)**, **Gprof support (-pg)**, and **Gcov support (-fprofile-arcs-ftest-coverage)** check boxes are not selected. If you need any of these flags to be set at build time, you must re-export them together with optimization level flags as described in the next step.
6. In the **Additional command-line options** field, set the value of CFLAGS (C) or CXXFLAGS (C++) with the appropriate optimization level, such as -O3 or -Ofast. For example, type the following in the field:  
`CFLAGS="-g -O3"`
7. Click **Apply**.

---

## 4.2 Editing a project

After you have imported or created a project, you can use the IBM Software Development Kit for Linux on Power graphical user interface to edit project preferences, build options, and application source code for the project.

The C/C++ Development User Guide ([http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt\\_o\\_home.htm](http://help.eclipse.org/neon/topic/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm)) in the Eclipse platform help information contains the information you need to edit projects using IBM SDK for Linux on Power.

Most project preferences and build options you might want to customize are documented in the C/C++ Development User Guide. See 4.1.1, “Recommended debug, compiler, and linker settings for Power processor tuning,” on page 35 for information about setting options for Power Systems.

### 4.2.1 Setting the Linux tools path

A customization that is unique to IBM SDK for Linux on Power is setting the Linux tools path. The Linux tools path is set automatically when creating a project, but you can change it. See 2.4.7, “Changing the Advance Toolchain version,” on page 17 for information about this customization.

### 4.2.2 Using the coding assistant

IBM SDK for Linux on Power provides a coding assistant that helps with code completion, function templates, and context-sensitive information for libauxv and libsphde libraries and the altivec API when using the C/C++ editor. While you are editing your code, you can use the following methods to display information:

- Move your mouse over a function.
- Type the initial letters, press **Ctrl+Spacebar**, and select a listed function.

---

## 4.3 Building a project

After a project is created or imported, you can select to build it or perform a clean build.

The C/C++ Development User Guide ([http://help.eclipse.org/mars/topic/org.eclipse.cdt.doc.user/concepts/cdt\\_o\\_home.htm](http://help.eclipse.org/mars/topic/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm)) in the Eclipse platform help information contains additional information about building projects.

### 4.3.1 Building a project

This topic describes a basic project build.

#### About this task

To build, or compile, a project, complete the following steps.

#### Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Click **Project > Build Project** to build the project.
3. While the project is building, you can click the **Console** tab to monitor progress.

### 4.3.2 Building a project with clean build

A clean build discards previous build status information and problem markers, and causes the project to be rebuilt from the beginning.

## About this task

Complete the following steps.

### Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Click **Project > Clean**.
3. In the Clean window, click the **Clean projects specified below** radio button.
4. Select the check box for the project you want to clean. Click **OK**.
5. While the project is building, you can click the **Console** tab to monitor progress.

### 4.3.3 Adding and using a Make target

For Makefile and Autotools projects, you can add Make targets, for transforming the source file to a specific target result. Example Make targets include “clean”, “check”, and “install”.

## About this task

**Note:** Adding Make targets is intended for advanced users.

Complete the following steps.

### Procedure

1. View the project by clicking the project name in the Project Explorer pane.
2. Right-click the project name and select one of the following, as appropriate:
  - **Make Target > Create** to add a new Make target to your project. Click **OK** after specifying the information for the new target.
  - **Make Target > Build > Target name** to select a specific Make target. Click **Build** build the selected target.



---

## 5 Migrating an application to Power Systems servers using Migration Advisor

The IBM Software Development Kit for Linux on Power includes the Migration Advisor to help in moving Linux applications to Power Systems servers. The Migration Advisor uses the Eclipse CDT (C/C++ Development Tools) code analysis tool (CODAN). The code analysis tool locates potential migration problems within a project, such as source code that might produce different results when run on Power Systems servers.

---

### 5.1 Running Migration Advisor

You can use the IBM Software Development Kit for Linux on Power Migration Advisor to help you migrating your application to a Power Systems server.

#### About this task

To run Migration Advisor, click a C/C++ project in the Project Explorer pane, right-click the project name, and select **Run Migration Advisor**. The Migration Advisor Wizard displays. See 5.2, “Using the Migration Wizard to configure the Migration Advisor” for information on how to configure the Migration Advisor by using the wizard.

When the migration process is complete, the Migration Advisor View opens and all potential migration issues are displayed as warnings.

The following are actions you can take using Migration Advisor and the Migration Advisor View.

#### Procedure

- To display the code that has a reported potential issue, double-click the warning and the correct file is loaded in the editor view.
- You can continue to type in the editor window while the Migration Advisor is running. Because Migration Advisor is a live tool, the warnings are updated automatically as you edit.
- To ignore an issue, right-click the warning and select the **Ignore Warning** option. The warning no longer displayed.
- To display ignored warnings again, click the menu arrow in the upper right of the Migration Advisor View, and select one of the following options.
  - Click **Disable Ignored Warnings** to display all ignored warnings. This option displays the ignored warnings, but it still retains the information that they are ignored.
  - Click **Enable Ignored Warnings** to hide all ignored warnings.
  - Click **Forget All Ignored Warnings** to stop ignoring all warnings.

**Note:** You cannot undo this option, so ensure that you want to stop ignoring all warnings before you select it.

---

### 5.2 Using the Migration Wizard to configure the Migration Advisor

You can use the automated wizard to configure the Migration Advisor.

#### About this task

To migrate your source code, you must provide some details about the origins of your C/C++ Linux application and the target system you want to migrate.

Migration Advisor supports two different target systems (ppc64be and ppc64le) and four source systems (x86, AMD64, ppc32be and ppc32le). Depending on the origin of your source code and what the target processor is, Migration Advisor optimizes its configuration to achieve the best migration result.

Every time you run the Migration Advisor in a new project, the Migration Wizard is shown.

To configure Migration Advisor, complete the following steps.

## Procedure

1. Click **Next** to open the configuration screen.
2. Select an appropriate migrating setup. See 5.2.1, “Basic Migration Advisor configuration” and 5.2.2, “Advanced Migration Advisor configuration” to help you decide the best configuration for your application.
3. On the configuration screen, click **Next** to review the setup. The review screen shows all selected checkers, providing a fine grained adjustment. On this screen, you can disable or enable a checker manually and see what each checker does. If you select the **Always use this configuration for project <your\_project>** option every time you press **Run Migration Advisor** in the Project Explorer, the Migration Advisor runs automatically.
4. Click **Finish**.

### 5.2.1 Basic Migration Advisor configuration

This topic describes the basic configuration options for the IBM Software Development Kit for Linux on Power Migration Advisor.

The following basic configuration options are available in the Migration Advisor.

#### Select your project

If you run the Migration Advisor by selecting a project in the Project Explorer pane, this field is automatically configured. Otherwise, select a valid C/C++ project in the list.

#### Your Linux application was designed to run on

In the Migration Advisor, you must select the machine architecture that your application is coming from. Select one of the supported source systems: x86, AMD64, PPC32BE, or PPC32LE. If you are not sure, hover over the button to view more information.

#### You want to migrate your Linux application to

Select the target system for the migration. Select PPC64LE to migrate to a little endian Linux system, or PPC64BE to migrate to a big endian Linux system.

### 5.2.2 Advanced Migration Advisor configuration

This topic describes the advanced configuration options for the IBM Software Development Kit for Linux on Power Migration Advisor.

The following advanced configuration options are available in the Migration Advisor.

#### Apply basic fixes automatically

Select this option if you want the Migration Advisor to migrate your application automatically. Selecting this option applies fixes during the code analysis step, reducing the effort to apply them manually. If this option is activated, the Migration Advisor also generates a log with all changes in the project so that the developer can review the fixes.

#### Enable POWER8 optimization support

This feature enables target specific optimizations. Select this option if you are migrating your application to a POWER8 processor.

### Enable checkers in blocks inactivated by the preprocessor and macro definitions

If you select this option, Migration Advisor analyzes inactive blocks. For example, if this option is not selected, Migration Advisor would not look for problems in this sample code:

1. `#if 0`
2. `//Code with problems`
3. `#endif`

**Note:** If this option is enabled, the Migration Advisor might suggest fixes based on a code that is different from the compiled version.

### Report problems in statements that come from macro expansion

This option enables Migration Advisor to flag problems in macro expansions. For example, Migration Advisor displays a warning for the macro definition, which is shown in line 1 of the following sample code only if this option is not selected. Conversely, Migration Advisor would display a warning on line 2 of the following code if this option is selected.

1. `#define MACRO asm("asm_with_errors");`
2. `MACRO`

### Enable Scalability mode in Migration Advisor

Inactive blocks in large files might not be analyzed if this option is enabled. If this option is disabled, blocks from large files are analyzed and a large amount of memory is used.

---

## 5.3 Migration Advisor checkers

These topics describe the checkers used by the Migration Advisor.

**Note:** By using the code examples, you agree to the terms of the 10, “Code license and disclaimer information,” on page 95.

### x86-specific compiler built-in checker

Some x86 compiler built-ins are not available for 32-bit POWER or 64-bit POWER architectures. This checker finds all occurrences of x86-specific built-ins.

#### Example

```
void foo() {
    __builtin_infq(); //x86-specific built-in
}
```

### x86-specific assembly checker

Inline assembly code usually cannot be migrated without problems. Therefore, warnings are displayed for inline assembly code.

#### Example

```
void foo() {
    asm("mov %ax, 0"); //assembly code
    __asm__("mov %ax, 0"); //assembly code
}
```

### Struct with bitfields checker

x86 and POWER architectures have different endianness, which refers to the ordering of separately addressable components. Because of this, the order of bit fields in a struct is different in a Power Systems server, which can result in migration issues. It is important to ensure that bit fields are used correctly to avoid problems.

It is also a good idea to see where the struct is referenced to check whether there are endianness issues. To do this, you can navigate in struct declarations and references.

- To navigate in struct references in your project, select the object name, right-click, and select **References** > **Project**. All references display in the Search View. Double-click the file names in the Search View to open them in the Editor View.
- To navigate in struct declarations, select the struct name, right-click, and select **Declarations** > **Project**. The results also display in the Search View.
- Declarations can contain typedefs. Be sure to also check all references of the typedefs.

## Examples

The following example would also be flagged by the cast with endianness issues checker.

```
#include <stdio.h>
struct _my_struct {
int a:4;
int b:4;
} my_struct;

void foo() {
char my_char;
my_struct.a = 0xFF;
my_struct.b = 0x00;
my_char = *(char *)&my_struct;
//This prints 0x0f in x86 (or x86_64) machines and 0xf0 in ppc (or ppc64) machines.
printf("%x\n", my_char);
}
```

The following example would also cause endianness problems if it is changed to use the network instead of a file to read and write data.

```
#include
#include
#include

void foo() {
int file;
char my_char;
struct _my_struct {
int a:4;
int b:4;
} my_struct;

//initial values
my_struct.a = 0x1;
my_struct.b = 0x2;

//Writing to a file
file = open("my_file", O_CREAT | O_WRONLY);
write(file, &my_struct, 1);
close(file);

//Reading from a file
file = open("my_file", O_RDONLY);
read(file, &my_char, 1);
close(file);
printf("%x\n", my_char);
}
```



## Cast with endianness issues checker

Casting can create endianness problems in C or C++ code. This happens when casting pointers for types with different sizes. The cast with endianness issues checker analyzes all casts in the code and flags the casts that can cause an endianness problem.

### Example

```
void foo() {
    short int val = 0xFF00;
    char *char_pointer = (char *) &val;
    //This prints 0 in x86 (or x86_64) machines and ff in ppc (or ppc64) machines.
    printf("%x\n", *char_pointer);
}
```

## Union with endianness issues checker

Using the several fields from a union in the same object can cause endianness issues. The union with endianness issues checker analyzes all unions in the code and flags the unions that can cause endianness problems.

It is a good idea to see where the struct is referenced to check whether there are endianness issues. To do this, you can navigate in union declarations and references.

- To navigate in union references in your project, select the object name, right-click, and select **References > Project**. All references display in the Search View. Double-click the file names in the Search View to open them in the Editor View.
- To navigate in union declarations, select the union name, right-click, and select **Declarations > Project**. The results also display in the Search View.
- Declarations can contain typedefs. Be sure to also check all references of the typedefs.

### Example

```
void foo() {
    union {
        short int val1;
        char val2;
    } u;
    u.val1 = 0xFF00;
    //This prints 0 in x86 (or x86_64) machines and ff in POWER 32-bit (or 64-bit) machines.
    printf("%x\n", u.val2);
}
```

## Long double usage checker

The use of long double might be a migration problem because of the differences in size and format between x86 and POWER architecture. With current GCC compilers, long double is 128 bits (AIX® double double format) for POWER architecture. Therefore, long double for POWER architecture is in a different format than long double for x86 ILP32 (96 bits) or x86 ILP64 (binary 128).

## Performance degradation checker

It is usual to see performance improvements made for one specific architecture in an area of the code. This is mainly done using a preprocessor `#if` statement to check the current architecture. The performance degradation checker looks for `#if` and `#ifdef` statements that contain preprocessor definitions belonging to x86 or x86\_64 architectures, but do not contain preprocessor definitions for POWER architectures. This is a sign that the code might not be optimized for POWER processors.

