

Copyright © 2006

Pakistan Software Export Board (G) Limited
Ministry of Information Technology
Government of Pakistan

Printing

Artland Communications, Lahore. September 2006

Published by

Pakistan Software Export Board

The Funding Agency

This open source toolkit is funded by the Open Source Resource Center (OSRC) project of the Pakistan Software Export Board (PSEB). PSEB is the entity within Government charged with the task of enhancing exports of software and IT enabled services (ITES) from Pakistan. PSEB is a guarantee limited company totally owned and funded by the Government of Pakistan. Any questions or comments about this toolkit may be directed to PSEB Islamabad at 92-51-111-333-666 or through e-mail at osrc@pseb.org.pk.

Disclaimer

This toolkit is published by the PSEB for members of the IT industry and the public-at-large. The toolkit's compilers, or the editor, are not responsible, in any way possible, for the errors/omissions of this toolkit. The OSRC does not accept any liability for any direct and consequential use of this toolkit or its contents. The contents of this toolkit may be distributed only subject to the terms and conditions set forth in the Open Publication License v 1.0 or later. The latest version is presently available at <http://opencontent.org/openpub/>

TABLE OF CONTENTS

INTRODUCTION.....	1
PHP: AN INTRODUCTION.....	2
1. WHAT IS PHP.....	3
1.1. History of PHP.....	3
1.2. PHP 4 Architecture.....	3
1.3. Language Syntax.....	3
2. DOWNLOADING AND INSTALLING PHP.....	4
3. EMBEDDING PHP CODE.....	5
4. VARIABLES.....	6
4.1. Arrays.....	6
4.2. Built-in Variables.....	7
4.2.1. PHP internal variables.....	7
4.2.2. CGI/Web server provided variables.....	7
4.2.3. HTTP Request Variables.....	8
5. CONDITIONALS AND LOOPING CONSTRUCTS.....	8
5.1. Conditionals.....	8
5.2. Loops.....	8
6. WEB APPLICATION FEATURES.....	8
7. WORKING WITH COOKIES.....	9
8. DATABASE HANDLING.....	9
8.1. Communication with Other Databases.....	10
9. REFERENCE.....	10
PYTHON: AN INTRODUCTION.....	11
1. DOWNLOADING AND INSTALLING PYTHON.....	12
1.1. Downloading Python.....	12
1.2. Installing Python.....	12
2. STARTING AND STOPPING PYTHON.....	12
3. YOUR FIRST PROGRAM.....	12
4. VARIABLES AND EXPRESSIONS.....	13
4.1. Expressions.....	13
4.2. Variable assignment.....	13
5. CONDITIONAL STATEMENTS.....	13
5.1. if-else.....	13
5.2. The pass statement.....	13
5.3. elif statement.....	13
5.4. Boolean expressions: and, or, not.....	14
6. BASIC TYPES (NUMBERS AND STRINGS).....	14
6.1. Numbers.....	14
6.2. Strings.....	14
7. BASIC TYPES (LISTS).....	14
7.1. Lists of Arbitrary Objects.....	14
7.2. List Manipulation.....	14
8. BASIC TYPES (TUPLES).....	14
8.1. Tuples.....	14
8.2. Tuple Manipulation.....	14
9. BASIC TYPES (DICTIONARIES).....	15
9.1. Dictionaries (Associative Arrays).....	15
9.2. Dictionary Access.....	15
10. LOOPS.....	15
10.1. The while statement.....	15
10.2. The for statement (loops over members of a sequence).....	15
11. FUNCTIONS.....	15
12. CLASSES.....	16
12.1. The class statement.....	16
12.2. Using a class.....	16
13. EXCEPTIONS.....	16
13.1. The try statement.....	16

13.2. The raise statement.....	16
13.3. Uncaught exception.....	16
14. FILES.....	16
14.1. The open() function.....	16
14.2. Reading and writing data.....	16
14.3. Formatted I/O.....	16
15. MODULES.....	17
15.1. Large programs can be broken into modules.....	17
15.2. The import statement.....	17
PERL: AN INTRODUCTION.....	18
1. INSTALLATION.....	19
2. RUNNING A PERL PROGRAM.....	19
2.1. First PERL Program.....	19
3. FUNCTIONS AND STATEMENTS.....	19
4. USING NUMBERS AND STRINGS IN PERL.....	20
5. VARIABLES.....	20
6. LOOPS.....	21
8. REFERENCES.....	22
ECLIPSE: AN INTRODUCTION.....	23
1. DOWNLOADING AND INSTALLING ECLIPSE.....	24
1.1. Downloading Eclipse.....	24
1.2. Installing Eclipse.....	24
2. STARTING AND STOPPING ECLIPSE.....	24
3. YOUR FIRST JAVA PROJECT.....	25
3.1 Setting up the environment.....	25
3.2 Creating a new project.....	26
3.3 Creating the project's name.....	26
3.4 Creating the project's settings.....	27
4. CREATING A SIMPLE JAVA APPLICATION.....	27
4.1 Creating a Java Class.....	28
4.2. Editing and compiling.....	29
4.3. Running the application.....	29
4.4. Getting Help.....	29
5. FURTHER TOPICS.....	30
6. REFERENCES.....	30
JEDIT: AN INTRODUCTION.....	31
1. DOWNLOADING AND INSTALLING JEDIT.....	32
1.1. Downloading jEdit.....	32
1.2. Installing jEdit.....	32
2. STARTING AND STOPPING JEDIT.....	32
3. JEDIT GUI.....	33
4. DEVELOPMENT USING JEDIT.....	33
4.1. Opening and closing files.....	33
4.2. Java development.....	35
5. REFERENCES.....	35
KDEVELOP.....	36
1. OVERVIEW.....	37
2.1.1. From your distribution.....	37
2.1.2. From Tarball.....	37
2.1.3. CVS HEAD.....	37
2.2. Lexicon.....	38
3. CREATING AN APPLICATION.....	38
3.1. Creating the framework with KDevelop.....	38
4. MAKE THE TRANSLATIONS FOR A SIMPLE KDE PROJECT.....	40
4.1. Install a gettext patched for KDE.....	40
4.2. Prepare the translations.....	40
4.3. Make the translations.....	40
4.4. Compile and install the translations files.....	41

5. A FEW GENERAL TIPS.....	41
5.1. <i>General hints</i>	41
5.2. <i>Coding practice</i>	41
5.3. <i>Importing your project in KDE CVS</i>	41
5.4. <i>How do I release my application as a tarball?</i>	42
6. REFERENCES.....	43
GNU COMPILER COLLECTION(GCC).....	44
1. INTRODUCTION.....	45
2. HOW TO COMPILE?.....	45
3. COMPILING MULTIPLE FILES AND LINKING.....	46
4. LINKING WITH EXTERNAL LIBRARIES.....	46
5. REFERENCES.....	47

Introduction

This open source toolkit has been developed by the Open Source Resource Center (OSRC), a project of the Ministry of Information Technology (MoIT). This toolkit contains step-by-step manuals related to open source applications for databases, application servers, desktop applications, office productivity suites, Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) software, and open source desktop applications for the Microsoft Windows platform. A set of CDs, including some Linux distributions and other applications, forms an integral part of this open source toolkit.

I would like to thank the OSRC team, including Mr. Abubakar Shoaib, Mr. Iftikhar Ahmad, Mr. Muhammad Hammmad, Mr. Muazzam Ali, Mr. Sher Shah Farooq, and Mr. Qandeel Aslam, who have compiled this toolkit; and Miss Seema Javed Amin, who has edited it. The OSRC would especially wish to thank PSEB's Director (Projects) Mr. Nasir Khan Afridi, Former Project Manger(OSRC) Mr. Osman Haq and Ministry of Information Technology's Member (IT) Mr. M. Tariq Badsha for their generous moral support, without which this toolkit would never have been completed.

This is the first edition of this toolkit, and the OSRC hopes to continue to improve it with the help of your feedback and comments.

Sufyan Kakakhel

Open Source Resource Center,
Pakistan Software Export Board,
2nd Floor, ETC, Agha Khan Road, F-5,
Islamabad, Pakistan.

Ph: +92-51-9208748

Fax: +92-51-9204075

Email: skakakhel@pseb.org.pk

<http://www.osrc.org.pk>

PHP: An Introduction

1. What is PHP

Taken directly from PHP's home, PHP.net, "PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly."

This is generally a good definition of PHP. However, it is often easier to think of PHP in terms of what it can do for you. PHP will allow you to:

- Reduce the time to create large websites.
- Create a customized user experience for visitors based on information that you have gathered from them.
- Open up thousands of possibilities for online tools.
- Allow creation of shopping carts for e-commerce websites.

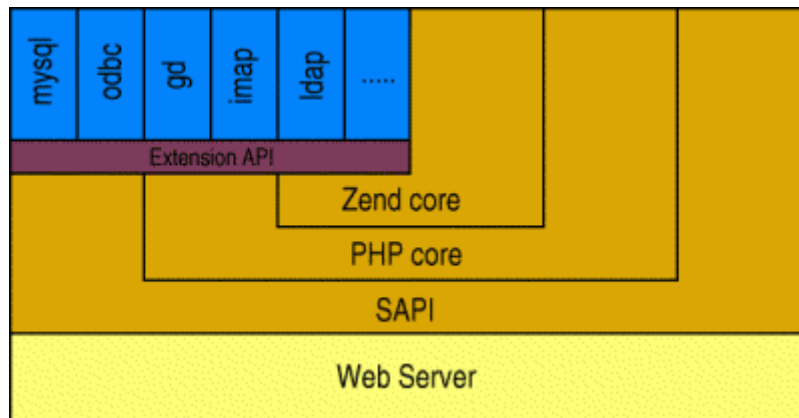
1.1. History of PHP

PHP started as a quick Perl hack written by Rasmus Lerdorf in late 1994. Over the next two to three years, it evolved into what we today know as PHP/FI 2.0. PHP/FI started to get a lot of users, but things didn't start flying until Zeev Suraski and Andi Gutmans suddenly came along with a new parser in the summer of 1997, leading to PHP 3.0. PHP 3.0 defined the syntax and semantics used in both versions 3 and 4.