The performance degradation will also try to make the best attempt to convert the code inside the x86 block to PPC. If the code has a specific x86 built-in, the conversion, in some cases will be automatically. For example:

```
#ifdef __x86__
__m128 ra = _mm_load_ps(&a + i);
__m128 rb = _mm_load_ps(&b + i);
_mm_store_ps(&c + i, _mm_add_ps(ra,rb));
#else
c[i] = a[i] + b[i];
c[i + 1] = a[i + 1] + b[i + 1];
c[i + 2] = a[i + 2] + b[i + 2];
c[i + 3] = a[i + 3] + b[i + 3];
#endif
```

The previous code example has an x86 block. The MA will migrate all Intel built-ins to PPC. After the fix, the code will look similar to the following version:

```
#ifdef __x86__
__m128 ra = _mm_load_ps(a + i);
__m128 rb = _mm_load_ps(b + i);
_mm_store_ps(c + i, _mm_add_ps(ra,rb));
#elif defined __PPC__
// TODO: write an optimized code for Power
__vector float ra = *(__vector float*) (a + i);
__vector float rb = *(__vector float*) (b + i);
*(__vector float*) (c + i) = vec_add(ra, rb);
#else
c[i] = a[i] + b[i];
c[i + 1] = a[i + 1] + b[i + 1];
c[i + 2] = a[i + 2] + b[i + 2];
c[i + 3] = a[i + 3] + b[i + 3];
#endif
```

You can change the preprocessor definitions used by the performance degradation checker. For information, see 5.3.4, “Customizing the performance degradation checker,” on page 54.

## Example

```
#if defined(__x86__)
//code specific for x86 architectures
#else
//code that will work for all architectures, but it isn't optimized.
#endif
```

## Syscall not available for Linux on Power checker

Almost all x86 and x86\_64 Linux system calls (syscalls) are available for POWER architecture. Using an x86-specific system call causes problems when migrating your code to POWER architecture. The syscall not available for Linux on Power checker analyzes all system calls in your code and displays a warning for system calls that are not available for 32-bit or 64-bit POWER architectures. The warning message states whether the system call is unavailable only for 32-bit POWER, only for 64-bit POWER, or unavailable for both architectures.

You can specify that this checker look for only those system calls that are not available for 32-bit or 64-bit POWER architectures. For information, see 5.3.5, “Customizing the syscall not available for Linux on Power checker,” on page 55.

## Example

```
int main() {
    int rc;
    struct stat64 s;
    char *file = "test_file.txt";
```

```

if ((rc = syscall(SYS_stat64, file, &s)) == -1) {
    printf("Failed to execute syscall with rc = %d (%s)", errno, strerror(errno));
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}

```

## Linux/x86-specific API checker

The Intel-specific API checker looks for usage of functions from the following libraries:

- Intel Integrated Performance Primitives 7.0
- Intel Math Kernel Library 10.3
- Message Passing Interface Library 2.2
- Decimal Floating-Point Math Library 2.0

You can customize the Linux/x86-specific API checker by selecting which library APIs the checker looks for. In addition, you can add, edit, and remove APIs from the checker. For information, see 5.3.6, “Customizing the Linux/x86-specific API checker,” on page 55.

## Hardware Transaction Memory checker

This checker finds occurrences of x86 hardware transaction memory usage, which is not supported for POWER architectures. Transactional Synchronization Extensions (TSX), an x86 instruction set extension, enables hardware transaction memory support to speed up the execution of multithreaded programs.

```

#include <rtmintrin.h>

void foo(unsigned int a, unsigned int b){
    while(1){
        unsigned status = _xbegin();
        if(status == XBEGIN_STARTED) {
            trans_func(a, b);
            _xend();
            break;
        }
        else{
            _xabort();
        }
    }
}

```

## Non-portable Pthreads implementation checker

Some Pthreads functions and data types are not supported on POWER architectures. This checker finds occurrences of non-portable Pthreads API usage.

```

#include <pthread.h>
void foo(){
    pthread_id_np_t tid;
    tid = pthread_getthreadid_np();
}

```

### 5.3.1 Linux/x86 to Linux/Power application migration checker

This topic describes the Linux/x86 to Linux/Power application migration checker.

#### Char usage checker

The use of char might be a problem when assigning a numeric value to a variable of type char. By default, the char type is signed in x86 but it is unsigned in POWER architecture. The safe mode to use a char that receives a numeric value is to declare it as signed or unsigned.

## Example

```
void foo() {
    char c = -1;
    int x = (int) c;
    // This prints -1 in x86 machines and 255 in ppc machines.
    printf("x = %d\n", x);
}
```

## 5.3.2 Linux/Power application optimization checker

This topic describes the Linux/Power application optimization checker.

### Sync built-in to C11 standard atomic built-in checker

Sync built-ins are defined for IA64 and adopted for general use by GCC. However, it does not have a good performance for POWER architecture. Instead of using Sync, use the Atomics built-ins that are intended to replace the legacy Sync built-ins.

**Note:** The Atomic built-ins assume that programs conform to the C++11 memory model.

## Example

```
void foo() {
    int x = 20;
    int y = 80;
    int result = 0;
    result = __sync_fetch_and_add(&x, y);
}
```

## 5.3.3 Linux/Power 32bit to Linux/Power 64bit application migration checkers

These topics describe the checkers used for Linux/Power 32bit to Linux/Power 64bit application migration.

### Long usage checker

When you are migrating an application from ILP32 to LP64, you must take care about the difference in size of some types. The long type for example, has 32 bit in ILP32 and 64 in LP64. Therefore, this checker searches for long declarations and warns you about possible problems.

### Example - unsafe code

```
long x = 0xFFFFFFFF;
cout << x << endl;
```

### Example - fixed code

```
int x = 0xFFFFFFFF;
cout << x << endl;
```

### Implicit Cast Checker

This checker performs a deep static analysis of the code, searching for patterns where an implicit cast could cause an invalid result depending on the value generated by a variable or expression.

### Example - unsafe code

For example, a downcast from unsigned int type to long type can generate a different result in LP64 power computer.

```

unsigned int a = 0;
  long b = --a;
  cout << b << endl;

  // Output ppc32 -> -1
  // Output ppc64 -> 4294967295

```

### Example - fixed code

```

unsigned int a = 0;
  int b = --a;
  cout << b << endl;

```

### Example - unsafe code

Moreover, some libraries might have a different type declaration depending on the system environment. Therefore, this checker is designed to also detect cast between types declared in libraries and macros.

```

#include <stdint.h>
  intptr_t a = -1;
  uintptr_t b = a; // dangerous implicit cast.

```

### Example - fixed code

```

#include <stdint.h>
  int a = -1;
  unsigned int b = a;

```

## Pointer Cast Checker

This checker is an extension of the implicit cast checker and also performs a static analysis, searching for problems related to explicit casts. Moreover, it reports cast between pointers with different sizes. For example, if you declare a pointer and force a cast to another pointer of a different type, it might result in an unexpected behavior.

### Example - unsafe code

```

const char *a = "abcdefghj";
  long *b = (long *) a;
  cout << *b << endl;

```

### Example - fixed code

```

const char *a = "abcdefghj";
  int *b = (int *) a;
  cout << *b << endl;

```

## Heterogeneous Type checker

This checker searches for structs, unions, and classes and warns you about incompatible declarations inside those data types. For example, if you declare a long variable inside a heterogeneous data structure, you can face a variety of problems such as, different result due to a memory overlap, different data alignment, endianness incompatibility or difference in size. The code below shows a memory overlap problem:

### Example - unsafe code

```

union Pixel {
  unsigned long color;
  unsigned char rgba[4];
};

```

## Example - fixed code

```
union Pixel {
    unsigned int color; // on uint32_t color;
    unsigned char rgba[4];
};
```

## Format Specifiers Checker

A format specifier is a symbol used in ANSI C/C++ to format input/output strings. The table below summarizes all valid C/C++ combinations. For more information, see PRINTF.

	d or i	u, o, x or X	f, F, e, E, g, G, a or A	c	s	p	n
-	int	unsigned int	double	unsigned char	char*	void*	int*
h	short	unsigned short	double				short*
hh	char	unsigned char	double				char*
l	long	unsigned long	double	wint_t	wchar_t*		long*
ll	long long	unsigned long long	double				long long*
j	intmax_t	uintmax_t	double				intmax_t*
z	size_t	size_t	double				size_t*
t	ptrdiff_t	ptrdiff_t	double				ptrdiff_t*
L	long	unsigned long	long double		char		

The format specifier usage is a common problem because, C/C++ performs type promotion and downcast depending on the type passed as parameter. Therefore, this checker searches for variables that are not compatible with the format specifier. It also reports the usage of types that changes its size in an LP64 programming model.

## Example - unsafe code

```
int x;
scanf("%ld",&x);
```

## Example - fixed code

```
int x;
scanf("%d",&x);
```

## 5.3.4 Customizing the performance degradation checker

You can change the preprocessor definitions used by the performance degradation checker.

### About this task

To change the definitions, complete the following steps.

#### Procedure

1. Click **Window > Preferences**.
2. In the left pane, expand **C/C++** and click **Code Analysis**. The Code Analysis window displays all the installed Migration Advisor checkers.
3. Locate the group of checkers provided by the SDK: **Linux/x86 to Linux on Power, Linux/Power 32 bits to Linux/Power 64 bits application migration**, or **Linux/Power application optimization** in the window, by scrolling if needed. Expand this group.
4. Select the **Performance Degradation** checker and click **Customize > Customize Selected**.
5. Edit preferences as needed.
  - Add or remove the predefined macros for Power and non-Power architectures.
  - Select or clear the check box for **Case sensitive when comparing preprocessor definitions**.

Click **OK** when finished.

### 5.3.5 Customizing the syscall not available for Linux on Power checker

You can specify that this checker look for only those system calls that are not available for 32-bit or 64-bit POWER architectures.

#### About this task

To select this option, complete the following steps.

#### Procedure

1. Click **Window > Preferences**.
2. In the left pane, expand **C/C++** and click **Code Analysis**. The Code Analysis window displays all the installed Migration Advisor checkers.
3. Locate the group of checkers provided by the SDK: **Linux/x86 to Linux on Power**, **Linux/Power 32 bits to Linux/Power 64 bits application migration**, or **Linux/Power application optimization** in the window, by scrolling if needed. Expand this group.
4. Select the **Syscall not available for Linux on Power** checker and click **Customize > Customize Selected**.
5. Edit the preferences to select the check box for **Check for syscalls only unavailable for 32-bit Power architecture**, **Check for syscalls only unavailable for 64-bit Power architecture**, or both. Click **OK** when finished.
6. Click **Apply**.

### 5.3.6 Customizing the Linux/x86-specific API checker

You can customize the Linux/x86-specific API checker by selecting which library APIs the checker looks for. In addition, you can add, edit, and remove APIs from the checker.

#### About this task

To customize the checker, complete the following steps.

#### Procedure

1. Click **Window > Preferences**.
2. In the left pane, expand **C/C++ > Code Analysis > Migration Advisor**.
3. In the Migration Advisor window, click **Customize the API Checker**. The API customization window is displayed.
4. In the API customization window, choose from the following options:
  - Select the check box for each API that you want the API checker to look for. Clear the check box for each API that you do not want the checker to look for.
  - To add an API to the checker, click **Add API**. Type the name, optionally type a description, and click **OK**.
  - To edit an API, click the API name to select it, then click **Edit API**. Type a different name or description, and click **OK**.
  - To add a function to an API, click the API name to select it, then click **Add function**. Type the function name, and click **OK**. To display the new function name if it is not already displayed, expand the API name in the API customization window. You can add multiple functions to an API.
  - To add a type to an API, click the API name to select it, then click **Add type**. Type the type name, and click **OK**. To display the new type name if it is not already displayed, expand the API name in the API customization window. You can add multiple types to an API.
  - To add a function from a file, click the API name to select it, then click **Add function from a file**. Type the file name or click **Browse** to locate and select it.

**Note:** The file must contain only the function names, with one name per line.

After you specify the file, click **OK**. All the functions that are listed in the specified file are added and displayed.

- To add a type from a file, click the API name to select it, then click **Add type from a file**. Type the file name or click **Browse** to locate and select it.

**Note:** The file must contain only the type names, with only one name per line.

After you specify the file, click **OK**. All the types that are listed in the specified file are added and displayed.

- To remove an API from the checker, click the API name to select it, then click **Remove**. When prompted, click **OK** to confirm.
- To export an API and any functions or types that it contains to an XML file to be imported later, click the API name to select it. Then, click **Export to a XML file**. Type the file name, including file type xml, or click **Browse** to locate and select it. Then, click **OK**.
- To import an API and any functions or types that it contains from an existing XML file, click the API name to select it, then click **Import a XML file**. Type the file name, including file type xml, or click **Browse** to locate and select it. Then, click **OK**.
- To restore the API checker to the default list of APIs included in the Migration Advisor, click **Restore Defaults**.

**Note:** All customization changes will be lost.

5. When finished customizing, click **OK** to save your changes.

---

## 5.4 Configuring Migration Advisor checkers manually

The Migration Advisor contains several checkers that look for code in the project that might produce a different result in Power Systems servers. Warnings are displayed showing the kind of problem found.

### About this task

To enable Migration Advisor checkers manually, complete the following steps.

#### Procedure

1. Click **Window > Preferences**.
2. In the left pane, expand **C/C++** and click **Code Analysis**. The Code Analysis window displays all the installed Migration Advisor checkers.
3. Locate the group of checkers that are provided by the SDK. Expand this group.
  - Linux/x86 to Linux on Power
  - Linux/Power 32-bits to Linux/Power 64-bits application migration
  - Linux/Power application optimization
4. Select the check box beside the name of the group to select all the checkers, or click the check boxes for the specific checkers that you want to select.
5. Click **Apply**.

---

## 5.5 Using Migration Advisor quick fixes

Quick fixes can help you resolve migration problems found by running the Migration Advisor. Quick fixes are suggestions or tips that might help correct identified migration issues.



## About this task

Migration Advisor quick fixes are available only for the following checkers:

- x86-specific compiler built-in checker. See “x86-specific compiler built-in checker” on page 47.
- Struct with bitfields checker. See “Struct with bitfields checker” on page 47.
- Performance degradation checker. See “Performance degradation checker” on page 49.
- Decimal Floating-Point Math Library as part of Linux/x86-specific API checker. See “Linux/x86-specific API checker” on page 51.
- Hardware Transaction Memory checker. See “Hardware Transaction Memory checker” on page 51.
- Long usage checker. See “Long usage checker” on page 52.
- Char usage checker. See “Char usage checker” on page 51.
- Implicit cast checker. See “Implicit Cast Checker” on page 52.
- Pointer cast checker. See “Pointer Cast Checker” on page 53.
- Heterogeneous type checker. See “Heterogeneous Type checker” on page 53.
- Sync built-in to C11 standard atomic built-in checker. See “Sync built-in to C11 standard atomic built-in checker” on page 52.
- Syscall not available for Linux on Power checker. See “Syscall not available for Linux on Power checker” on page 50.



---

## 6 Analyzing application performance on Power Systems servers

IBM Software Development Kit for Linux on Power provides many performance analysis tools.

---

### 6.1 Analyzing performance with the CPI breakdown plug-in

The CPI breakdown plug-in profiles C/C++ applications with the CPI (cycles per instruction) breakdown model for POWER8 and POWER7 systems.

If you want to use the CPI Breakdown plug-in, you must download and install Perf as described in 2.2, “Recommended and optional packages,” on page 9.

#### 6.1.1 CPI analysis overview

CPI (cycles per instruction) analysis can be used to improve application performance.

CPI refers to how many processor cycles are needed to complete an instruction. An instruction can be a read/write from memory operation, an arithmetic calculation, or bit-wise operation. The more cycles the processor takes to complete an instruction, the poorer the performance of the application in the processor.

Application performance can be improved by decreasing the number of cycles that are needed for the processor to complete instructions. In the CPI breakdown model, a set of processor events is broken down into components. Processor performance counters calculate metrics for the event components. This approach provides a complete view of how the application behaves concerning processor performance.

Because each processor architecture has different performance counters, POWER and Intel have different CPI breakdown models. Even within Power Systems servers, differences exist between each version of the processor.

Processor performance can be measured by profiling the application with tools such as OProfile or Perf. The CPI breakdown plug-in automates this process, enabling you to access the CPI breakdown model of any C/C++ application without manually tracking the events and calculating the metrics.

#### 6.1.2 CPI events and metrics for POWER7

The CPI breakdown plug-in collects several required events, and then calculates metrics for the CPI breakdown model. These are the CPI events and metrics collected for POWER7.

The **ocount** command collects the following POWER7 events:

- PM\_RUN\_CYC: Run cycles.
- PM\_1PLUS\_PPC\_CMPL: 1 or more ppc instructions finished.
- PM\_CMPLU\_STALL: No groups completed, GCT not empty.
- PM\_CMPLU\_STALL\_BRU: Completion stall due to BRU.
- PM\_CMPLU\_STALL\_DCACHE\_MISS: Completion stall caused by D cache miss.
- PM\_CMPLU\_STALL\_DFPU: Completion stall caused by Decimal Floating Point Unit.
- PM\_CMPLU\_STALL\_DIV: Completion stall caused by DIV instruction.
- PM\_CMPLU\_STALL\_ERAT\_MISS: Completion stall caused by ERAT miss.
- PM\_CMPLU\_STALL\_FXU: Completion stall caused by FXU instruction.
- PM\_CMPLU\_STALL\_IFU: Completion stall due to IFU.
- PM\_CMPLU\_STALL\_LSU: Completion stall caused by LSU instruction.

- PM\_CMPLU\_STALL\_REJECT: Completion stall caused by reject.
- PM\_CMPLU\_STALL\_SCALAR: Completion stall caused by FPU instruction.
- PM\_CMPLU\_STALL\_SCALAR\_LONG: Completion stall caused by long latency scalar instruction.
- PM\_CMPLU\_STALL\_STORE: Completion stall due to store instruction.
- PM\_CMPLU\_STALL\_THRD: Completion stall due to thread conflict. Group ready to complete but it was another thread's turn.
- PM\_CMPLU\_STALL\_VECTOR: Completion stall caused by Vector instruction.
- PM\_CMPLU\_STALL\_VECTOR\_LONG: Completion stall due to long latency vector instruction.
- PM\_GCT\_NOSLOT\_BR\_MPRED: GCT empty by branch misprediction.
- PM\_GCT\_NOSLOT\_BR\_MPRED\_IC\_MISS: GCT empty by branch misprediction + IC miss.
- PM\_GCT\_NOSLOT\_CYC: No itags assigned.
- PM\_GCT\_NOSLOT\_IC\_MISS GCT: empty by I-cache miss.
- PM\_GRP\_CMPL: Group completed

After collecting the events, the **ocount** command calculates the necessary metrics for the CPI breakdown model.

Table 7. CPI breakdown metrics for POWER7

Metric	Formula
BASE_COMPLETION_CPI: Base Completion Cycles	PM_1PLUS_PPC_CMPL/PM_RUN_INST_CMPL
COMPLETION_CPI: Cycles in which a Group Completed	PM_GRP_CMPL/PM_RUN_INST_CMPL
EXPANSION_OVERHEAD_CPI: Cycles due to go overhead of expansion	COMPLETION_CPI - BASE_COMPLETION_CPI
FXU_STALL_CPI Cycles: stalled by Fixed-point Unit	PM_CMPLU_STALL_FXU/PM_RUN_INST_CMPL
FXU_MULTI_CYC_CPI: Cycles stalled by FXU Multi-Cycle Instructions	PM_CMPLU_STALL_DIV/PM_RUN_INST_CMPL
FXU_STALL_OTHER_CPI: Other cycles stalled by FXU	FXU_STALL_CPI - FXU_MULTI_CYC_CPI
GCT_EMPTY_CPI GCT: empty cycles	PM_GCT_NOSLOT_CYC/PM_RUN_INST_CMPL
GCT_EMPTY_IC_MISS_CPI: Cycles GCT empty due to I-Cache Misses	PM_GCT_NOSLOT_IC_MISS/PM_RUN_INST_CMPL
GCT_EMPTY_BR_MPRED_CPI: Cycles GCT empty due to Branch Mispredicts	PM_GCT_NOSLOT_BR_MPRED/PM_RUN_INST_CMPL
GCT_EMPTY_BR_MPRED_IC_MISS_CPI: Cycles GCT empty due to Branch Mispredicts and I-cache Misses	PM_GCT_NOSLOT_BR_MPRED_IC_MISS/PM_RUN_INST_CMPL
GCT_EMPTY_OTHER_CPI: Other GCT empty cycles	(PM_GCT_NOSLOT_CYC-PM_GCT_NOSLOT_IC_MISSPM_GCT_NOSLOT_BR_MPREDPM_GCT_NOSLOT_BR_MPRED_IC_MISS) / PM_RUN_INST_CMPL
IFU_STALL_CPI: Cycles stalled due to Instruction Fetch Unit	PM_CMPLU_STALL_IFU/PM_RUN_INST_CMPL
IFU_STALL_BRU_CPI: Cycles stalled by branches	PM_CMPLU_STALL_BRU/PM_RUN_INST_CMPL
IFU_STALL_OTHER_CPI: Cycles stalled by other IFU operations	IFU_STALL_CPI - IFU_STALL_BRU_CPI
LSU_STALL_CPI: Cycles stalled by Load/Store Unit	PM_CMPLU_STALL_LSU/PM_RUN_INST_CMPL
LSU_STALL_REJECT_CPI: Cycles stalled by LSU Rejects	PM_CMPLU_STALL_REJECT/PM_RUN_INST_CMPL
LSU_STALL_ERAT_MISS_CPI: Cycles stalled by ERAT Translations	PM_CMPLU_STALL_ERAT_MISS/PM_RUN_INST_CMPL

Table 7. CPI breakdown metrics for POWER7 (continued)

Metric	Formula
LSU_STALL_REJECT_OTHER_CPI: Cycles stalled by Other LSU Rejects	LSU_STALL_REJECT_CPI - LSU_STALL_ERAT_MISS_CPI
LSU_STALL_DCACHE_MISS_CPI: Cycles stalled by Data Cache (L1) Misses	PM_CMPLU_STALL_DCACHE_MISS/ PM_RUN_INST_CMPL
LSU_STALL_STORE_CPI: Cycles stalled by Data Store (L1) Misses	PM_CMPLU_STALL_STORE/PM_RUN_INST_CMPL
LSU_STALL_OTHER_CPI: Cycles stalled by Other LSU Operations	LSU_STALL_CPI - LSU_STALL_REJECT_CPI - LSU_STALL_DCACHE_MISS_CPI - LSU_STALL_STORE_CPI
OTHER_STALL_CPI: Other stall cycles	STALL_CPI - FXU_STALL_CPI - VSU_STALL_CPI - LSU_STALL_CPI - IFU_STALL_CPI - SMT_STALL_CPI
RUN_CPI: Total cycles	PM_RUN_CYC/PM_RUN_INST_CMPL
SMT_STALL_CPI: Cycles stalled due to Symmetric Multithreading	PM_CMPLU_STALL_THRD/PM_RUN_INST_CMPL
STALL_CPI: Completion Stall Cycles	PM_CMPLU_STALL/PM_RUN_INST_CMPL
VSU_STALL_CPI: Cycles stalled by Vector-and-Scalar Unit	(PM_CMPLU_STALL_SCALAR + PM_CMPLU_STALL_VECTOR + PM_CMPLU_STALL_DFU)/PM_RUN_INST_CMPL
VSU_STALL_DFU_CPI: Cycles stalled by Decimal Floating-point Unit	PM_CMPLU_STALL_DFU/PM_RUN_INST_CMPL
VSU_STALL_SCALAR_CPI: Cycles stalled by VSU Scalar Operations	PM_CMPLU_STALL_SCALAR/PM_RUN_INST_CMPL
VSU_STALL_SCALAR_LONG_CPI: Cycles stalled by VSU Scalar Long Operations	PM_CMPLU_STALL_SCALAR_LONG/ PM_RUN_INST_CMPL
VSU_STALL_SCALAR_OTHER_CPI: Cycles stalled by Other VSU Scalar Operations	VSU_STALL_SCALAR_CPI - VSU_STALL_SCALAR_LONG_CPI
VSU_STALL_VECTOR_CPI: Cycles stalled by VSU Vector Operations	PM_CMPLU_STALL_VECTOR/PM_RUN_INST_CMPL
VSU_STALL_VECTOR_LONG_CPI: Cycles stalled by VSU Vector Long Operations	PM_CMPLU_STALL_VECTOR_LONG/ PM_RUN_INST_CMPL
VSU_STALL_VECTOR_OTHER_CPI: Cycles stalled by VSU Vector Other	VSU_STALL_VECTOR_CPI - VSU_STALL_VECTOR_LONG_CPI

The CPI breakdown model flow defined for POWER7 Systems servers is depicted in the following table. The sum of the events in a column is the same for each column across the table.

Table 8. CPI breakdown model flow for POWER7

Cycles	Breakdown 1	Breakdown 2	Breakdown 3	Breakdown 4	
PM_RUN_CYC	PM_CMPLU_STALL	PM_CMPLU_STALL_FXU	PM_CMPLU_STALL_DIV		
			PM_CMPLU_STALL_FXU_OTHER		
		PM_CMPLU_STALL_VSU	PM_CMPLU_STALL_SCALAR	PM_CMPLU_STALL_SCALAR_LONG	
				PM_CMPLU_STALL_SCALAR_OTHER	
				PM_CMPLU_STALL_VECTOR	PM_CMPLU_STALL_VECTOR_LONG
			PM_CMPLU_STALL_VECTOR_OTHER		
			PM_CMPLU_STALL_DFP		
		PM_CMPLU_STALL_LSU	PM_CMPLU_STALL_REJECT	PM_CMPLU_STALL_ERAT_MISS	
				PM_CMPLU_STALL_REJECT_OTHER	
			PM_CMPLU_STALL_DCACHE_MISS		
			PM_CMPLU_STALL_STORE		
			PM_CMPLU_STALL_LSU_OTHER		
		PM_CMPLU_STALL_THRD			
	PM_CMPLU_STALL_IFU	PM_CMPLU_STALL_BRU			
		PM_CMPLU_STALL_IFU_OTHER			
	PM_CMPLU_STALL_OTHER				
	PM_GCT_NOSLOT_CYC	PM_GCT_NOSLOT_IC_MISS			
		PM_GCT_NOSLOT_BR_MPRED			
		PM_GCT_NOSLOT_BR_MPRED_IC_MISS			
		PM_GCT_NOSLOT_EMPTY_OTHER			
	PM_GRP_CMPL	PM_1PLUS_PPC_CMPL			
		Overhead of Expansion (PM_GRP_CMPL - PM_1PLUS_PPC_CMPL)			

### 6.1.3 CPI events and metrics for POWER8

The CPI breakdown plug-in collects several required events, and then calculates metrics for the CPI breakdown model. These are the CPI events and metrics collected for POWER8.