1.2. PHP 4 Architecture

PHP has undergone major architectural changes since version 3 which are as follows:

- The language parser itself has become a self-contained component called The Zend Engine.
- PHP function modules, now called PHP extensions, are also basically self-contained.
- There is a Web server abstraction layer called SAPI that greatly simplifies the task of adding native support for new Web servers. SAPI also has the advantage of increasing PHP 4 stability, in addition to supporting multi-threaded Web servers.



All of PHP's functions are part of one of the layers with a side facing up in the architecture figure. Most functions such as the MySQL support are provided by an extension. Most extensions are optional. They can be linked into PHP at compile time or built as dynamically loadable extensions that can be loaded on demand.

1.3. Language Syntax

Major portion of the PHP syntax is taken/inherited from C, although there are some elements taken from Perl, C++ and Java.

2. Downloading and Installing PHP

1. **Download** the latest PHP sources PHP.tar.gz from php.net
2. **Extract the source code** to a directory under `/usr/local/src`

```
cp php-4.3.0.tar.gz /usr/local/src
cd /usr/local/src
gunzip php-4.3.0.tar.gz
tar -xvf php-4.3.0.tar
rm -f php-4.3.0.tar
cd php-4.3.0
```

3. **Set compiler options (optional)**

If you want you can set some compiler options, this is typically done to create optimized code. One very common thing to do is to set `CFLAGS=-O2` or `CFLAGS=-O3` (that's an Oh, not a Zero) that tells the compiler how much code optimization to do, setting it to a higher value does more optimization, but also takes longer to compile and may potentially cause unexpected things (not common). `O2` is a fairly safe level to use. To do this type the following:

```
export CFLAGS=-O2
```

4. **Configure php with autoconf**

Now you need to set the configuration options, and check that all libraries needed to compile are present. This is done with a script called `configure`, to find out what options you can set type the following:

```
./configure --help
```

You will see quite a few options, here's a [page](#) that defines the `configure` options. We will tell `configure` to enable `mysql`, and also tell it where to find `apxs` Apache's tool for building modules.

```
./configure --with-mysql --with-apxs2=/usr/local/apache2/bin/apxs
```

5. **Compile PHP**

```
make
```

6. **Install PHP**

```
make install
```

7. **Tell apache to load the module** Edit `httpd.conf` `/usr/local/apache2/conf/httpd.conf` with your text editor. Add the following to `httpd.conf`

```
Include conf.d/*.conf
```

This allows you to create a specific configuration file for each module that you install, for instance `php.conf`. Now create a directory in your apache directory if its not there called `conf.d`

```
mkdir /usr/local/apache2/conf.d
```

```
cd /usr/local/apache2/conf.d
```

Make a file called `php.conf` located at `/usr/local/apache2/conf.d/php.conf` with the contents:

```

# PHP Configuration for Apache

#
# Load the apache module
#
LoadModule php4_module modules/libphp4.so

#
# Cause the PHP interpreter handle files with a .php extension.
#
<Files *.php>
    SetOutputFilter PHP
    SetInputFilter PHP
    LimitRequestBody 9524288
</Files>

AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps

#
# Add index.php to the list of files that will be served as directory
# indexes.
#
DirectoryIndex index.php

```

Note: You could have just inserted the above in your httpd.conf file, and omit the conf.d step if you desire. I feel that the conf.d approach is a cleaner way to do it. That's it, restart apache and you should have PHP working.

3. Embedding PHP Code

To get an idea of what embedding PHP would entail, consider the following three "hello world" examples, all of which will give the similar output:

Example 1: HTML alone

Hello, World!

Example 2: PHP code alone

`<?php print "Hello, World!"; ?>`

Example 3: PHP embedded within HTML

`<?php print "Hello,"; ?> World!`

The file in which PHP embedded within HTML is interpreted provided the extension of the file is php (.php, .php3) etc. Web servers supporting PHP will, by default, scan a file in HTML mode. HTML code will be passed over to the browser as usual, up until the server finds a PHP line of code. In examples 2 and 3 above, the "`<?php`" tag informs the server that PHP code is to begin. The server then switches over to PHP mode and starts interpret PHP code and send it to client in form of HTML. The "`?>`" tag closes out the PHP mode with the server resuming its scanning in HTML mode once more.

Embedding code in this manner is, conceptually, a more fluid approach to designing a Web page because you are working within the output setting, namely an HTML page. Traditionally, you had to fragment the output (i.e. the header, body, footer etc...) and then put it into the code. Now we are inserting the code directly into the output.

Dynamic vs. Static Web pages

The "Hello, World" example we chose would certainly not require you to use PHP. That's because it is static, meaning its display will always remain the same. But what if you wanted to

greet the world in any number of ways? Say, for example, "Bonjour, World!", or "Yo, World!" and so on.

Since HTML tags are purely descriptive they cannot function as a variable. Nor can they convey even the simplest of uncertainty such as a "Bonjour" or a "Yo". You need a command language to handle variability in a Web page. Based on either a conditional statement or direct user input, a command language can generate the "static" HTML necessary to correctly display a Web page's content.

Let us reconsider example #3. This time we want to let the user decide how to greet the world:

Example 4: PHP embedded within HTML revisited!

```
<?php print $greeting, ", "; ?> World!
```

From the above example, \$greeting is assigned a value, and together with the comma and the word "World!", this value is sent to the browser.

Dynamic Web page design, however, is more than just about inserting variables. What if you wanted not only to greet the world in French, but also to present the page using the colors of the French flag?

Both a Web page's structure as well as its content can be customized. This means dynamic Web page programming can also entail on-demand Web page building.

4. Variables

In PHP, a variable does not require formal declaration. It will automatically be declared when a value is assigned to it. Variables are prefixed by a dollar sign: (\$VariableName).

Variables do not have declared types. A variable's type does not have to be fixed, meaning it can be changed over the variable's lifetime. The table below lists PHP's variable types:

Type	Description
Integer	integer number
Double	floating point number
bool1	Boolean (true or false), available from PHP 4.0
Array	hybrid of ordered array and associative array
object2	an object with properties and methods (not discussed in this article)

In the following example, four variables are automatically declared by assigning a value to them:

```
<?php
    $number = 5;
    $string1 = "this is a string\n";
    $string2 = 'this is another "string"';
    $real = 37.2;

?>
```

4.1. Arrays

PHP arrays are a cross between numbered arrays and associative arrays. This means that you can use the same syntax and functions to deal with either type of array, including arrays that are:

- Indexed from 0
- Indexed by strings
- Indexed with non-continuous numbers
- Indexed by a mix of numbers and strings

In the example below, three literal arrays are declared as follows:

- A numerically indexed array with indices running from 0 to 4.
- An associative array with string indices.
- A numerically indexed array, with indices running from 5 to 7.

For example,

```
<?php
$array1 = array(2, 3, 5, 7, 11);
$array2 = array("one" => 1, "two" => 2, "three" => 3);
$array3 = array(5 => "five", "six", "seven");
printf("7: %d, 1: %d, 'six': %s\n", $array1[3], $array2["one"], $array3[6]);
?>
```

From the above example, the indices in the array1 are implicit, while the indices in array2 are explicit. When specifically setting the index to a number N with the => operator, the next value has the index N+1 by default. Explicit indices do not have to be listed in sequential order. You can also mix numerical and string indexes, but it is not recommended.

4.2. Built-in Variables

PHP has a number of built-in variables that give you access to your Web server's CGI environment, form/cookie data and PHP internals. Here are some of the most useful variables:

4.2.1. PHP internal variables

The \$GLOBALS and \$PHP_SELF variables shown in the table below are specific to PHP:

Variable Name	Description
\$GLOBALS	An associative array of all global variables. This is the only variable in PHP that is available regardless of scope. You can access it anywhere without declaring it as a global first.
\$PHP_SELF	This is the current script, for example /~ssb/phpinfo.php3.

4.2.2. CGI/Web server provided variables

The variables listed below are derived from CGI protocols.

Note that the \$HTTP_*_VARS variables are available only when the "track_vars" directive is enabled. You can enable the directive by default when installing PHP with the "enable-track-vars" switch set to configure. Alternatively, you can set it in php.ini, or from your Web server's configuration.

Variable Name	Description
\$DOCUMENT_ROOT	Your Web server's base directory with user-visible files.
\$REQUEST_METHOD	The HTTP method used to access this page, for example GET or POST.
\$REQUEST_URI	Full local part of the request URL, including parameters.
\$HTTP_GET_VARS	An associative array with the GET parameters passed to PHP, if any.
\$HTTP_POST_VARS	An associative array with the POST parameters passed to PHP, if any.
\$HTTP_COOKIE_VARS	An associative array with the cookies passed by the browser, if any.
\$SCRIPT_FILENAME	File name of the top-level page being executed.
\$SCRIPT_NAME	Local URI part of the page being executed.
\$SERVER_ADMIN	Server administrator's email address.
\$SERVER_NAME	Domain name for the server.

<code>\$SERVER_PORT</code>	TCP port number the server runs on.
<code>\$SERVER_PROTOCOL</code>	Protocol used to access the page, for example "HTTP/1.1".

4.2.3. HTTP Request Variables

HTTP Request Variables are derived from their corresponding HTTP headers. For example, `$HTTP_USER_AGENT` is derived from the User-Agent header, presented in the table below:

<code>\$HTTP_HOST</code>	Host name in the browser's "location" field.
<code>\$HTTP_USER_AGENT</code>	User agent (browser) being used.
<code>\$HTTP_REFERER</code>	URL of the referring page.