The **ocount** command collects the following POWER8 events:

- **PM\_RUN\_CYC**: Processor Cycles gated by the run latch. Operating systems use the run latch to indicate when they are doing useful work. The run latch is typically cleared in the OS idle loop. Gating by the run latch filters out the idle loop.
- **PM\_CMPLU\_STALL**: Cycles where a thread was not completing any groups, when the group completion table had entries for that thread.
- **PM\_GCT\_NOSLOT\_CYC**: Cycles when the Global Completion Table has no slots from this thread.
- **PM\_GRP\_CMPL**: A group completed. Microcoded instructions that span multiple groups will generate this event once per group.
- **PM\_CMPLU\_STALL\_LSU**: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was from the Load store Unit.
- **PM\_CMPLU\_STALL\_VSU**: Cycles stalled by Vector-and-Scalar Unit (**PM\_CMPLU\_STALL\_VECTOR**, **PM\_CMPLU\_STALL\_SCALAR** and **PM\_CMPLU\_STALL\_DFP**).

- PM\_CMPLU\_STALL\_FXU: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was from the Fixed Point Unit.
- PM\_CMPLU\_STALL\_SCALAR: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a scalar floating point instruction.
- PM\_CMPLU\_STALL\_VECTOR: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a vector instruction.
- PM\_CMPLU\_STALL\_REJECT: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes suffered a load/store reject.
- PM\_CMPLU\_STALL\_DIV: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a fixed-point divide instruction.
- PM\_CMPLU\_STALL\_SCALAR\_LONG: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a floating point divide or square root instruction.
- PM\_CMPLU\_STALL\_VECTOR\_LONG: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a long latency vector instruction.
- PM\_CMPLU\_STALL\_DFU: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was from the Decimal Floating Point Unit.
- PM\_CMPLU\_STALL\_ERAT\_MISS: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes suffered an ERAT miss.
- PM\_CMPLU\_STALL\_DCACHE\_MISS: Cycles stalled by Data Cache (L1) misses.
- PM\_CMPLU\_STALL\_STORE: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was a store. This generally happens when we run out of real SRQ entries, which prevents stores from issuing.
- PM\_CMPLU\_STALL\_THRD: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the thread could not complete a group because the completion port its sharing was being used by another thread. In SMT4 mode Thread0 and Thread2 share a completion port and Thread1 and Thread3 share another completion port.
- PM\_CMPLU\_STALL\_IFU: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was from the Instruction fetch unit ( either Branch Unit or CR unit).
- PM\_CMPLU\_STALL\_BRU: Following a completion stall (any period when no groups completed, while group completion table was not empty for that thread) the last instruction to finish before completion resumes was from the Branch Unit. This mostly occurs where the branch has dependencies on a long latency instruction such as a load.
- PM\_GCT\_NOSLOT\_IC\_MISS: Cycles when the Global Completion Table has no slots from this thread because of an Instruction Cache miss.
- PM\_GCT\_NOSLOT\_BR\_MPRED: Cycles when the Global Completion Table has no slots from this thread because of a branch misprediction.
- PM\_GCT\_NOSLOT\_BR\_MPRED\_IC\_MISS: Cycles when the Global Completion Table has no slots from this thread because of a branch misprediction and Instruction cache miss.
- PM\_1PLUS\_PPC\_CMPL: A group containing at least one PowerPC instruction completed. For microcoded instructions that span multiple groups, this will only occur once.

- PM\_1PLUS\_PPC\_DISP: Cycles at least one Instr Dispatched.
- PM\_CMPLU\_STALL\_VECTOR\_LONG: Completion stall due to long latency vector instruction
- PM\_CYC: Processor Cycles.
- PM\_CMPLU\_STALL\_BRU\_CRU: Completion stall due to IFU.
- PM\_CMPLU\_STALL\_FXLONG: Completion stall due to a long latency fixed point instruction.
- PM\_CMPLU\_STALL\_DMISS\_L2L3: Completion stall by Dcache miss which resolved in L2/L3.
- PM\_CMPLU\_STALL\_DMISS\_L2L3\_CONFLICT: Completion stall due to cache miss due to L2 L3 conflict.
- PM\_CMPLU\_STALL\_DMISS\_L3MISS: Completion stall due to cache miss resolving missed the L3.
- PM\_CMPLU\_STALL\_DMISS\_LMEM: Completion stall due to data cache miss that resolved in local memory.
- PM\_CMPLU\_STALL\_DMISS\_L21\_L31: Completion stall by Dcache miss which resolved on chip (excluding local L2/L3).
- PM\_CMPLU\_STALL\_DMISS\_REMOTE: Completion stall by Dcache miss which resolved from remote chip (cache or memory).
- PM\_CMPLU\_STALL\_REJECT\_LHS: Completion stall due to reject (Load Hit Store).
- PM\_CMPLU\_STALL\_ERAT\_MISS: Completion stall due to LSU reject ERAT miss.
- PM\_CMPLU\_STALL\_REJ\_LMQ\_FULL: Completion stall due to LSU reject LMQ full.
- PM\_CMPLU\_STALL\_LOAD\_FINISH: Completion stall due to a Load finish.
- PM\_CMPLU\_STALL\_ST\_FWD: Completion stall due to store forward.
- PM\_CMPLU\_STALL\_NTCCG\_FLUSH: Completion stall due to NTCCG flush.
- PM\_NTCCG\_ALL\_FIN: Cycles in which all instructions in the group have finished but completion is still pending.
- PM\_CMPLU\_STALL\_LWSYNC: Completion stall due to ISYNC/LWSYNC.
- PM\_CMPLU\_STALL\_HWSYNC: Completion stall due to HWSYNC.
- PM\_CMPLU\_STALL\_MEM\_ECC\_DELAY: Completion stall due to mem ECC delay.
- PM\_CMPLU\_STALL\_FLUSH: Completion stall due to flush by own thread.
- PM\_CMPLU\_STALL\_COQ\_FULL: Completion stall due to CO q full.
- PM\_GCT\_NOSLOT\_IC\_L3MISS: GCT empty for this thread due to Icache L3 miss.
- PM\_GCT\_NOSLOT\_BR\_MPRED\_ICMISS: GCT empty for this thread due to Icache Miss and branch mispred.
- PM\_GCT\_NOSLOT\_DISP\_HELD\_MAP: GCT empty for this thread due to dispatch hold on this thread due to Mapper full.
- PM\_GCT\_NOSLOT\_DISP\_HELD\_SRQ: GCT empty for this thread due to dispatch hold on this thread due to SRQ full.
- PM\_GCT\_NOSLOT\_DISP\_HELD\_ISSQ: GCT empty for this thread due to dispatch hold on this thread due to Issue q full.
- PM\_CMPLU\_STALL\_DMISS\_DISTANT: Cycles stalled by L1 reloads from distant interventions and distant memory.

After collecting the events, the **ocount** command calculates the necessary metrics for the CPI breakdown model.

*Table 9. CPI breakdown metrics for POWER8*

Metric	Formula
PM_CMPLU_STALL_CRU	$PM\_CMPLU\_STALL\_BRU\_CRU - PM\_CMPLU\_STALL\_BRU$



Table 9. CPI breakdown metrics for POWER8 (continued)

Metric	Formula
PM_CMPLU_STALL_FXU_OTHER	PM_CMPLU_STALL_FXU - PM_CMPLU_STALL_FXLONG
PM_CMPLU_STALL_VECTOR_OTHER	PM_CMPLU_STALL_VECTOR - PM_CMPLU_STALL_VECTOR_LONG
PM_CMPLU_STALL_SCALAR_OTHER	PM_CMPLU_STALL_SCALAR - PM_CMPLU_STALL_SCALAR_LONG
PM_CMPLU_STALL_VSU_OTHER	PM_CMPLU_STALL_VSU - PM_CMPLU_STALL_VECTOR - PM_CMPLU_STALL_SCALAR_LONG
PM_CMPLU_STALL_DMISS_L2L3_NO_CONFLICT	PM_CMPLU_STALL_DMISS_L2L3 - PM_CMPLU_STALL_DMISS_L2L3_CONFLICT
PM_CMPLU_STALL_DMISS_DISTANT	(PM_CMPLU_STALL_DMISS_L3MISS - (PM_CMPLU_STALL_DMISS_LMEM + PM_CMPLU_STALL_DMISS_L21_L31 + PM_CMPLU_STALL_DMISS_REMOTE))
PM_CMPLU_STALL_REJECT_OTHER	(PM_CMPLU_STALL_REJECT - (PM_CMPLU_STALL_REJECT_LHS + PM_CMPLU_STALL_ERAT_MISS + PM_CMPLU_STALL_REJ_LMQ_FULL))
PM_CMPLU_STALL_LSU_OTHER	(PM_CMPLU_STALL_LSU - (PM_CMPLU_STALL_DCACHE_MISS + PM_CMPLU_STALL_REJECT + PM_CMPLU_STALL_STORE + PM_CMPLU_STALL_LOAD_FINISH + PM_CMPLU_STALL_ST_FWD ))
PM_CMPLU_STALL_OTHER	PM_CMPLU_STALL - PM_CMPLU_STALL_BRU_CRU - PM_CMPLU_STALL_FXU - PM_CMPLU_STALL_VSU - PM_CMPLU_STALL_LSU - PM_CMPLU_STALL_NTCTG_FLUSH
PM_CMPLU_STALL_BLOCK_OTHER	(PM_CMPLU_STALL_THRD - (PM_CMPLU_STALL_LWSYNC + PM_CMPLU_STALL_HWSYNC + PM_CMPLU_STALL_MEM_ECC_DELAY + PM_CMPLU_STALL_FLUSH + PM_CMPLU_STALL_COQ_FULL))
PM_GCT_NOSLOT_IC_L2L3	PM_GCT_NOSLOT_IC_MISS - PM_GCT_NOSLOT_IC_L3MISS
PM_GCT_NOSLOT_DISP_HELD	PM_GCT_NOSLOT_DISP_HELD_MAP + PM_GCT_NOSLOT_DISP_HELD_SRQ + PM_GCT_NOSLOT_DISP_HELD_ISSQ + PM_GCT_NOSLOT_DISP_HELD_OTHER
PM_GCT_EMPTY_OTHER	PM_GCT_NOSLOT_CYC - PM_GCT_NOSLOT_IC_MISS - PM_GCT_NOSLOT_BR_MPRED - PM_GCT_NOSLOT_BR_MPRED_IC_MISS - PM_GCT_NOSLOT_DISP_HELD_MAP - PM_GCT_NOSLOT_DISP_HELD_SRQ - PM_GCT_NOSLOT_DISP_HELD_ISSQ - PM_GCT_NOSLOT_DISP_HELD_OTHER
OTHER_CPI	PM_RUN_CYC - PM_CMPLU_STALL - PM_NTCTG_ALL_FIN - PM_CMPLU_STALL_THRD - PM_GCT_NOSLOT_CYC - PM_GRP_CMPL

Table 9. CPI breakdown metrics for POWER8 (continued)

Metric	Formula
STALL_CPI	$PM\_CMPLU\_STALL / PM\_RUN\_INST\_CMPL$
RUN_CPI	$PM\_RUN\_CYC / PM\_RUN\_INST\_CMPL$

The CPI breakdown model flow that is defined for POWER8 technology enablement is depicted in the following table. The sum of the events in a column is the same for each column across the table.