5. Conditionals and Looping Constructs

PHP includes if and elseif conditionals, as well as while and for loops, all with syntax similar to C. The example below introduces these four constructs:

5.1. Conditionals

```
<?php
    if ($a) {
        print "a is true<BR>\n";
    } elseif ($b) {
        print "b is true<BR>\n";
    } else {
        print "neither a or b is true<BR>\n";
    }
}
```

5.2. Loops

```
do {
    $c = test_something();
} while ($c);

while ($d) {
    print "ok<BR>\n";
    $d = test_something();
}

for ($i = 0; $i < 10; $i++) {
    print "i=$i<BR>\n";
}

?>
```

6. Web Application Features

One of PHP's oldest features is the ability to make HTML form and cookie data available directly to the programmer. By default, any form entry creates a global PHP variable of the same name.

In the following example, a user name is retrieved and assigned to a variable. The name is then printed by the sub-routine "submit.php":

```
<FORM METHOD="GET" ACTION="submit.php">
```

```
What's your name? <INPUT NAME="myname" SIZE=3>
</FORM>
```

submit.php

```
<?php
print "Hello, $myname!";
?>
```

From the above example, note that variables can also be referenced from within double quoted strings.

7. Working With Cookies

You can set cookies in the browser from PHP using the `setcookie()` function. `setcookie()` adds headers to the HTTP response. Since headers must be inserted before the body, you will need to finish all of your `setcookie()` calls before any body output (usually HTML) is printed.

The following example uses a cookie to store a form value until your browser is terminated.

```
<?php

if (!$myname) {
    print "What is your name? ";
    print "<FORM ACTION=\"\$PHP_SELF\"METHOD=\"GET\">\n";
    print "<INPUT NAME=\"myname\" SIZE=20>\n";
    print "</FORM>";
    exit;
}

setcookie("myname", $myname);

?>
```

8. Database Handling

PHP and MySQL are often referred to as the "dynamic duo" of dynamic Web scripting. PHP and MySQL work very well together, in addition to the speed and features of each individual tool.

The following is a simple example of how to dump the contents of a MySQL table using PHP. The example assumes you have a MySQL user called "nobody" who can connect with an empty password from localhost:

MySQL Example

In the example below, PHP implements the following procedure:

- Connect to MySQL.
- Send a query.
- Print a table heading.
- Print table rows until end of the table has been reached.

```
<?php

//Connect to the database on the server's machine as
//user "Nobody".

$conn = mysql_connect("localhost", "nobody", "");
$res = mysql_query("SELECT * FROM mytable", $conn);
$header_printed = false;
print "<TABLE>\n";
do {
```

```

$data = mysql_fetch_array($res);

// Retrieve the next row of data.

if (!is_array($data)) {
    break;
}

// This part is done only on the first loop. It prints
// out the names of the fields as table headings. This
// ensures that the headings will only be printed if the
// database returns at least one row.

if (!$header_printed) {
    print " <TR>";
    reset($data);
    while (list($name, $value) = each($data)) {
        print " <TH>$name</TH>\n"
    }
    print " </TR>\n";
    $header_printed = true;
}
print " <TR>\n";
print " <TD>";

// Instead of looping through the returned data fields,
// we use implode to create a string of all data items
// with the required HTML between them.

print implode("</TD>\n <TD>", $data);
print " </TR>\n";
} while ($data);
print "</TABLE>\n";

?>

```

8.1. Communication with Other Databases

Unlike other scripting languages for Web page development, PHP is open-source, cross-platform, and offers excellent connectivity to most of today's common databases including Oracle, Sybase, MySQL, ODBC (and others). PHP also offers integration with various external libraries which enable the developer to do anything from generating PDF documents to parsing XML.

9. Reference

- The PHP Manual at <http://www.php.net/manual/>
- The PHP FAQ at <http://www.php.net/FAQ.php3>
- Support page (mailing lists and more) at <http://www.php.net/support.php3>

Python: An Introduction

1. Downloading and Installing Python

1.1. Downloading Python

Python is a portable, interpreted, object-oriented programming language. Python's latest source code and binaries are available from its official website, at <http://www.python.org>. These binaries are also available on Linux distribution CDs.

1.2. Installing Python

Python is generally installed when you install Linux with its default options. If it is not, then you can access the binaries' discussions from your Linux CDs.

Installing Python from its source distribution is also very simple. Follow these steps in order to build your own Python interpreter:

- Download and extract files, customizing build files (if applicable)
- run `./configure` script
- `make`
- `make install`

Python is generally installed in `/usr/local/bin` or `/usr/bin`, while the libraries are in `/usr/local/lib/python2.x` or `/usr/include/python2.x`, where the "2.x" is the version of Python you are using.

2. Starting and Stopping Python

```
unix % python
```

```
Python 2.2.3 (#1, Aug 8 2003, 08:44:02)  
[GCC 3.2.3 20030502 (Red Hat Linux 3.2.3-13)] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

Program Termination

Programs run until EOF is reached.
Type Control-D or Control-Z at the interactive prompt.
Or type

```
raise SystemExit
```

3. Your First Program

Hello World

```
>>> print "Hello World"  
Hello World  
>>>
```

Putting it in a file

Edit a file named `hello.py` and write

```
# hello.py  
print "Hello World"
```

into this file.

Running a file

```
unix % python hello.py
```

Or you can use the `#!` trick

```
#!/usr/bin/python  
print "Hello World"
```

In UNIX, it is also possible to automatically launch the Python interpreter without explicitly invoking it from the command-line. If you are using any UNIX-flavored system, you can use the shell-launching ("sh-bang") first line of your program:

```
#!/usr/local/bin/python
```

The "file path," i.e., the part that follows the "#!," is the full path location of the Python interpreter. It is usually installed in /usr/local/bin or /usr/bin. Pathnames that are incorrect will result in the "Command not found" error message.

As a preferred alternative, many UNIX systems have a command known as "env", either installed in /bin or /usr/bin, that will look for the Python interpreter in your path. If you have "env", your startup line can be converted into something like this:

```
#!/usr/bin/env python
```

4. Variables and Expressions

4.1. Expressions

Standard mathematical operators work like other languages:

```
3 + 5
3 + (5*4)
3 ** 2
'Hello' + 'World'
```

4.2. Variable assignment

```
a = 4 << 3
b = a * 4.5
c = (a+b)/2.5
a = "Hello World"
```

Variables are dynamically typed (No explicit typing, types may change during execution). Variables are simply names for an object. They are not tied to a memory location, as in C.

5. Conditional Statements

5.1. if-else

```
# Compute maximum (z) of a and b
if a < b:
    z = b
else:
    z = a
```

5.2. The pass statement

```
if a < b:
    pass                # Do nothing
else:
    z = a
```

Notes:

- Indentation used to denote bodies.
- pass used to denote an empty body.
- There is no '?' operator.

5.3. elif statement

```
if a == '+':
```

```

    op = PLUS
elif a == '-':
    op = MINUS
elif a == '*':
    op = MULTIPLY
else:
    op = UNKNOWN

```

Note: There is no switch statement.

5.4. Boolean expressions: and, or, not

```

if b >= a and b <= c:
    print "b is between a and c"
if not (b < a or b > c):
    print "b is still between a and c"

```

6. Basic Types (Numbers and Strings)

6.1. Numbers

```

a = 3                # Integer
b = 4.5             # Floating point
c = 517288833333L  # Long integer (arbitrary precision)
d = 4 + 3j          # Complex (imaginary) number

```

6.2. Strings

```

a = 'Hello'         # Single quotes
b = "World"        # Double quotes
c = "Bob said 'hey there.'" # A mix of both
d = """A triple quoted string

```

```

can span multiple lines
like this """
e = """Also works for double quotes"""

```

7. Basic Types (Lists)

7.1. Lists of Arbitrary Objects

```

a = [2, 3, 4]       # A list of integers
b = [2, 7, 3.5, "Hello"] # A mixed list
c = []              # An empty list
d = [2, [a,b]]     # A list containing a list
e = a + b           # Join two lists

```

7.2. List Manipulation

```

x = a[1]            # Get 2nd element (0 is first)
y = b[1:3]          # Return a sublist
z = d[1][0][2]     # Nested lists
b[0] = 42           # Change an element

```

8. Basic Types (Tuples)

8.1. Tuples

```

f = (2,3,4,5)      # A tuple of integers
g = (,)            # An empty tuple
h = (2, [3,4], (10,11,12)) # A tuple containing mixed objects

```

8.2. Tuple Manipulation

```

x = f[1]           # Element access. x = 3
y = f[1:3]         # Slices. y = (3,4)

```

```
z = h[1][1] # Nesting. Z = 4
```

Comments

Tuples are like lists, but **their** size is fixed at **the** time of creation.
Can't replace members (said to be "immutable")

9. Basic Types (Dictionaries)

9.1. Dictionaries (Associative Arrays)

```
a = { } # An empty dictionary
b = { 'x': 3, 'y': 4 }
c = { 'uid': 105,
      'login': 'beazley',
      'name': 'David Beazley'
    }
```

9.2. Dictionary Access

```
u = c['uid'] # Get an element
c['shell'] = "/bin/sh" # Set an element
if c.has_key("directory"): # Check for presence of a member
    d = c['directory']
else:
    d = None
d = c.get("directory",None) # Same thing, more compact
```

10. Loops

10.1. The while statement

```
while a < b:
    # Do something
    a = a + 1
```

10.2. The for statement (loops over members of a sequence)

```
for i in [3, 4, 10, 25]:
    print i
# Print characters one at a time
for c in "Hello World":
    print c
# Loop over a range of numbers
for i in range(0,100):
    print i
```

11. Functions

The def statement

```
# Return the remainder of a/b
def remainder(a,b):
    q = a/b
    r = a - q*b
    return r
# Now use it
a = remainder(42,5) # a = 2
```

Returning multiple values

```
def divide(a,b):
    q = a/b
    r = a - q*b
    return q,r
x,y = divide(42,5) # x = 8, y = 2
```

12. Classes

12.1. The class statement