Table 10. CPI breakdown model flow for POWER8

Cycles	Breakdown 1	Breakdown 2	Breakdown 3	Breakdown 4	Breakdown 5	
PM_RUN_CYC	PM_CMPLU_STALL	PM_CMPLU_STALL_BRU_CRU	PM_CMPLU_STALL_BRU			
			PM_CMPLU_STALL_CRU			
		PM_CMPLU_STALL_FXU	PM_CMPLU_STALL_FXLONG			
			PM_CMPLU_STALL_FXU_OTHER			
		PM_CMPLU_STALL_VSU	PM_CMPLU_STALL_VECTOR	PM_CMPLU_STALL_VECTOR_LONG		
				PM_CMPLU_STALL_VECTOR_OTHER		
			PM_CMPLU_STALL_SCALAR	PM_CMPLU_STALL_SCALAR_LONG		
				PM_CMPLU_STALL_SCALAR_OTHER		
		PM_CMPLU_STALL_VSU_OTHER				
		PM_CMPLU_STALL_LSU	PM_CMPLU_STALL_DCACHE_MISS	PM_CMPLU_STALL_DMISS_L2L3	PM_CMPLU_STALL_DMISS_L2L3_CONFLICT	
					PM_CMPLU_STALL_DMISS_L2L3_NO_CONFLICT	
				PM_CMPLU_STALL_DMISS_L3MISS	PM_CMPLU_STALL_DMISS_LMEM	
					PM_CMPLU_STALL_DMISS_L21_L31	
					PM_CMPLU_STALL_DMISS_REMOTE	
					PM_CMPLU_STALL_DMISS_DISTANT	
	PM_CMPLU_STALL_REJECT		PM_CMPLU_STALL_REJECT_LHS			
			PM_CMPLU_STALL_ERAT_MISS			
			PM_CMPLU_STALL_REJ_LMQ_FULL			
			PM_CMPLU_STALL_REJECT_OTHER			
	PM_CMPLU_STALL_STORE					
	PM_CMPLU_STALL_LOAD_FINISH					
	PM_CMPLU_STALL_ST_FWD					
	PM_CMPLU_STALL_LSU_OTHER					
	PM_CMPLU_STALL_NTGC_FLUSH					
	PM_CMPLU_STALL_OTHER					
	PM_NCTG_ALL_FIN					
	PM_CMPLU_STALL_THRD	PM_CMPLU_STALL_LWSYNC				
		PM_CMPLU_STALL_HWSYNC				
		PM_CMPLU_STALL_MEM_ECC_DELAY				
		PM_CMPLU_STALL_FLUSH				
		PM_CMPLU_STALL_COQ_FULL				
		PM_CMPLU_STALL_BLOCK_OTHER				
	PM_GCT_NOSLOT_CYC	PM_GCT_NOSLOT_IC_MISS	PM_GCT_NOSLOT_IC_L3MISS			
			PM_GCT_NOSLOT_IC_L2L3			
		PM_GCT_NOSLOT_BR_MPRED				
		PM_GCT_NOSLOT_BR_MPRED_ICMISS				
		PM_GCT_NOSLOT_DISP_HELD	PM_GCT_NOSLOT_DISP_HELD_MAP			
			PM_GCT_NOSLOT_DISP_HELD_SRQ			
			PM_GCT_NOSLOT_DISP_HELD_ISSQ			
			PM_GCT_NOSLOT_DISP_HELD_OTHER			
	PM_GCT_EMPTY_OTHER					
	PM_GRP_CMPL					
	OTHER_CPI					

## 6.1.4 Profiling a binary application with CPI from the command line

You can use a simplified version of the CPI breakdown plug-in in the command line, outside the IDE.

### About this task

Complete these steps to profile a binary application with CPI from the command line:

#### Procedure

1. On your Power Systems server, navigate to the `/opt/ibm/ibm-sdk-lop/bin` folder.
2. Locate the `cpi_profile.sh` script
3. Run the script.

**Note:** Run the script without parameters to see the usage instructions.

## 6.1.5 Profiling a project with the CPI breakdown plug-in

You can use the CPI breakdown plug-in to analyze your project for performance improvement through the CPI breakdown model.

### Before you begin

Ensure that you have compiled the binary files. Right-click on the project name and select **Build Project**.

### About this task

Complete these steps to profile a project on the local system using the CPI breakdown plug-in.

#### Procedure

1. In the Project Explorer pane, right-click the project name. Click **Profiling Tools > Profiling Tools Configuration**.
2. In the Profiling Tools Configuration window, click **Profile with CPI** in the left pane, and then select the New launch configuration icon near the upper left of the window.
3. On the Main tab, beside the **C/C++ executable** field, select the binary file that you want to profile.
4. Optional: On the Main tab, click **Browse** from the **Working directory** field to select the folder to be used by the profiling tool.
5. Optional: On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
6. Optional: On the CPI Options tab, select option **Select Source Disassembly View**.
7. Click **Profile** to begin the profiling. After the profiling completes, a CPI Breakdown perspective opens, with the CPI breakdown model (CBM) view shown. The CPI Breakdown Model view shows all the metrics, events, and their values. Red boxes represent hot spot events.

In addition, the following views open in the other tabs:

- The Events view shows all the events that were gathered using the **ocount** tool. Use the search field to quickly find a specific event.
- The Metrics view shows all the metrics that were calculated. Use the search field to quickly find a specific metric.
- The Radar Chart View provides a graphical visualization of the profiling, allowing you to compare several rounds of execution.

To compare rounds, locate the files with `*.cpi` extension at the project explorer view and proceed as following:

- a. Load a CPI-Breakdown round: Right click on the `*.cpi` file and select **CPI Analyzes > Load events**.
- b. Compare CPI-Breakdown rounds: Select at least two `*.cpi` files, and right click to select **CPI Analyzes > Events Compare**.

- c. Average of CPI-Breakdown rounds: Select one or more \*.cpi files, and right click to select **CPI Analyzes > Events Average**.
8. You can drill down through the profiling results to the source code.
    - a. In the CPI breakdown model view, double-click any colorized (red, soft red, orange) box that represents a hot spot. The Drilldown view opens in a new tab. The Drilldown view lists the source code elements that caused the event you selected.
    - b. In the Drilldown view, double-click a source code element. This action displays the source code, at the line that corresponds to the selected event, in the source code view on the lower pane. The source code view lists the source file name as the tab name.

---

## 6.2 Analyzing performance with OProfile

OProfile is a system-wide profiler for Linux systems that allows for the analysis of system performance and the identification of code hot spots. It also contains support for hardware performance counters. IBM Software Development Kit for Linux on Power provides integration with OProfile to allow you to profile running applications while conserving system resources.

OProfile profiles hardware and software interrupt handlers, kernel modules, the kernel, shared libraries, and applications. OProfile provides commands, or utilities, for controlling profiling.

Legacy OProfile consists of the **opcontrol** shell script for configuring, starting, and stopping a profiling session. To this purpose, a kernel driver is used for collecting samples, which are recorded into sample files. A disadvantage of this mode is the necessity of elevated user privileges to run **opcontrol**.

**Operf** was designed to be used in place of **opcontrol** for profiling. It uses the Linux Performance Events Subsystem, and therefore, does not require the use of the **opcontrol** daemon or any elevated privileges.

You can use either method, **opcontrol** or **operf**, to profile with OProfile and IBM SDK for Linux on Power.

### 6.2.1 Profiling a project with OProfile

You can use OProfile to profile your project to analyze system performance while running applications.

#### Before you begin

Ensure that you have completed the following:

1. OProfile available on your system.
2. Building the project.
3. Verifying that the Linux tools path specifies the version of Advance Toolchain for Linux on Power you want to use during profiling. See 2.4.7, “Changing the Advance Toolchain version,” on page 17 for information.

#### About this task

Complete these steps to profile a project using OProfile.

#### Procedure

1. Switch to the C/C++ perspective by clicking **Window > Open Perspective > C/C++**. Click **OK**.
2. View the project by clicking the project name in the Project Explorer pane.
3. Click **Profiling Tools > Profiling Tools Configurations**.
4. In the Profiling Tools Configurations window, click **Profile with OProfile** in the left pane, and then click the **New launch configuration** icon near the upper left of the window.

5. On the Main tab, beside the **C/C++ Application** field, click **Search Project** to select the binary file that you want to profile.
6. If necessary, click the arrows in the **Build configuration** field and select **Default**.
7. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
8. On the Global tab, click the arrow in the **Profile with** field to select which profiling method, **opcontrol** or **opperf**, to use. If you select **opcontrol**, you can also select from the following options:
  - Specify a **kernel image file**. This option collects more detailed information about the operation of a program in the Linux kernel. Type the file name or click **Browse** to locate and select it.  
  
**Note:** When you select this option, you must also select the **Include dependent kernel modules** option.
  - **Include dependent shared libraries**. Select this option to include samples from shared libraries that are used by the profiled program. These samples are then aggregated in the profile results.
9. Optional: On the Events tab, you can clear the **Use default event** check box to select a different event to be monitored. You can also select multiple events within the same group. The number of events varies by system, but typically you can select up to six events.
10. Click **Profile** to begin the profiling. After the profiling completes, an OProfile tab opens and displays the event types that were profiled.
11. You can expand the events to view the binary image, function, and line numbers where the profiling collected most of the events that were profiled. The results are sorted with the functions that the program spent the most time executing shown first. You can click on an item to open the source code.

#### Related information:

 [Taking advantage of OProfile](#)

## 6.2.2 Profiling a synchronized project with OProfile

You can use OProfile to profile a synchronized project to analyze system performance while running applications.

### Before you begin

Ensure that you have completed the following:

1. OProfile available on your system.
2. Building the project.
3. Verifying that the Linux tools path specifies the version of Advance Toolchain for Linux on Power you want to use during profiling. See 2.4.7, “Changing the Advance Toolchain version,” on page 17 for information.

### About this task

Complete these steps to profile a synchronized project using OProfile.


#### Procedure

1. In the Project Explorer pane, right-click the project name. Click **Profiling Tools > Profiling Tools Configurations**.
2. In the Profiling Tools Configurations window, click **Profile with OProfile (Remote)** in the left pane, and then click the New launch configuration icon near the upper left of the window.
3. On the Main tab, below C/C++ Executable, click **Browse** to select the binary file that you want to profile.
4. If necessary, click the arrows in the **Build configuration** field and select **Default**.
5. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.

6. Optional: On the Events tab, you can clear the **Use default event** check box to select a different event to be monitored. You can also select multiple events within the same group. The number of events varies by system, but typically you can select up to six events.
7. On the Global tab, click the arrow in the **Profile with** field to select which profiling method, **opcontrol** or **operf**, to use. If you select **opcontrol**, you can also select from the following options:
  - Specify a **kernel image file**. This option collects more detailed information about the operation of a program in the Linux kernel. Type the file name or click **Browse** to locate and select it.

**Note:** When you select this option, you must also select the **Include dependent kernel modules** option.
  - **Include dependent shared libraries**. Select this option to include samples from shared libraries that are used by the profiled program. These samples are then aggregated in the profile results.
8. Click **Profile** to begin the profiling. After the profiling completes, an OProfile tab opens and displays the event types that were profiled.
9. You can expand the events to view the binary image, function, and line numbers where the profiling collected most of the events that were profiled. The results are sorted with the functions that the program spent the most time executing shown first. You can click on an item to open the source code.

#### Related information:

 Taking advantage of OProfile

---

## 6.3 Analyzing performance with Perf

You can use Perf to profile your project and analyze its performance. Perf uses information produced by the Linux kernel perf events subsystem to profile a running application. Perf analyzes software, hardware, and tracepoint events.

Perf is an optional but recommended package. If you want to use Perf, you must download and install it as described in 2.2, “Recommended and optional packages,” on page 9.

### 6.3.1 Profiling a project with Perf

You can use Perf to profile your project to analyze its performance.

#### Before you begin

Before you can use Perf, you must ensure that you have downloaded and installed it as described in 2.2, “Recommended and optional packages,” on page 9.

#### About this task

Complete these steps to profile a project using Perf.

#### Procedure

1. Switch to the C/C++ perspective by clicking **Window > Open Perspective > C/C++**. Click **OK**.
2. View the project by clicking the project name in the Project Explorer pane.
3. Click **Profiling Tools > Profiling Tools Configurations**.
4. In the Profiling Tools Configurations window, click **Profile with Perf** in the left pane, and then click the New launch configuration icon near the upper left of the window.
5. On the Main tab, beside the **C/C++ Application** field, click **Search Project** to select the binary file that you want to profile.
6. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
7. Optional: On the Perf Options tab, you can change default profiling settings.

- Optional: On the Perf Events tab, you can clear the **Default Event** check box to select a different event to be monitored. You can also select multiple events.
- Click **Profile** to begin the profiling. After the profiling completes, a Perf tab opens and displays the event types that were profiled.
- You can expand the events to view the binary image, function, and line numbers where the profiling collected most of the events that were profiled. The results are sorted with the functions that the program spent the most time executing shown first. You can click on an item to open the source code.

### 6.3.2 Profiling a synchronized project with Perf

You can use Perf to profile a synchronized project to analyze its performance.

#### Before you begin

Before you can use Perf, you must ensure that you have downloaded and installed it as described in 2.2, “Recommended and optional packages,” on page 9.

#### About this task

Complete these steps to profile a synchronized project using Perf.

#### Procedure

- In the Project Explorer pane, right-click the project name. Click **Profiling Tools > Profiling Tools Configurations**.
- In the Profiling Tools Configurations window, click **Profile with Remote Perf** in the left pane, and then click the New launch configuration icon near the upper left of the window.
- On the Main tab, below **C/C++ Executable**, click **Browse** to select the binary file that you want to profile.
- On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
- Optional: On the Perf Options tab, you can change default profiling settings.
- Optional: On the Perf Events tab, you can clear the **Default Event** check box to select a different event to be monitored. You can also select multiple events.
- Click **Profile** to begin the profiling. After the profiling completes, a Perf tab opens and displays the event types that were profiled.
- You can expand the events to view the binary image, function, and line numbers where the profiling collected most of the events that were profiled. The results are sorted with the functions that the program spent the most time executing shown first. You can click on an item to open the source code.

---

## 6.4 Profiling a project with gprof

The GNU profiler, or gprof, is a performance analysis tool that creates an execution profile of your C or C++ program. Gprof uses static instrumentation to calculate the amount of time expended in a function of the program, and the number of times the function is called.

#### Before you begin

You must compile the binary files with the `-pg` option. This option produces profiling information required for analysis by gprof. See “Recommendations for debug flags” on page 35 and 4.1.2, “Setting debug flags for Autotools-based projects,” on page 40 for details and instructions for setting the flags.

#### About this task

Complete these steps to profile a project with gprof.



## Procedure

1. Run the application. See 2.4.8, “Running an executable program in a project,” on page 18. Profiling information is output to the `gmon.out` file in the same folder location as the application binary file.
2. Refresh the view by right-clicking the project name and clicking **Refresh**.
3. Double-click the `gmon.out` file.
4. In the Binary File selection window, click **Workspace** or **File System** to select the binary file that produced the profile data. Click **OK**. A `gprof` view tab appears. The view enables you to visualize profiling information.
5. Optional: Use the controls in the upper right of the view to change information visualization.

---

## 6.5 Analyzing application behavior using Valgrind

Valgrind is an open source programming tool used for detecting memory leaks, memory debugging, and performing detailed profiling to find blockages in programs. Supported Valgrind tools include Memcheck memory error detector, Helgrind thread error detector, Massif memory usage profiler, and Cachegrind cache and branch-prediction profiler.

Memcheck detects memory management problems, reporting these errors as they occur. Memcheck gives the source line number at which the error occurred, and a stack trace of the functions called up to that line. Problems detected by Memcheck include the following:

- Illegal read and write operations
- Use of uninitialized values
- Use of uninitialized or unaddressable values in system calls
- Illegal freeing of memory
- Memory freed with inappropriate deallocation functions
- Overlapping of source and destination blocks on memory copy functions
- Memory leaks

Helgrind is a POSIX thread (Pthreads) debugger that finds data races in multithreaded programs as well as other Pthread-related problems. Problems detected by Helgrind include the following:

- Misuses of the Pthreads API
- Deadlock caused by lock ordering
- Data race conditions

Massif is a memory profiler that analyzes heap and stack usage.

Cachegrind is a cache and branch-prediction profiler. Cachegrind identifies the number of cache misses, memory references, and instructions executed for each line of source code.

### 6.5.1 Profiling a project using Valgrind

You can use Valgrind to profile your project to detect memory leaks, debug memory issues, and perform detailed profiling.

#### Before you begin

Before you profile a project, ensure that the project has been built.

#### About this task

Complete these steps to profile a project using Valgrind.

## Procedure

1. Switch to the C/C++ perspective by clicking **Window > Open Perspective > C/C++**. Click **OK**.
2. In the Project Explorer pane, right-click the project name. Click **Profiling Tools > Profiling Tools Configurations**.
3. In the Profiling Tools Configurations window, click **Profile with Valgrind** in the left pane, and then click the **New launch configuration** icon near the upper left of the window.
4. On the Main tab, beside the **C/C++ Application** field, click **Search Project** to select the binary file that you want to profile.
5. If necessary, click the arrows in the **Build configuration** field and select **Default**.
6. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
7. On the **Valgrind Options** tab, select the tool to run and select the options that best fit your profiling needs.
8. Click **Profile** to begin the profiling.

## Results

The source code is displayed. Lines that have errors that were detected by Valgrind are highlighted with an “X”. See 6.5.3, “Applying quick fixes for Valgrind-reported errors,” on page 75 for information about how to resolve them.

## 6.5.2 Profiling a synchronized project using Valgrind

You can use Valgrind to profile your synchronized project to detect memory leaks, debug memory issues, and perform detailed profiling.

### Before you begin

Before you profile a project, ensure that the project has been built.

### About this task

Complete these steps to profile a synchronized project using Valgrind.

## Procedure

1. Switch to the remote C/C++ perspective by clicking **Window > Open Perspective > Other > Remote C/C++**. Click **OK**. The synchronized project should be displayed under the Project Explorer view.
2. In the Project Explorer pane, right-click the project name. Click **Profiling Tools > Profiling Tools Configurations**.
3. In the Profiling Tools Configurations window, click **Profile with Valgrind (Remote)** in the left pane, and then click the **New launch configuration** icon near the upper left of the window.
4. On the Main tab, beside the **C/C++ Application** field, click **Browse** to select the binary file that you want to profile.
5. If necessary, click the arrows in the **Build configuration** field and select **Default**.
6. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
7. On the **Valgrind Options** tab, select the tool to run and select the options that best fit your profiling needs.
8. Click **Profile** to begin the profiling.

## Results

The source code is displayed. Lines that have errors that were detected by Valgrind are highlighted with an “X”. See 6.5.3, “Applying quick fixes for Valgrind-reported errors” for information about how to resolve them.

### 6.5.3 Applying quick fixes for Valgrind-reported errors

IBM SDK for Linux on Power provides quick fixes for errors found by Valgrind. Quick fixes are suggestions or tips that might help correct identified errors.

#### Before you begin

Ensure that you have profiled your project with Valgrind, as described in 6.5.1, “Profiling a project using Valgrind,” on page 73 and 6.5.2, “Profiling a synchronized project using Valgrind,” on page 74.

#### About this task

After you run Valgrind, the source code is displayed, and the lines that have errors are highlighted with an “X”. Complete these steps to fix problems that are found with Valgrind.

#### Procedure

1. Right-click the line with the error, and select **Quick Fix**. If there are any quick fixes available, they are displayed in a list.
2. Select the appropriate quick fix. The fix is applied and the program source changed automatically to fix the source.

---

## 6.6 Analyzing POSIX Threads using Trace Analyzer

The IBM Software Development Kit for Linux on Power includes Trace Analyzer, a plug-in for graphical and numerical analysis of performance traces.

### 6.6.1 Trace Analyzer overview

Trace Analyzer allows graphic visualization and analysis of POSIX Threads (Pthreads) concurrency.

Concurrency information is generated by a Pthread monitoring tool running on Pthread-based applications. The monitoring tool can be invoked from within the visualizer or from the command line. The concurrency information includes synchronization using mutexes, spinlocks, and conditional variable signaling.

With the Trace Analyzer, you can perform the following analysis tasks:

- Collect the Pthreads profile and view the resulting trace.
- View the active threads and the blocking operations.
- View the active locks and spinlocks, threads waiting on locks, threads holding locks, and queue sizes.
- View conditional variables, threads waiting for them, and the contention for the associated mutexes.
- Collect a trace of blocking I/O requests, and view the trace together with the synchronization data.

### 6.6.2 Profiling a project with Trace Analyzer

You can profile a project with Trace Analyzer to analyze Pthreads concurrency.

#### Before you begin

Before you profile a project, ensure that it has been built.

## Procedure

1. In the Project Explorer pane, right-click the project name and select **Profile Tools > Profile Configurations**.
2. In the Profile Configurations window, select **Profile with Trace Analyzer for Pthreads** in the left pane, and then select the **New launch configuration** icon near the upper left of the window.
3. On the Main tab, click **Browse** from the **C/C++ executable** to select the binary to be profiled.
4. On the Main tab, click **Browse** from the **Working directory** to select the folder to be used by the profiling tool.
5. On the Arguments tab, specify the necessary arguments to run the application.
6. Click **Profile**.
7. Optional: To enable operating system I/O monitoring using SystemTap along with data collection, complete this step. On the Trace Analyzer Options tab in the Operating system monitoring section, select the **I/O** option from the list of available options. The default is no I/O monitoring. If I/O monitoring is enabled, the I/O monitoring traces read and write system calls. The system call times are shown with application times.
8. Optional: To adjust the depth of stack trace information to be collected in the traces, complete this step. On the Trace Analyzer Options tab in the Number of frames to back trace section, select the number of frames from the list of available options. Available values are 0 - 9; the default is 2.

### 6.6.3 Collecting a trace by setting variables from the command line

You can set variables from the command line to affect how a trace is collected.

#### About this task

Select from the following options.

#### Procedure

- Set the **SASSTOREPATH** variable to point to a directory for temporary log files. Ensure that you have write permission and enough disk space in the directory. 32-bit and 64-bit applications must use different directories. Different profiling runs can use the same directory if they use different tags.
  - If the application under monitoring terminates abnormally, clean the data directory by setting the **SASSTOREPATH** variable as follows:

```
SASSTOREPATH=directory <${PMHOME}>/pthreadmon/ bin/sasutil
```

Specify the same directory as in the monitoring run for *directory*. Running this command displays a list of options; select 6 to remove the data.

- Set the **MONITOREDAPPTAG** variable to be the identifier of the monitored execution. Allowed characters are a to z, A to Z, and 0 to 9. Usually the tag contains the application name, settings description, and date and time of the run. The tag cannot be longer than 100 characters.
- Optional: Set the **BT\_LEVEL** variable to be the depth of the call stack to be recorded with monitored events. Allowed values are 0 - 4; the default is 2.
- Optional: Set the **OSMONITOR** variable to describe the operating system data to collect. The only supported value is `io`, which triggers logging of read/write system calls. It is supported only for 64-bit programs and might not work with large logs.

#### Results

The log files are written to `($MONITOREDAPPTAG)_logfile_number.pthreads` files in the current directory. If the monitored data is large, the log is split into multiple files, for example: `($MONITOREDAPPTAG)_log0.pthreads`, `($MONITOREDAPPTAG)_log1.pthreads`, `($MONITOREDAPPTAG)_log2.pthreads`, and so on.

**Note:** If there is not enough space to hold all of the events, the logger might drop events in the beginning of the trace.

## Example

**Note:** By using the code examples, you agree to the terms of the 10, “Code license and disclaimer information,” on page 95.

```
[pmhome/pthread-mon/test]# SASSTOREPATH=/tmp/sasdata64
MONITOREDAPPTAG=appTag BT_LEVEL=2 ../scripts/pthread-mon ./app64
<... snip ...>
Hello world!
```

```
[pmhome/pthread-mon/test]# cat appTag_log0.pthreadss
0 1350069905206059 27241 0x10010c90
1 1350069905216439 27241 0 4398060282400
2 1350069905216827 27241 4398060282400 0x10000758 0x4000014276c
99 0x10000758 ../app/app ./app.c:20
99 0x4000014276c /lib64/power5/libc.so.6 ??:0
3 1350069905296762 27241 0
```

## 6.6.4 Record Details view

The Record Details view shows name/value pairs for all the fields defined for the selected record.

The call-stack information in the Record Details view can be used to match specific blocking events to source code lines. For example, BacktraceExec0 is the executable program, and BacktraceLine0 is the line in that executable program from which the blocking function is called.

The Record Details view includes the following tabs: Overview, Waits by thread, Hot locks, Hot spins, and Hot Cond Var. In each tab of the view, the x axis is the time, increasing from left to right. Each Pthread view has options available in the Properties view for toggling the display of each metric.

### Overview tab

The Overview tab view gives an overview of all threads synchronization actions . The threads are shown as gray blocks spanning the entire thread lifetime. Synchronization events are shown above the thread block, and operating system events (if available) are shown below it.

**Start** Indicates when the thread is created.

**End** Indicates when the thread is ended.

**Join** Indicates that the thread is waiting to join another thread.

#### Mutex\_lock

Indicates that the thread is blocking because it is trying to acquire a mutex.

#### Condvar\_wait

Indicates that the thread is blocking because it is waiting on a conditional variable.

**IO** Indicates that the thread was blocked in I/O (operating system event).

### Waits by thread tab

The Waits by thread tab view displays the Pthread locks for each thread that is blocked, waiting to acquire a specific mutex. The display includes colors to differentiate between the different mutexes. The list of displayed threads can be edited through the Properties view.

### Hot locks tab

The Hot locks tab view shows the number of Pthreads that are blocked, waiting to acquire each mutex.

**Blocking before lock**

Shows the number of threads blocking as a result of calling `pthread_mutex_lock`. The color displayed is determined by the last thread in the queue.

**Hold** Shows the lock lifetime and the intervals for which the lock was held. The colors displayed for lock lifetime match those in the Waits by thread view. The color displayed for the intervals is determined by the holding thread.

**Blocking after wait**

Shows the number of threads blocking while trying to reacquire a mutex that was lost while waiting on a conditional variable. The color displayed is determined by the last thread in the queue.

**Hot spins tab**

The Hot spins tab view shows the thread holding the spin lock and number of threads waiting on that spin lock over time.

**Wait** Shows the number of threads blocking as a result of calling `pthread_spin_lock`. The color displayed is determined by the oldest thread in the queue.

**Hold** Shows the spin lock lifetime and the intervals in which it was held. The colors displayed for lock lifetime match those in the Waits by thread view. The color displayed for the intervals is determined by the holding thread.

**Hot Cond Var tab**

The Hot Cond Var tab view shows the number of Pthreads waiting for each conditional variable. In addition, the view shows how many threads are blocking, waiting to acquire this mutex, for each mutex guarding the conditional variable. As in the Hot locks tab, the view on this tab contains two graphs for each mutex.

**Blocking before lock**

Shows the number of threads blocking as a result of calling `pthread_mutex_lock`. The color displayed is determined by the last thread in the queue.

**Hold** Shows the lock lifetime and the intervals for which the lock was held. The colors displayed for lock lifetime match those in the Waits by thread view. The color displayed for the intervals is determined by the holding thread.

**Blocking after wait**

Shows the number of threads blocking while trying to reacquire a mutex that was lost while waiting on a conditional variable. The color displayed is determined by the last thread in the queue.

The list of displayed conditional variables can be edited through the Properties view.

---

## 6.7 Analyzing coverage with gcov

The GNU test coverage program, or `gcov`, is a tool that uses static instrumentation to generate test coverage information.

**Before you begin**

Ensure that the following steps have been completed before you use `gcov`.

1. Compile the binary files with the `-ftest-coverage` and `-fprofile-arcs` options.

-ftest-coverage produces a text file that gcov uses to show program coverage. -fprofile-arcs instruments the resulting binary flow arcs. See “Recommendations for debug flags” on page 35 and 4.1.2, “Setting debug flags for Autotools-based projects,” on page 40 for details and instructions for setting the flags.

2. Link the binary files to the gcov library. Complete the following steps.
  - a. Expand **C/C++ Build > Settings**.
  - b. Expand **GCC C++ Linker > Libraries**.
  - c. In the Libraries window, click +, the “Add...” icon.
  - d. In the **Libraries** field in the Enter Value window, type gcov.

## About this task

Complete these steps to profile a project with gcov.

### Procedure

1. Run the application. See 2.4.8, “Running an executable program in a project,” on page 18. Coverage data is saved to *object\_name.gcda* files, one for each object.
2. Refresh the view by right-clicking the project name and clicking **Refresh**.
3. Double-click the *object\_name.gcda* file.
4. In the Binary File selection window, click **Workspace** or **File System** to select the binary file that produced the coverage data. Click **OK**. A gcov view tab appears. The view enables you to visualize profiling information. Double-click any row in the report to display source code with coverage highlighted.
5. Optional: Use the controls in the upper right of the view to change information visualization.

---

## 6.8 Analyzing performance with IBM Power System Performance Advisor

The IBM Power Systems Performance Advisor (PPA) plug-in allow users to profile C/C++ applications selecting a set of metrics based on the chosen target processor. PPA leverages Ocount tool; an OProfile tool used to count native hardware events, to gather the processor performance data and calculate the metrics.

### About this task

Complete those steps to profile a local or a synchronized project that uses PPA plug-in.

### Procedure

1. Switch to the C/C++ perspective by selecting **Window > Open Perspective > C/C++**. Click **OK**.
2. Right-click the project in the Project Explorer pane.
3. Select **Profile Tools > Profile Configurations**.
4. In the Profiling Configurations window, select **Profile with PPA** in the left pane, and then select the **New launch configuration** icon near the upper left of the window.
5. On the Metrics tab, complete the following fields:
  - POWER CPU version in **CPU Model** field
  - Metric analysis type in the **Analysis type** field
  - Metrics in the **Metrics group**field

**Note:** To change the Ocount that is used during the profile, select **Change path** from the **Counter Tool** group.

6. On the Main tab, click **Browse** from the **C/C++ executable** field to select the binary file to be profiled.
7. On the Main tab, click **Browse** from the **Working directory** field to select the folder to be used by the profiling tool.
8. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
9. Click **Profile** to begin the profiling. After the profiling completes, a PPA view opens and displays each metrics value with the corresponding events. For some metrics, a chart is displayed, which shows comparisons between metrics values and in some cases events values.
10. It is possible to drill down through the events that are part of metrics to the source code.
  - a. In the PPA view, double-click any drill-down button in front of events. The Drilldown view opens and displays the source code elements that caused the selected event.
  - b. In the view, the results are sorted with the functions that the program spent the most time running. You can double-click an item to open the source code at the specific file and line.

---

## 6.9 Analyzing projects built with the Build Advisor

The Build Advisor plug-in allows users to receive advice on the set of flags that can be tuned for Power during the build process in order to improve the application performance. It parses the build output, analyzes the set flags that are used, and based on that analysis, provides advice on the flags that can make the application perform better.

### 6.9.1 Enabling the Build Advisor

By default, the Build Advisor is disabled for new projects.

#### About this task

Use the following steps to activate it:

#### Procedure

1. Right-click the project in the **Project Explorer** pane and select **Properties**.
2. In the **Properties** window, expand **C/C++ Build** and select **Settings**.
3. In **Settings** window, change to **Error Parsers** tab and mark the option **Build Advisor Output Parser**.
4. Press **OK**

#### Results

The Build Advisor is now successfully enabled. By default, the Build Advisor gives advice for Advance Toolchain compiler flags.

### 6.9.2 Enabling extra advice from Build Advisor

There are some advice provided by Build Advisor that are not activate by default because they might change the binary behavior. For example, some flags can increase the binary size or reduce the floating point operation precision in order to gain performance.

#### About this task

Follow the steps below to activate those extra advice:

#### Procedure

1. Right-click the project in the **Project Explorer** pane and select **Properties**.
2. In the **Properties** window, select **Build Advisor**.



3. In the **Select Compiler** option, select the compiler being used in the project. By default, Advance Toolchain is selected.
4. Under the option **Enable extra advice**, enable the extra advice that will be used during the analysis.
5. Press **OK**.

## Results

The Build Advisor is now successfully activated.

## 6.9.3 Using the Build Advisor

### About this task

Complete the steps below to use Build Advisor.

**Note:** It is important to ensure that the Build Advisor is enabled.

### Procedure

1. Right-click the project in the **Project Explorer** pane and select **Build Project**.
2. After the project build, the advice will be shown in the **Build Advisor** view.



---

## 7 Analyzing performance with Source Code Advisor and FDPR

The IBM Software Development Kit for Linux on Power provides two related tools, Source Code Advisor (SCA) and the Feedback Directed Program Restructuring tool (FDPR), that implement feedback-directed, post-link program analysis and optimization technology. SCA finds and visualizes performance problems in the application source code, using journaling information produced by using FDPR. SCA alerts you to problems, and provides you with tips for restructuring source code and modifying compiler flags.

---

### 7.1 Source Code Advisor and FDPR overview

Source Code Advisor (SCA) and Feedback-Directed Program Restructuring (FDPR) work together to allow you to analyze and optimize your applications.

FDPR works similarly to a compiler: it reads a linked executable program and produces an optimized version of it. Both regular executable and shared library forms are supported. The optimization uses an execution profile, collected by running an instrumented version of the input.

During the code optimization process, FDPR produces a journal of the optimizations performed. The Source Code Advisor uses this journal, produced as an XML file, to highlight potential problems in your source code and to offer suggested solutions. The journal explains each optimization site, including the source location, execution count, the performance problem found, and the user action required to resolve the problem. It is important to select a representative workload for both SCA and for FDPR so that the optimization step is effective.

Because SCA uses information gathered by FDPR, knowledge of this tool is important. To understand the main requirements and principles, review the FDPR Getting Started document, available in the IBM SDK for Linux on Power online help. (You can access this document from the user interface by clicking **Help > Help Contents**. Then expand **FDPR Optimization Tools Documentation > Getting Started**.)

The combination of SCA and FDPR provide you with two major approaches to performance analysis and optimization:

- Find and visualize performance problems in the source program using feedback-directed analysis.
- Perform feedback-directed optimization of an executable program (or a shared library).

#### Performance problem visualization using Source Code Advisor

The SCA configuration allows you to specify the workload needed to collect the profile of the program. When running this configuration, the program is built, if necessary, using the standard project build process. Once the executable is available, FDPR creates an instrumented version and runs it using the specified workload. FDPR then performs a pseudo optimization step producing a journal of the performance problems found. The result is an XML-formatted file that lists the specific problems found, their exact location in the source, and so on. With the XML journal available, the Source Code Advisor view is displayed to visualize the set of problems, allowing you to navigate through the problems and the corresponding places in the source where they were found. The view provides a recommended course of action for each problem at various abstraction levels (source change, compiler switches, and so on.)

#### Feedback-directed optimization using FDPR

FDPR optimization is begun through the FDPR configuration. As with the SCA configuration, you specify a workload, typically the same workload. You can also specify the set of optimizations to be used. Running the FDPR configuration includes optional instrumentation and profiling steps. These steps are

needed if the original program has changed, if no profile exists yet, or if you specifically requested to re-create it. Following that, the optimization step is performed, producing the optimized version.

---

## 7.2 Running the Source Code Advisor on a project

You can run SCA on a project to begin the process of performance problem visualization.

### Before you begin

Ensure that you compile the binary files with the `-Wl` and `-q` flags. See “Recommendations for debug flags” on page 35 and 4.1.2, “Setting debug flags for Autotools-based projects,” on page 40 for details and instructions for setting the flags.

### About this task

Complete the following steps.

#### Procedure

1. In the Project Explorer pane, right-click the project name. Click **Profile Tools > Profile Configurations**.
2. In the Profile Configurations window, expand **Profile with Source Code Advisor** in the left pane and select the **New launch configuration** icon near the upper left of the window.
3. On the FDPR Options tab, select appropriate options. See 7.6, “Optimizing executable code using FDPR,” on page 86 for more details on specifying FDPR options.
4. On the Main tab, click **Browse** from the **C/C++ Application** field to select the binary file to be profiled.
5. On the Main tab, click **Browse** from the **Working directory** field to select the folder to be used by the profiling tool.
6. On the Arguments tab, specify any arguments to pass to the binary file to be profiled.
7. Click **Profile**.

### Results

SCA provides quick fixes for some events that are reported. Quick fixes are suggestions that might help to correct the identified performance problems. For the events where a quick fix is available, a yellow warning is created in the source code at the specific reported line. To apply a quick fix, complete the following steps:

1. Move the cursor to the highlighted source code line warning and press **Ctrl + 1**.
2. Select the appropriate quick fix and press **Enter**. The fix is applied and the source is changed automatically.

SCA first builds the program, if necessary, using the standard project build process. After the executable file is available, an instrumented version is created by FDPR and run using the specified workload. FDPR then performs a pseudo-optimization step, producing a journal of the performance problems found and fixed. The journal is an XML formatted file that lists the specific problems found and their exact location in the source. The journal corresponds to a specific workload, as specified on the Arguments tab in the profile configuration.

When the journal is available, the SCA results are displayed automatically in the Source Code Advisor view. See 7.4, “Reviewing the Source Code Advisor results,” on page 85 for information about using the SCA view.

---

## 7.3 Viewing Source Code Advisor on an existing journal

You can open SCA on a journal that has already been collected.

### About this task

Complete the following steps.

#### Procedure

1. If the SCA view is not already open, complete the following steps to open the SCA view.
  - a. Click **Window > Show View > Other**.
  - b. In the Show View window, expand **FDPR** and click **Source Code Advisor**. Click **OK**.
2. In the Project Explorer pane, select the journal file to be viewed. The journal file name begins with the name of the program and ends with `-jour.xml`. See 7.4, “Reviewing the Source Code Advisor results” for information about using the SCA view.
3. If you reprofile the program with SCA, the profile will be replaced. If you want to keep the journal file but collect a journal for a different workload, you can save it with a different name. To save the file, copy and paste it with a different name. Ensure that the new name ends with `-jour.xml`.

---

## 7.4 Reviewing the Source Code Advisor results

After you run SCA, you can review the results including the specific performance problems.

### About this task

The SCA results are shown in the Source Code Advisor view. This view consists of the following panes:

- A performance event pane, in an expandable tree format, on the left.
- A problem/solution pane on the right.

At the top level, the performance event tree shows each of the performance event types, such as `FIX LOAD-HIT-STORE`, `UNROLL LOOP`, and `INLINE FUNCTION`. The right pane shows the problem and solution for the selected event. Refer to 7.5, “Source Code Advisor events” for information about the performance events and the various levels (source code, compiler, and linker).

The performance event tree provides detailed information about the actual instances of the performance events in the user program.

#### Procedure

- To display the functions where a performance event occurred, expand an event type at the root level . The function entries are sorted by frequency of occurrence when the instrumented program was run.
- To display site-specific parameters for the event, expand a line entry for the event. For example, select a parameter that represents a site in the program such as a line number or function. This displays the corresponding source code in the editor.

---

## 7.5 Source Code Advisor events

Events flagged by the Source Code Advisor are journal entries produced by FDPR. Each journal entry includes information about the performance problem and a general approach for solving it. It also includes suggested changes for compiler or linker command-line flags, and suggested source code changes.

The Source Code Advisor supports the following FDPR journal entries.

Table 11. SCA events

Event	Problem description
INLINE FUNCTION	High overhead for frequent calls to a small function
UNROLL LOOP	High branch penalty in a small loop
DIRECT TOC ACCESS	Data cache pressure is caused by TOC-load instructions
REDUCE FOR EARLY EXIT	High overhead for function which very frequently returns quickly (for example, saving registers that are not changed before returning)
MOVE HOT CODE TO COLD AREA	Invariant or infrequently executed code found within a loop
FIX LOAD-HIT-STORE	A data store operation followed closely by a load from the same address causes the load to take extra time to complete
DE-VIRTUALIZE FUNCTION CALL	Indirect function calls (virtual functions or by function pointers) have higher overhead than direct function calls
SHORTCUT PLT CALL	Call-through-PLT to local procedure has high call overhead. A call to a function that is contained in the same object is routed indirectly through the Procedure Linkage Table (PLT). This is normal, to allow for symbols being overridden within the process. However, it is more expensive than a direct call.
EXTSW INSTRUCTION	A frequently encountered conversion of a signed 32-bit integer type (int) to a signed 64-bit integer type (long), possibly implicitly, requires an extra "sign extension" instruction that can also delay subsequent dependent instructions
FMRTXXLOR	High latency of the FMR/FPR instruction
XSCPSGNDPXXLOR	High latency of the Xscpsgndp instruction
TARGET TO SOURCE DEPENDENT INSTRUCTIONS	Sequence of dependent instructions is not optimal
INT TO FLOAT CONVERSION	High penalty for int to float conversion
LOADING FROM A CONST AREA	High penalty for loading from a const area
BRANCH PREDICTION	High penalty for wrong branch prediction
KILLED REGISTERS	High penalty for stores and restores of registers that are killed (overwritten) after frequently executed function calls
LINK REGISTER OPTIMIZATION	High penalty for saves and restores of the link register in frequently executed functions
TOC STORE IN LOOP OPTIMIZATION	High penalty for TOC store in loop operation

## 7.6 Optimizing executable code using FDPR

Feedback Directed Program Restructuring (FDPR) allows you to perform feedback-directed optimization of an executable or shared library. Also known as the post-link optimization tool, FDPR optimizes the executable image of a program based on a typical profile.

FDPR optimization is performed in three distinct steps:

- **Instrumentation:** FDPR analyzes the input program and creates an instrumented version and an empty profile.
- **Profiling:** The instrumented program is run with some representative input. During this run, the profile is filled with various counts, such as how many times each branch was executed.

- Optimization: FDPR processes the input program together with the now filled profile. It performs various optimizations based on this profile, such as code restructuring, making the program run more efficiently.

The IBM Software Development Kit for Linux on Power allows you to run FDPR through the FDPR plug-in.

## 7.6.1 Specifying FDPR optimization for a project

You can specify workload and optimization flags for FDPR for a project.

### About this task

To specify workload and optimization flags for FDPR, complete the following steps.

#### Procedure

1. In the Project Explorer pane, right-click the project name. Click **Run as > Run Configurations**.
2. In the Profile Configurations window, right click in **FDPR** and click **New** to create a configuration profile.
3. On the Main tab, click **Browse** from the **C/C++ Application** field to select the binary file to be profiled.
4. On the Main tab, click **Browse** from the **Working directory** field to select the folder to be used by the profiling tool.
5. On the Arguments tab, specify the workload to be used to form the execution profile. It is important to select a representative workload so that the following optimization step is effective. Typically the same workload should be used for both SCA and for FDPR.
6. On the FDPR Options tab, select appropriate options.
  - The Optimized binary section specifies the component of the application that is being optimized. Select the main executable program or a shared library.
  - The Optimization level section allows you to specify the optimization flags used by FDPR. Select from options such as **O**, **O2**, **O3**, and **O4**. Or you can specify **Other** to specify specific options.
  - The **Architecture** field controls architecture-specific optimizations, like code alignment. Select a specific architecture, such as **power8** or **power7**.
  - The Info output section lists options to control the informational output. Select the maximum level of warnings printed to the console and whether to print progress messages to the console.
  - The Start/continue profile and program section lists options for profiling or optimizing a running server application. This section includes three radio button groups to control the way the profile and program are started or continued.
    - **Start program** (default) or **Continue program**: **Start program** causes the program to be restarted. The program is kept running if **Continue program** is selected.
    - **Start with empty profile** (default) or **Continue with last profile**: By default, the profile is cleared before actual profiling begins. If **Continue with last profile** is selected, the last profile collected is used to initialize the profile.
    - **Profile until completion** (default) or **Profile for \_\_\_ seconds**: By default, the profile is collected until the program completes. If the other option is selected, a number of seconds must be specified. The profile is collected only for that amount of time, and the program continues to run.

For example, some typical cases might include the following:

- a. Start the program with an empty profile and profile until the program terminates.
- b. Start the program with an empty profile. Profile for a specified number of seconds, and leave the program running.

- c. Begin with the previous case to start the program or server, and collect an initial profile. Then, continue the program with the existing profile and profile for a specified number of seconds. This case can be repeated multiple times, accumulating profile information across several time periods.
7. On the Environment tab, you can specify the following environment variables:
  - FDPR\_BINDIR**  
The directory where FDPR binary files were installed. Default: `/opt/ibm/fdprpro/bin`.
  - FDPR\_LIBDIR**  
The directory where FDPR libraries were installed. Default: `/opt/ibm/fdprpro/lib`.
  - FDPR\_PROF\_COMMAND**  
A user-provided script used to profile the program. The script is invoked as script *prog* arg *...*, where *prog* is the instrumented program or, when profiling a shared library, the original program. The default is to directly execute the program with the specified arguments.

8. Click **Run**.

**Related information:**

 [Feedback Directed Program Restructuring \(FDPR\)](#)



---

## 8 Automatic updates

The SDK provides an update site that allows consuming updates automatically.

By activating the automatic updates the SDK will periodically look for updates at the IBM public FTP located at `ftp://public.dhe.ibm.com/software/server/iplsdk/<SDK_Version>/repository`. The SDK will not collect or share any information about your environment and will not install anything without your authorization. Once an update is found you will be asked on how to proceed. You can review your decision at **Window > Preferences > Install/Update > Automatic Updates**.

**Note:** When installing the SDK, a group called `ibmsdklop` is created and the users whom have access to the graphical interface will be automatic added to it. Only the members of this group can perform the automatic updates of the SDK components.

### Adding a new user to the `ibmsdklop` group

To add a user to the `ibmsdklop` group, run `#usermod -aG ibmsdklop <USER>`.

### Granting access to the updates installed to all users

When an user installs an update for the SDK, the permissions of the files that are installed are assigned to that specific user. All other users are not able to access the newest installed resources. In order to make these resources available to all users, you must restore the correct permissions. Follow the instructions below:

```
#chgrp -R ibmsdklop /opt/ibm/ibm-sdk-lop
#chmod -R 775 /opt/ibm/ibm-sdk-lop
```



---

## 9 Support for IBM SDK for Linux on Power

This section provides information about troubleshooting problems with using IBM Software Development Kit for Linux on Power.

---

### 9.1 Getting customer support

The IBM Software Development Kit for Linux on Power is provided as is only. Customers are not entitled to IBM Software Support. However, you can get help from the Linux on Power Community.

#### Getting product updates

The IBM SDK for Linux on Power latest ISO image, related packages, and this user guide can be found at the IBM SDK for Linux on Power page (<http://www14.software.ibm.com/webapp/set2/sas/f/lopdiags/installtools/home.html>).

#### Getting help from the Linux on Power Community

You can submit questions and review technical information about the IBM SDK for Linux on Power in the IBM developerWorks Linux on Power Community. You can go directly to the Linux on Power Community page (<http://ibm.biz/BdxXrC2>), or go to IBM developerWorks (<https://www.ibm.com/developerworks/mydeveloperworks/>) and search for the “PowerLinux™ community group”.

You can also use IBM SDK for Linux on Power integrated bug reporting, which allows you to create a report that contains source code, error markers, and logs to be posted in the IBM developerWorks Linux on Power Community message board. See 9.2, “Using integrated bug reporting” for information.

---

### 9.2 Using integrated bug reporting

The IBM SDK for Linux on Power includes integrated bug reporting, which allows you to create a report that contains source code, error markers, and logs to be posted in the IBM developerWorks Linux on Power Community message board. You can include specifics about your question or problem.

#### 9.2.1 Creating a report

You can create a report that contains source code, error markers, and logs to be posted in the IBM developerWorks Linux on Power Community message board. The report also includes information about your system. You can include specifics about your question or problem.

#### About this task

You can create a report in three different ways. Choose from the following options.

#### Procedure

- Create a report about a code excerpt in your C/C++ project on which you have a question. Complete the following steps:
  1. Within your C/C++ project, select the code excerpt on which you have a question.
  2. Right-click the code excerpt and click **Ask for help in Linux on Power Community**. The Code Reports view is displayed.
  3. In the Code Reports view, expand **Code** to see the code report you created.
  4. To edit the report, double-click the report name. In the Report Editor window, type a name for the report and other details that you want to include in the report. Click **File > Save** or press **Ctrl+S**.

- Create a report about a problem that was flagged by IBM SDK for Linux on Power in the Problems view. Complete the following steps:
  1. Within the Problems view, select the problem for which you want to create a report.
  2. Right-click the problem and click **Ask for help in Linux on Power Community**. The Code Reports view is displayed.
  3. In the Code Reports view, expand **Problems** to see the problem report you created.
  4. To edit the report, double-click the report name. In the Report Editor window, type a name for the report and other details that you want to include in the report. Click **File > Save** or press **Ctrl+S**.
- Create an empty report to which you can add your own content. Complete the following steps.
  1. Open the Code Reports view, if it is not already open. To open the Code Reports view, click **Window > Show View > Reports**. The remaining steps take place in the Code Reports view.
  2. Optional: You can create the report in an existing report category, or in a new report category. To create a report category, click **+** (**Create a new Category**). Type a name for the category and click **OK**. The Code Reports view is updated to show the new category.
  3. Right-click the category and click **New Item**. The Report Editor window is displayed.
  4. In the Report Editor window, type a name for the report and other details that you want to include in the report. Click **File > Save** or press **Ctrl+S**.

## 9.2.2 Submitting a report to the Linux on Power Community

After you have created a report, you can use integrated bug reporting to submit the report to the IBM developerWorks Linux on Power Community message board.

### Before you begin

Ensure that you have an IBM ID and have joined the Linux on Power Community.

- To get an IBM ID, go to developerWorks registration.
- To join the Linux on Power Community, go to The Linux on Power Community, sign in, and click **Join this Group**.

### About this task

To submit a bug report that you have created, complete the following steps.

### Procedure

1. Open the Code Reports view, if it is not already open. To open the Code Reports view, click **Window > Show View > Reports**.
2. Expand the report category to see the report you want to submit.
3. Right-click the report and click **Ask for Help**. A warning is displayed to let you know that a report will be created including system data. Click **OK**.  
Your default browser opens with a new window to the Linux on Power Community message board.
4. If prompted, sign in with your IBM ID.
5. To submit the report, you must create a topic to contain the report. On the Linux on Power Community message board page, click **Start a topic**.
6. In the New Topic window, add your report as follows:
  - a. Type a topic name that describes your report.
  - b. In the content area, right-click and click **Paste** to add your report to the topic.
  - c. Click **OK**.
7. To subscribe to comments that are made by other Linux on Power Community members on your submission, use one of the **Feed** options at the bottom of the page.

---

## 9.3 Setting up SSH credentials

You can set up SSH credentials so that you can create a connection without a password.

### About this task

This method allows you to set up SSH with DSA public key authentication.

### Procedure

1. On your workstation, generate a DSA Key Pair. Log into the workstation and enter the following command:

```
ssh-keygen -t dsa
```

- a. When prompted to enter a file in which to save the key, press **Enter**.
- b. When prompted, specify a password, and then confirm it.

The private key is saved to `/home/user_name/.ssh/id_dsa`

2. Ensure that you have permissions for the `.ssh` directory. Enter the following commands:

```
cd  
chmod 755 .ssh
```

3. Copy the public key file from your workstation to the remote server as `~/.ssh/authorized_keys`. Enter the following command:

```
scp ~/.ssh/id_dsa.pub user@remote_server_name:~/.ssh/authorized_keys
```

4. On the remote server, ensure that you have permissions for the `.ssh/authorized_keys` file. Log into the remote server and enter the following command:

```
chmod 600 ~/.ssh/authorized_keys
```

### 9.3.1 Logging in from workstation to remote server with DSA key

#### About this task

You can use `scp` or `ssh` from the workstation to log in to the remote server. These methods still require you to enter the password you created.

### Procedure

- `ssh user@remote_server_name`
- `ssh user@remote-server.com`
- `scp file user@remote_server_name:/tmp`

### 9.3.2 Logging in with DSA key but no password

#### About this task

You can use a shell prompt to specify your password, and not be prompted to specify it again when using `ssh` or `scp`.

### Procedure

1. At a shell prompt, enter the following commands:

```
exec /usr/bin/ssh-agent $SHELL  
ssh-add
```

2. When you are prompted, enter your password. This sets your identity and allows you to avoid being prompted for passwords when using `ssh` or `scp`.



---

## 10 Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.





---

## Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Dept. LRAS/Bldg. 903  
11501 Burnet Road  
Austin, TX 78758-3400  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (<sup>®</sup> and <sup>™</sup>), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel, 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.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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





Printed in USA