```
class Account:
    def __init__(self, initial):
        self.balance = initial
    def deposit(self, amt):
        self.balance = self.balance + amt
    def withdraw(self,amt):
        self.balance = self.balance - amt
    def getbalance(self):
        return self.balance
```

12.2. Using a class

```
a = Account(1000.00)
a.deposit(550.23)
a.deposit(100)
a.withdraw(50)
print a.getbalance()
```

13. Exceptions

13.1. The try statement

```
try:
    f = open("foo")
except IOError:
    print "Couldn't open 'foo'. Sorry."
```

13.2. The raise statement

```
def factorial(n):
    if n < 0:
        raise ValueError,"Expected non-negative number"
    if (n <= 1): return 1
    else: return n*factorial(n-1)
```

13.3. Uncaught exception

```
>>> factorial(-1)
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in factorial
ValueError: Expected non-negative number
>>>
```

14. Files

14.1. The open() function

```
f = open("foo","w")           # Open a file for writing
g = open("bar","r")           # Open a file for reading
```

14.2. Reading and writing data

```
f.write("Hello World")
data = g.read()               # Read all data
line = g.readline()           # Read a single line
lines = g.readlines()         # Read data as a list of lines
```

14.3. Formatted I/O

Use the % operator for strings (works like C printf)
for i in range(0,10):

```
f.write("2 times %d = %d\n" % (i, 2*i))
```

15. Modules

15.1. Large programs can be broken into modules

```
# numbers.py
def divide(a,b):
    q = a/b
    r = a - q*b
    return q,r
def gcd(x,y):
    g = y
    while x > 0:
        g = x
        x = y % x
        y = g
    return g
```

15.2. The import statement

```
import numbers
x,y = numbers.divide(42,5)
n = numbers.gcd(7291823, 5683)
import creates a namespace and executes a file.
```

PERL: An Introduction

PERL is an acronym for "Practical Extraction and Report Language". It was developed by Larry Wall (author of the USENET newsreader). PERL is a stable, cross-platform programming language because of its easy manipulation of files and process information. Its syntax is similar to shell programs and C. PERL has the built-in functions of awk, sed, and grep. PERL is Open Source software, licensed under its Artistic License, or the GNU General Public License (GPL).

PERL's database integration interface supports third-party databases including Oracle, Sybase, PostgreSQL, MySQL, etc. PERL works with HTML, XML, and other mark-up languages. It also supports both procedural and object-oriented programming. PERL can be embedded into web servers to speed up processing by as much as 2000%. It is also equipped to handle encrypted Web data, including e-commerce transactions.

1. Installation

PERL is usually pre-installed in most Linux distributions. Type `perl -v` at the command line to find out which version has been installed. To install PERL, follow the step-by-step instructions given below:

1. The current version of PERL is 5.8.8. It can be downloaded from <http://search.cpan.org/CPAN/authors/id/N/NW/NWCLARK/perl-5.8.8.tar.gz>
2. Unpack the distribution:
`# tar zxvf perl-5.8.8.tar.gz`
`# cd perl-5.8.8`
3. Prepare PERL for compilation:
`./configure.gnu --prefix=/usr`
4. Run make to compile the package:
`# make`
5. Install PERL by running the following command:
`# make install`

2. Running a PERL Program

To run a PERL program from the command line, type:

```
# perl program.pl
```

Alternatively, write the following as the first line of your script:

```
#!/usr/bin/env perl
```

... and run the script as `/path/to/program.pl`

2.1. First PERL Program

A PERL script or program consists of one or more statements. These statements are simply written in the script in a straightforward fashion. PERL statements end in a semi-colon. To test your ability to store and run a PERL program, enter and execute the following code as an example:

```
# /usr/local/bin/perl -w  
print "Hello World!\n";
```

Comments begin with a hash symbol and run to the end of the line. You can either use parentheses for functions' arguments, or omit them according to your requirements.

```
print ("Hello World\n");
```

Save these statements in `first.pl`, and run the file with the PERL interpreter with the following command:

```
perl first.pl
```

3. Functions and Statements

PERL has many functions and libraries. The list of built-in functions is on the main page,

perfunc. The PERL functions take parameters. A PERL program consists of statements. The statement ends with a semicolon. The program consists of many lines or single line statements, separated by a semicolon.

The most common PERL function "print" is described below.

```
print "Hello world single statement";  
print "Look, ", "a ", "list!";
```

4. Using Numbers and Strings in PERL

There are two main data types in PERL numbers and strings. The number is written as a numerical, 1000. The string is written in single quotation marks, '1000'. Use the number without commas and spaces in PERL. Don't write it as 1,000 or 1 000.

The literals can be written as 'One thousand'. There is a difference between a double quote string and a single quote string. The double quote string is used in print statements. The single quote string is used for literals or assignment purposes.

5. Variables

PERL has three types of variables: *scalars*, *arrays* and *hashes*.

Scalars variables represent single things. This might be a number or a string. The name of a scalar begins with a dollar sign, such as \$k or \$fruit. You assign a value to a scalar by telling PERL what it equates to:

```
$k = 5;  
$fruit = 'apple';  
$string123 = "This is some string";
```

You can assign the number as a string or as a numeric. The conversion between the string and the numeric is done by PERL itself.

You can also use variables within the variables:

```
$fruit_number = 4;  
$number_show = "There are $fruit_number apples.";  
print "The report is: $number_show\n";
```

The final output from this code is:

```
The report is: There are 5 apples.
```

You can use variables in simple mathematical calculations like:

```
$a = 5;  
$b = $a + 10; # $b is now equal to 15.  
$c = $b * 10; # $c is now equal to 150.  
$a = $a - 1; # $a is now 4, and algebra teachers are cringing.
```

You can also use special operators, such as ++, --, +=, -=, /= and *=.

```
$a = 5;  
$a++; # $a is now 6; we added 1 to it.  
$a += 10; # Now it's 16; we added 10.  
$a /= 2; # And divided it by 2, so it's 8.
```

You can also concatenate the **strings**:

```
$a = "8"; # Note the quotes. $a is a string.  
$b = $a + "1"; # "1" is a string too.  
$c = $a . "1"; # But $b and $c have different values!
```

PERL converts strings to numbers whenever it is needed, so the value of b is 9 because PERL first converts them into numbers. The value of \$b is the number 9 and the value of \$c is 81

because of concatenation.

Note: The plus sign adds the numbers and the period sign concatenates the string.

In PERL, arrays begin with @. The length of the array starts with 0; if there is nothing in the array then -1 is returned.

```
@numbers_array = (1,2,3,4,5,6)
@months = ("Jan", "Feb", "Mar");
```

You can get the length of an array:

```
$length = $#numbers_array;
```

Hashes are just like hashtables. The hashes are **declared** as % in the beginning:

```
%student_numbers = ( "Jonn"=> 1, "Marry" => 2, "Petter" => 3 );
```

You can get the value from hash by reggerint to \$name{key}

```
print $student_numbers {"Jonn"};
```

You can also set the value again in the **hash**:

```
$student_numbers {"Jonn"} = 4;
```

Comments:

The # sign represents the comment:

```
# These words are in comments
```

6. Loops

The loops are used for flow control.

The simple loop in PERL is declared as:

```
for $k (1, 2, 3, 4, 5) {
    print "$k\n";
}
```

This loop prints the numbers from 1 to 5.

The loop can also work with range values:

```
@range_ten = (1 .. 10);
$last_limit = 25;
for $k (@range_ten, 13, 21 .. $last_limit) {
    print "$k\n";
}
```

You can also use the loop with arrays and **hashes**:

```
for $k (keys %student_has) {
    print "The roll number of student $k is $student_has {$k} value.\n";
}
```

7. Comprehensive PERL Archive Network (CPAN)

You can download any PERL module and documentation available online from the Comprehensive PERL Archive Network (CPAN).

The command to use CPAN is

```
$ perl -MCPAN -e COMMAND
```

Use the command **shell**:

```
$ perl -MCPAN -e shell
```

This will return you to a shell within PERL. Download any module you want. You will have to make some settings on the first time installation. If you want to install a bundle within PERL, for example, you can use a command, and then:

```
CPAN> install Bundle::Expect
```

8. References

- The PERL Directory: <http://www.perl.org/>

Eclipse: An Introduction

Eclipse is a popular Java and C++ development environment. Eclipse and other Eclipse-based tools are cost-effective, make application development easy, and reduce development time.

Eclipse is available for all popular platforms. Eclipse is used within the industry with most popular plug-ins that are available freely or on purchase.

The most popular plug-ins and tools available for Eclipse are:

Eclipse JBoss-IDE (open source) <http://www.jboss.com>

Eclipse tools (open source) <http://www.eclipse.org/tools>

Object web Lomboz Eclipse-based J2EE development environment (open source) <http://lomboz.objectweb.org>

MyEclipse plug-in for J2EE development (purchase)

1. Downloading and Installing Eclipse

1.1. Downloading Eclipse

Any version of Eclipse can be downloaded and installed from its official website at <http://www.eclipse.org/>

1.2. Installing Eclipse

Installing Eclipse is very simple. Eclipse is available in both .zip format and .tar.gz formats.

After downloading, extract “eclipse-SDK-3.1.1-linux-gtk.tar.gz” into your chosen location. Eclipse is now ready for use.

2. Starting and Stopping Eclipse

Prerequisites

Eclipse requires the Java Development Kit (JDK) in order to operate properly. Install JDK's version 1.5 into your system for Eclipse 3.1., and read JDK's requirement instructions for other versions.

Ensure that JAVA_HOME is set in ENVIRONMENT VARIABLES. Under Linux you can set Environment variables in /etc/profile file.

Starting

After extracting Eclipse and installing JDK, enter the Eclipse directory and type the following command:

```
./eclipse
```

This will launch the Eclipse workbench working environment.

Selecting Workspace

Workspace is the development environment's primary space, in which Eclipse physically builds its projects. All project-related information, such as the project's main directory, its source folder, and files, the project's compiled source and folders are placed in the workspace folder.

On starting Eclipse, a dialog box opens by default on its workspace folder; the folder's path is generally /root/workspace.

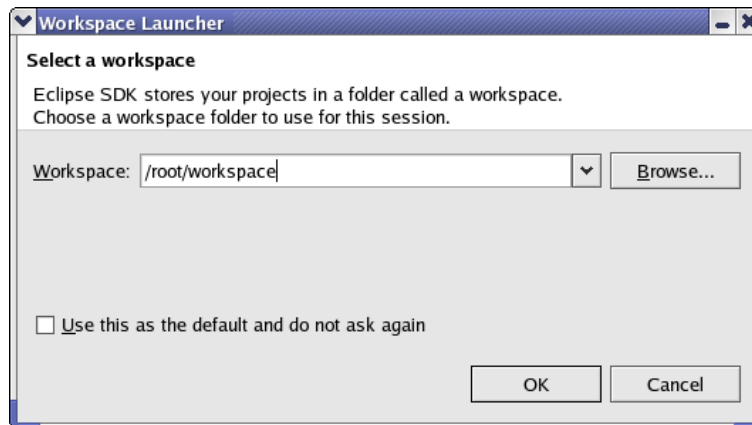


Fig 1.0 Selecting Workspace

Program Termination

Click the “X” (Close Window) button to exit from Eclipse. You can also close Eclipse from File -> Exit.

Eclipse GUI Perspective

Eclipse contains many different types of GUI styles tailored to meet specific requirements and developer modes.

Perspective is a GUI style that facilitates developers in making their projects. Eclipse Perspective generally includes a development window in the middle, project and package explorer on the left and right sides, and “Tasks”, “Problems” windows below. Eclipse opens Java Perspective by default. You can change the perspective by selecting Window -> Open Perspective -> Other and selecting any Eclipse perspective that suits your project’s requirements.

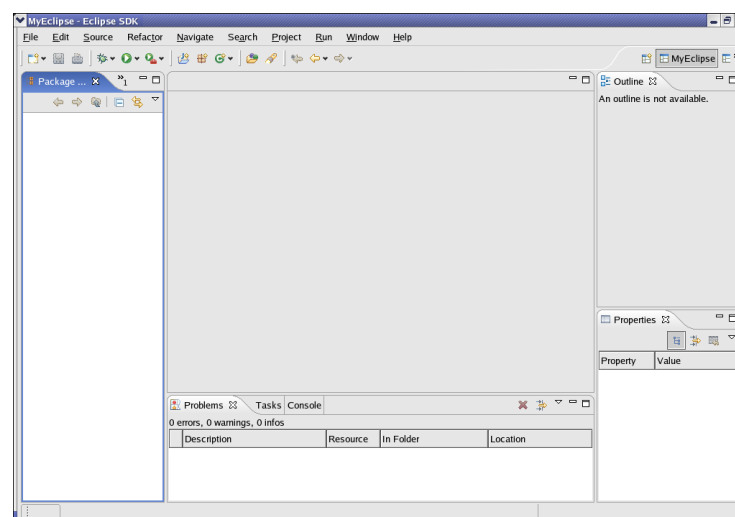


Fig 1.1 Eclipse Java Perspective

3. Your first Java Project

3.1 Setting up the environment

Ensure that JDK is properly installed and configured in Eclipse. Open Window -> Preferences. Select “Java”, and then select “Installed JRE’s”.

You can also set ANT home, CVS settings and other preferences from the Preferences settings tab.

3.2 Creating a new project

Select File -> New -> Project and select “Java Project”. The project creation wizard will guide you through a step-by-step process of preparing a Java project. You can also create other projects from this wizard.

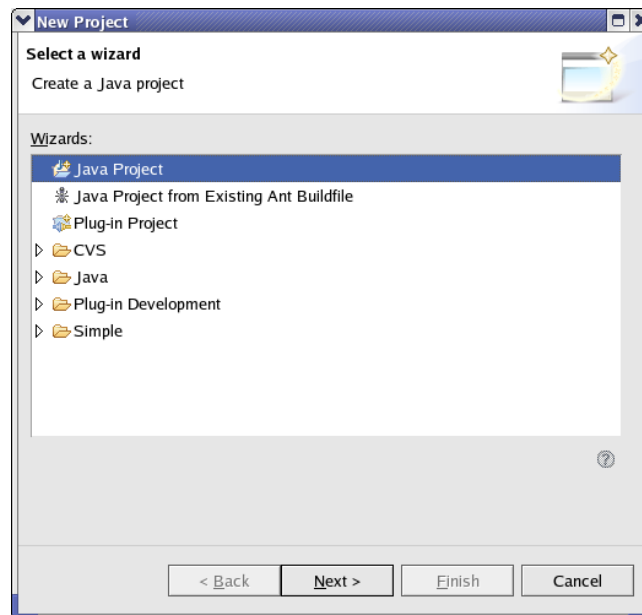


Fig 1.2 Java Project wizard

3.3 Creating the project's name

Enter the project's name, for example, “HelloWorld”. Click “Next”.

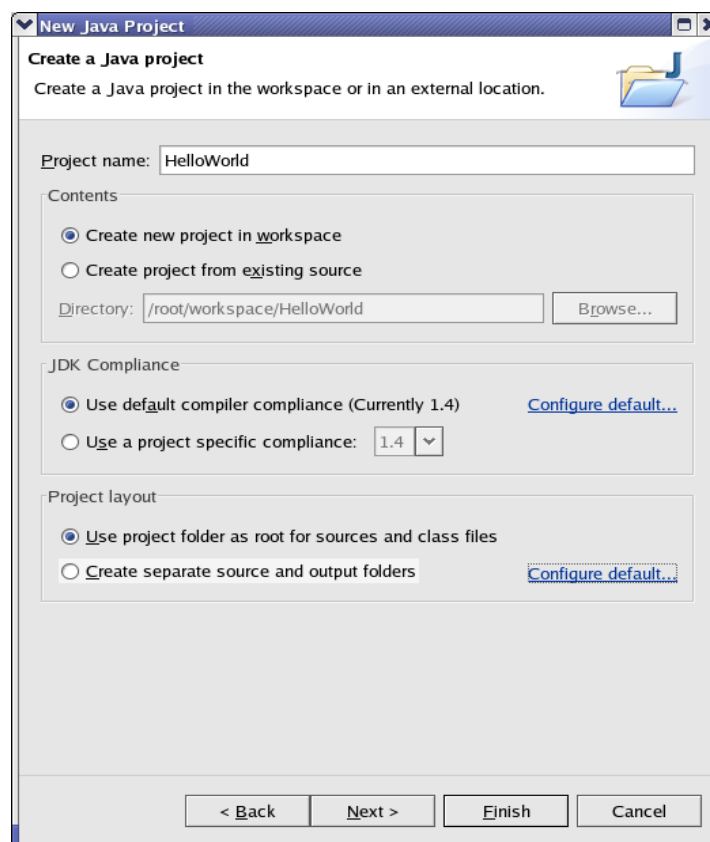
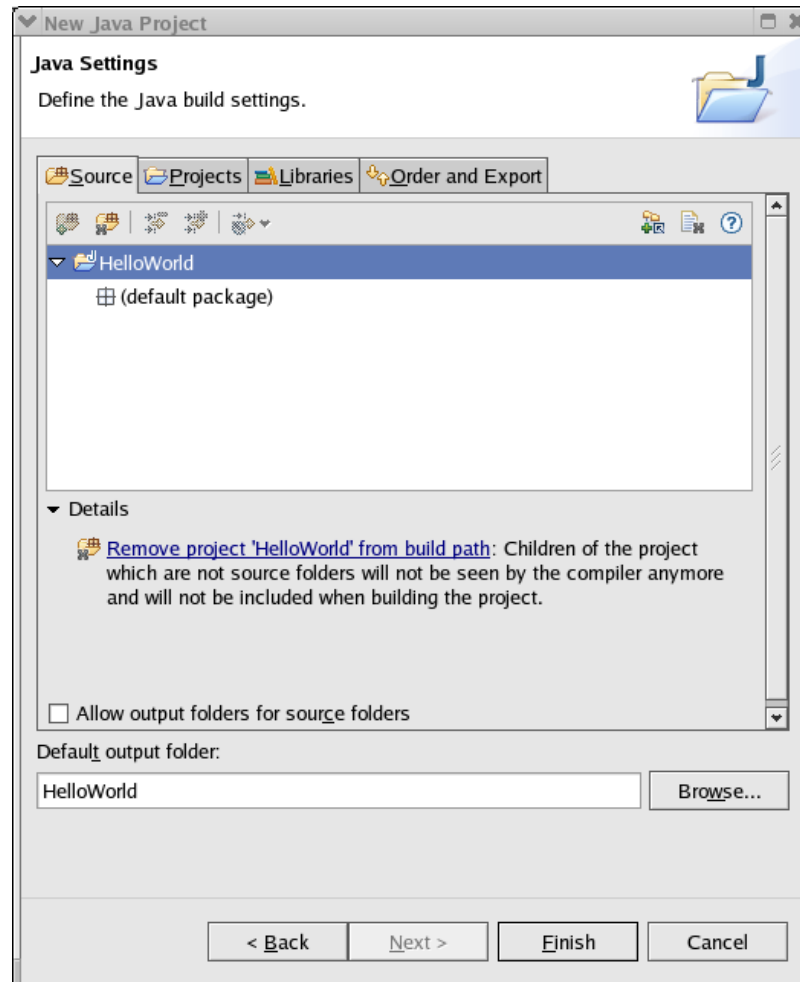


Fig 1.3 Enter project name

3.4 Creating the project's settings

You can change the project's settings by changing the project output folder; this is the folder that contains the compiled source.

The main feature that you can change is project libraries. If your project requires other libraries like `j2ee.jar` or `mysql.jar`, you can add them by giving their physical path. `j2ee.jar` is required for JSP/SERVLET (web project) development, and `mysql.jar` is required for MySQL database development. These are not provided by default with Java Development Kit (JDK) libraries.



Click “Finish”. Your Java project has been created, and Eclipse has been switched to Java Perspective.

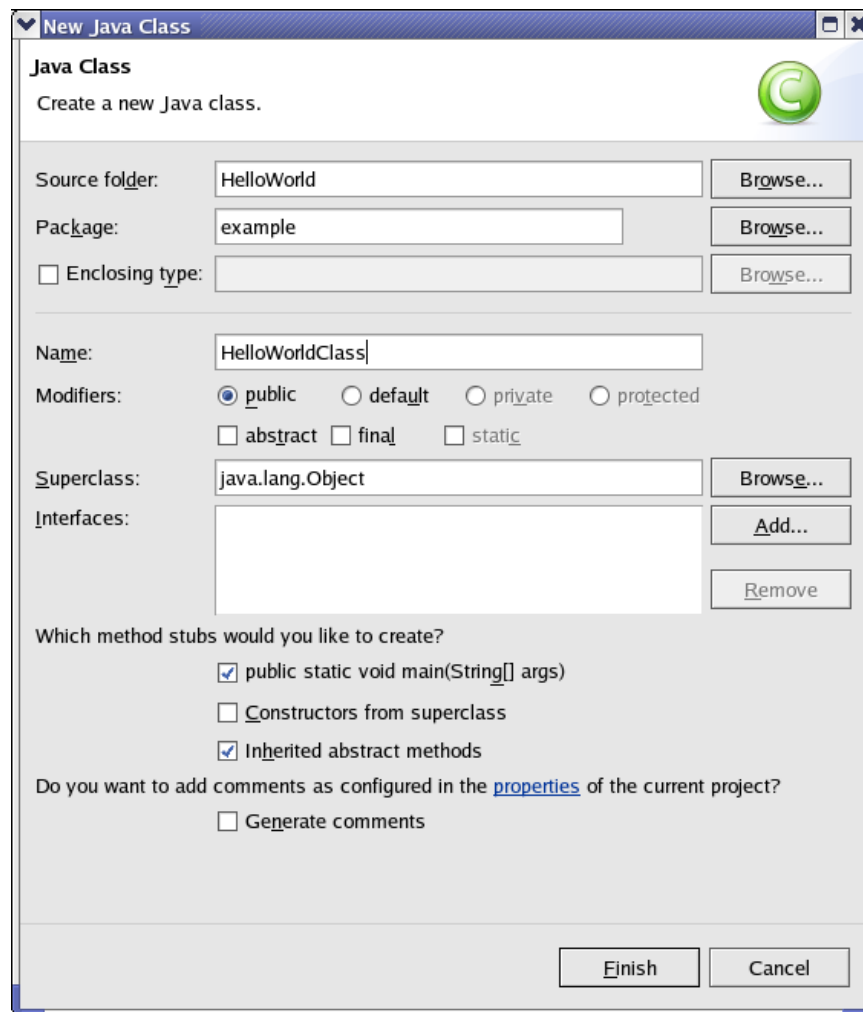
4. Creating a simple Java application

A simple Java application consists of a single Java class with main method printing Helloworld Eclipse.

The first step in this process is to create a Java class, and then compile and run this class as an application.

4.1 Creating a Java Class

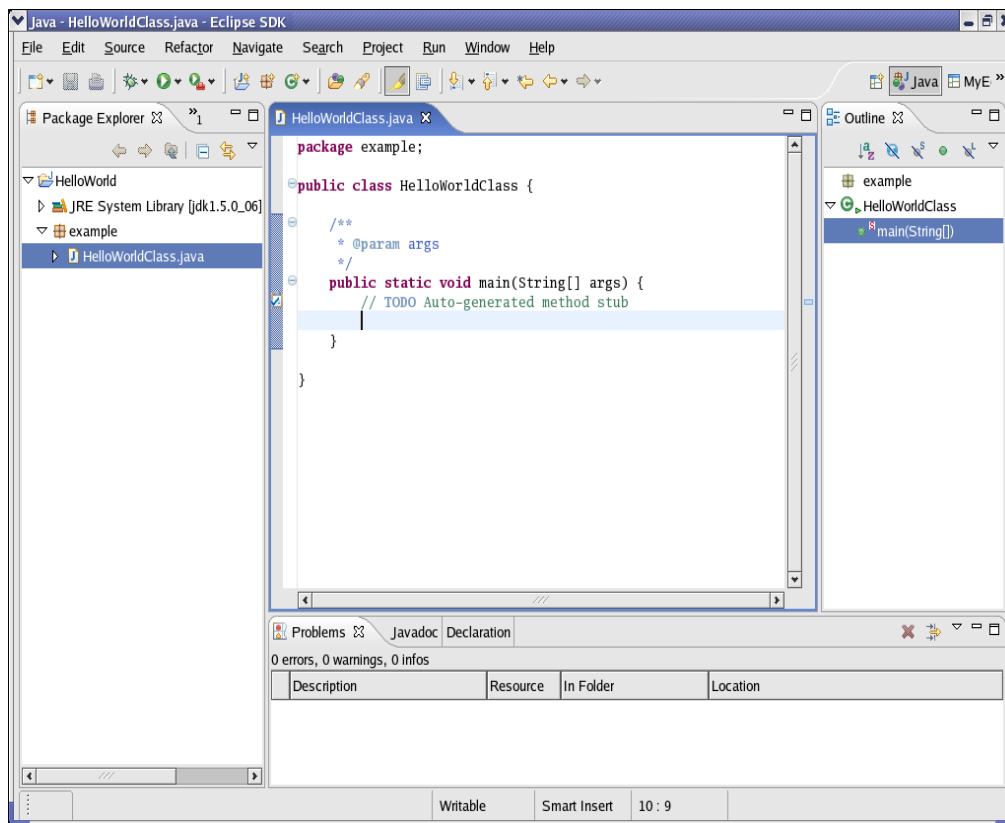
Select File -> New -> Class to open the Java class making wizard.



Fill in the specified fields:

Package: example
Name: HelloWorldClass
Select: public static void main(String[] args)
Click "Finish".

Your first Java class has now been created. You can edit it by opening it in the editor. To open the Java class, double click on the class name in Project Explorer.



4.2. Editing and compiling

Edit the Java class according to your application's requirement. You can, for example, write:

```
System.out.println("HelloWorld from eclipse");
```

into main method.

By saving the file, Eclipse automatically compiles Java class. You can enable and disable this option by selecting Project -> Build Automatically. Should you decide to disable this option, then you can manually build your project by selecting Project -> Build All.

4.3. Running the application

Select Run -> Run As -> Java Application

This will run "HelloWorldClass" as a Java application and output the results into a console window below.

4.4. Getting Help

The "Help" section in Eclipse contains a lot of information about Eclipse and Java development. Select Help -> Help Contents.

5. Further Topics

Building and deploying Enterprise Java applications is really quite simple. Object web Lomboz is also a very popular open source industry-based Eclipse tool for making J2EE and Java projects. The latest Java frameworks and technologies include:

- Servlet/JSP development
- EJB development
- Struts development
- Hibernate development
- JSF development

6. References

- Eclipse: <http://www.eclipse.org>
- JBoss: <http://www.jboss.com>
- Lomboz: <http://lomboz.objectweb.org>
- Java: <http://java.sun.com>
- Eclipse Help

jEdit: An Introduction

jEdit is a fully-functional Java-based text editor with a rich directory of plug-ins. You can install and easily get jEdit plug-ins, details of which are available from its official website at <http://www.jedit.org/>.

With the help of its plug-ins, jEdit can easily perform functions such as code-completion, project browsing, code formatting, source control integration, and much more.

jEdit is also helpful in HTML development. Its many features include spell-checking, code tidying, and tag insertion/completion. jEdit is not like other HTML-based editors such as Dreamweaver, for example. It is a user-friendly editor with much flexibility in code development.

1. Downloading and Installing jEdit

1.1. Downloading jEdit

jEdit is available for different operating systems, and the jEdit installer is available in the following formats:

- Java-based Installer
- Windows Installer
- Mac OS X packages
- Slackware packages

You can download the stable version according to your requirement. A Java-based installer is being used for the purposes of this tutorial. You can download the installer `jedit42install.jar` from <http://www.jedit.org/>.

1.2. Installing jEdit

Open the shell under Linux. Type the following command in the shell:

```
$ java -jar jedit42install.jar
```

This will display an Apache Software License screen. Please select your preferred location where the jEdit installer will install the editor.

Note: Please note that Java2 is a prerequisite for jEdit. JDK 1.5 is currently the latest version available at <http://java.sun.com/>

2. Starting and Stopping jEdit

Starting jEdit is quite simple. Type:

```
$ jedit &
```

Program Termination

Clicking on the Close window button will exit you from jEdit. You can also close it from File -> Exit.

Eclipse GUI Perspective

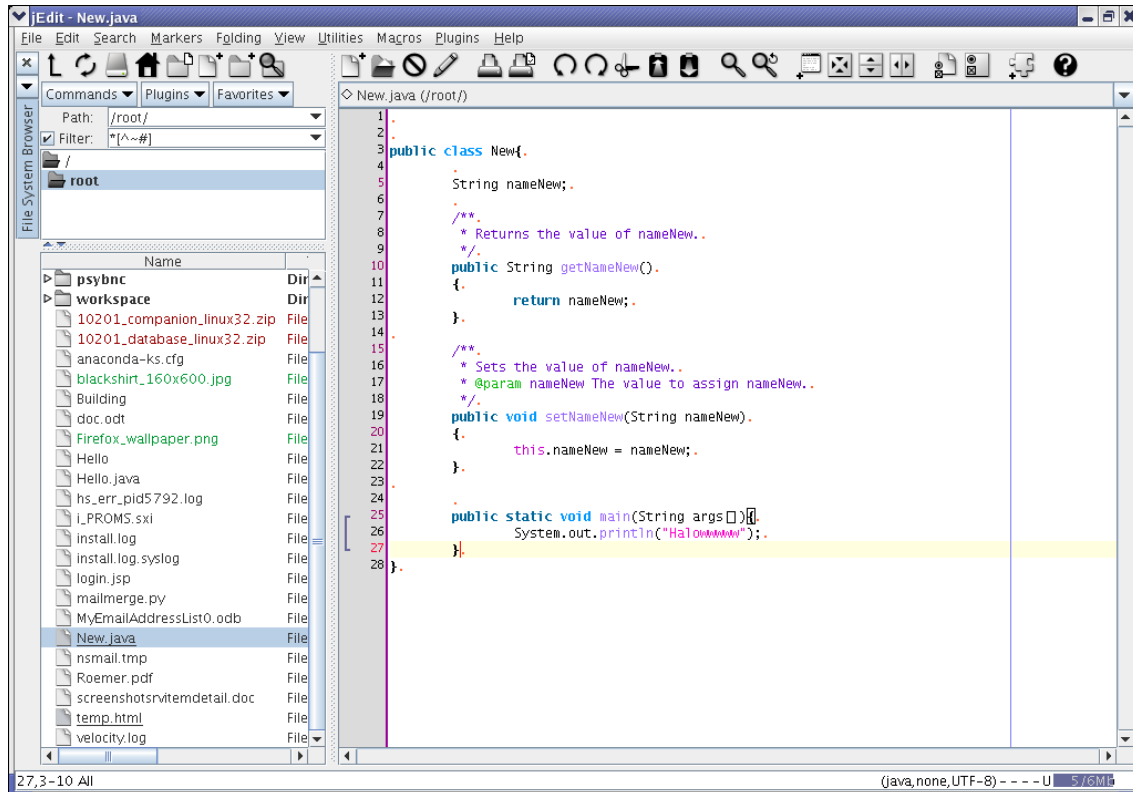
Eclipse has many different types of GUI styles according to your requirements and preferred developer mode of style.

Perspective is a GUI style that helps a developer make projects. Eclipse Perspective generally includes a development window in the middle, a project and package explorer on the right and left sides, and Tasks and Problems windows at the bottom. Eclipse opens Java Perspective by default. You can change the perspective by selecting Window -> Open Perspective -> Other and select any Eclipse Perspective that suits your project's requirements.

3. jEdit GUI

The main components of jEdit GUI include a working area in the middle, a toolbar at the top, and a File Browser on the left side. The working area in the middle is enabled by default. If you want to enable the File Browser, select the File System Browser option.

The main features of jEdit GUI are described below:



4. Development using jEdit

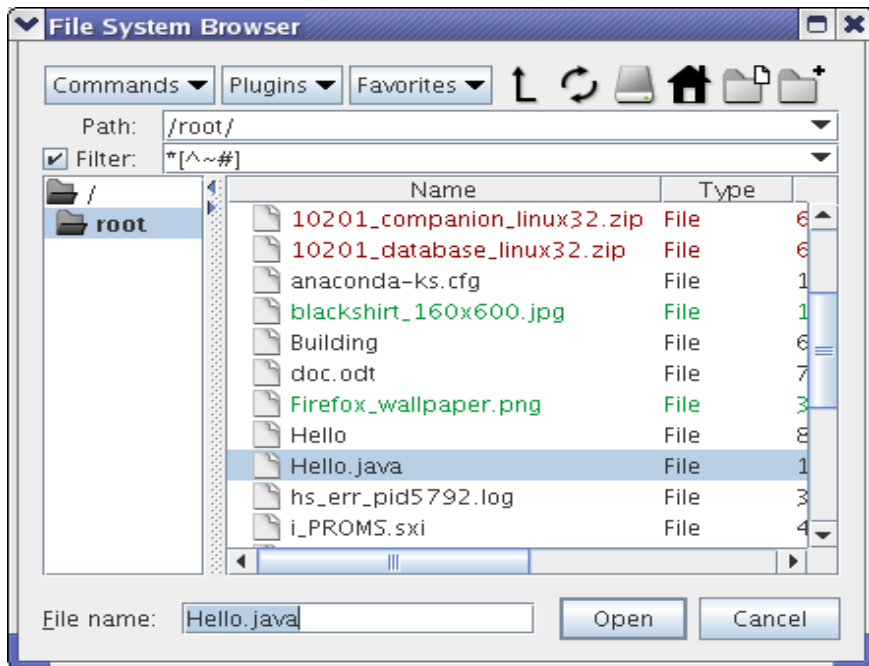
jEdit is an easy-to-use editor for quick development. If you need to type any file, you can add these files very quickly.

4.1. Opening and closing files

You can open and close the files from:

File -> Open

This will open a dialog box to open the file. The dialog box displays a directory window on the left side, from where you can select the directory. It also displays a File chooser window on the middle left side of the dialog box. Select the path and file, and the file will open in the editor window.

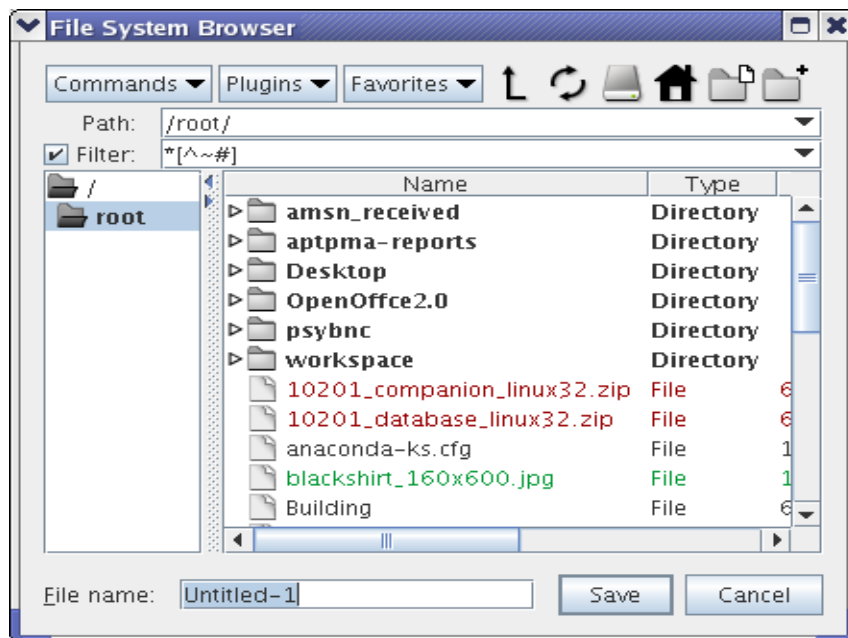


File -> New

Select File -> New. This will create a new file buffer.

File -> Save

To save the new file, jEdit will ask for a file name as it opens the File Save dialog box. Enter the name of the file and click Save.



File -> Close

Select File -> Close.

4.2. Java development

Open a new file and name this file with the .java extension.

Generating Getter and Setter

You can create getter and setter methods for any variable. Write a variable, select it, and type:

Macros -> Java -> Make Get and Set methods

This will create getter and setter methods for the chosen variable.

Other Java development options

Other options are also available in jEdit for Java development. You can select them by selecting the following option:

Macros -> Java -> Create Constructor
Get Class Name
Get Package Name
Java File Name
Make Get and Set methods
Preview JavaDoc of current buffer

Some of the options also require plug-ins, which are not installed in jEdit by default.

Further readings

jEdit has proved itself to be the best industry solution for small and easy editing options. Plug-ins are easily available, some of which operate with FTP, and perform remote server file opening and closing with easy uploading facilities.

5. References

- jEdit – Programmer's Text Editor: <http://www.jedit.org/>
- jEdit Help on the Internet

KDevelop

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation.

1. Overview

KDevelop is an open source and free Integrated Development Environment (IDE). It is a development tool for developing C/C++ applications and GUI-based applications for KDE. It is the most popular Linux desktop environment.

KDevelop is free software through the GNU General Public License (GPL). It is available from its official website at <http://www.kdevelop.org/>. Download any version you wish to install.

2. Requirements

2.1. How to get KDevelop 3

2.1.1. From your distribution

KDevelop 3 should be a part of your distribution. Pre-3 versions of KDevelop were nicknamed "Gideon", but they are now obsolete.

2.1.2. From Tarball

If you cannot find KDevelop 3 for your distribution, it can be downloaded from the KDevelop website in the "Downloads" section. At the time of writing, the latest KDevelop 3.0 source version is 3.0.4. Download the latest packages from <http://download.kde.org/download.php?url=stable/3.2.3/src/kdevelop-3.0.4.tar.bz2>

In order to compile KDevelop 3, you need Qt-3.1.0 or higher, and kdelibs-3.1.0 or higher. The environment variables QTDIR and KDEDIR should point to these directories.

Set up the KDE and Qt paths. The most common errors while using KDevelop come from the environment variables not having been set up properly. To see all your environment variables, check in a console by issuing the command 'set'. Set your PATH variable as well as your LD_LIBRARY_PATH as given below:

```
export PATH=$QTDIR/bin:$KDEDIR/bin:$PATH  
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
```

If you have any problems getting KDevelop to run, follow the forum link from the KDevelop website to find its solution. If you have never used KDevelop before, try to create a new project and to compile it in order to get used to its interface and icons.

2.1.3. CVS HEAD

To get KDevelop from CVS, you need Qt-3.2.3 or higher, and kdelibs-3.1.0 or higher:

```
mkdir KdevelopCVS  
cd KdevelopCVS  
export CVSROOT=:pserver:anonymous@anoncvs.kde.org:/home/kde  
cvs login  
(enter for password)  
cvs co kdevelop
```

When the download is finished, type:

```
make -f Makefile.cvs  
./configure --prefix='kde-config' --prefix'  
make  
make install (as root)
```

For more details, please visit http://www.kdevelop.org/index.html?filename=branches_compiling.html

2.2. Lexicon

Widget: A widget is an element of a graphical interface, such as a container window, a button or a field for entering text.

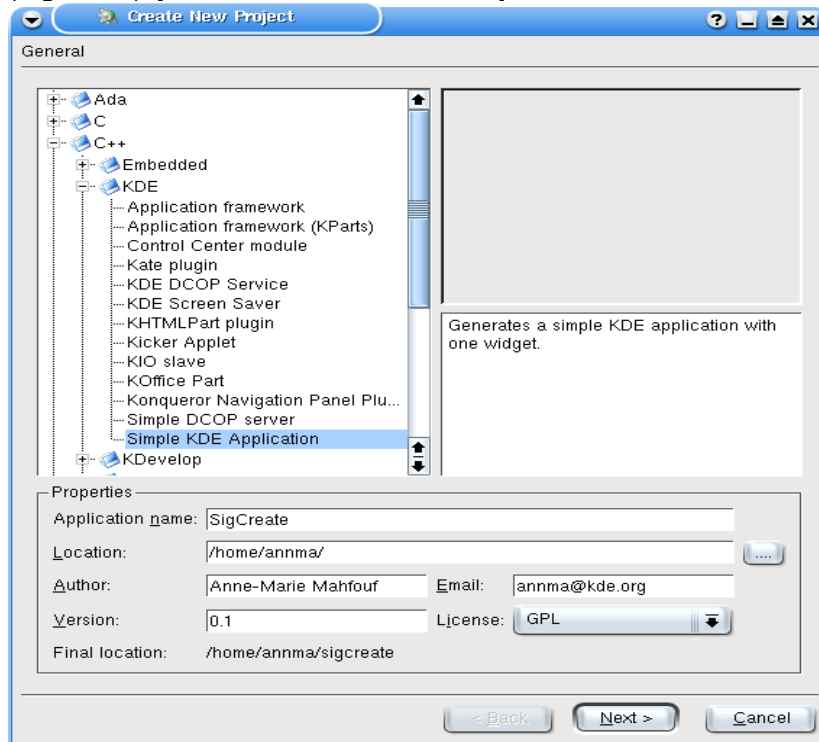
Layout management: This term describes the way in which widgets are arranged in a window. In its simplest form, an element may be placed at a specific position and given a specific height and width. But when the user resizes the window, the widgets should stay in their position and adjust their size accordingly. Linux allows users to do that by using layouts to place the widgets in.

Signals and slots: Signals and slots are used for communication between objects. The signal/slot mechanism is one of the central features of Qt. Objects emit signals when they change their state in a way that might be interesting to the outside world. Slots can be used for receiving signals, but they are normal member functions. You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you require. For more details, please visit <http://doc.trolltech.com/3.3/signalsandslots.html>. You can find the signals and the public slots that go with each class in the online documentation that comes with Qt. You can then implement your own slots.

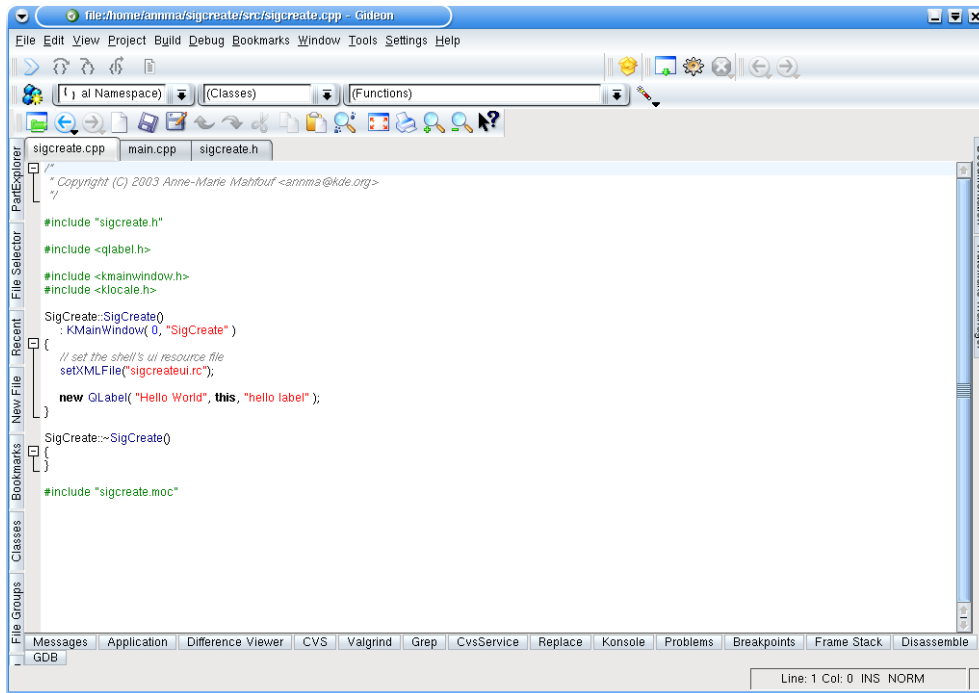
3. Creating an application

3.1. Creating the framework with KDevelop

The framework for the program (i.e. the main window) can be created quickly and easily by using KDevelop. Start KDevelop and select New Project in the Project menu. The Application Wizard is displayed. Choose C++ -> KDE -> Simple KDE Application. Write the project's name (SigCreate), your name as an author, and your e-mail address:



Click on Next, look at the CVS option and the header templates. Click on Finish at the last screen. KDevelop creates all the files that you need in order to compile your project. Use the file selector to view the code of the three files, which are main.cpp, sigcreate.cpp and sigcreate.h



Once the Application Wizard has created your application, compile it to ensure that everything is in order. To do that, select Build -> Run automake & friends then Build -> Run configure. The Messages output window should say:

```

" Good - your configure finished. Start make now
*
* *** Success ***

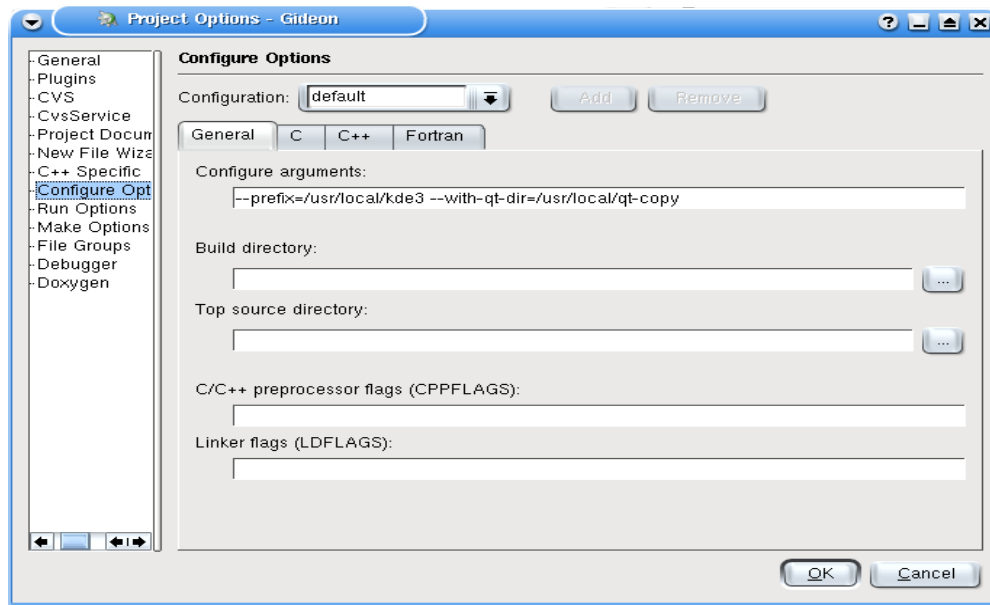
```

Run "make" with Build -> Build Project (or use the F8 shortcut). Then Build -> Install. Then Build -> Execute program (or F9). The result is given below:



The KDE Simple Application

The framework is complete. Prepare to start developing the program. If KDevelop does not recognize the QTDIR and KDEDIR variables, set them in Project -> Project Options in Configure Options.



Project Options

4. Make the translations for a simple KDE project

Once your own project is finished, you might like to have one or several translations for the GUI.

4.1. Install a gettext patched for KDE

Install a gettext patched that you can find on developer.kde.org and install it in your home directory. The patched gettext is available at <http://public.kde.org/planetmirror.com/pub/kde/devel/gettext-kde/>

```
$ tar xvfz gettext-0.10.35-kde.tar.gz
$ cd gettext-0.10.35-kde
$ ./configure
$ make
$ mkdir -p ~/bin
$ cp src/gettext src/xgettext ~/bin # copy gettext and xgettext into your HOME/bin directory
$ export PATH=~/bin:$PATH
```

4.2. Prepare the translations

Go into the project directory:

```
$ cd /path/to/myproject
```

Set KDEDIR to match your KDE installation - on Mandrake it is /usr. This path can also be found by typing "kde-config --prefix"

```
$ export KDEDIR=`kde-config --prefix` # or export KDEDIR=/usr
```

Create the translations' files:

```
$ make -f admin/Makefile.common package-messages
```

4.3. Make the translations

Translate the .po files using kbabel. These files are in the po directory of the project.

Warning

Do not touch the .pot file.

4.4. Compile and install the translations files

```
$ make package-messages
$ sudo make install
```

5. A few general tips

5.1. General hints

Application name

KDE application names generally start with the letter K, followed by a name suggesting the program's function. KMail, for example, informs the user that it is a KDE application, and pertains to e-mail. A good practice is to choose an English name for the name that follows the letter K. A good name can boost your program's popularity.

Note

Run a search in Google (in Konqueror, write `gg:your_app_name`) to confirm if the name you want to use is not already the name of a copyrighted program. If such is the case, or if in doubt, change it.

5.2. Coding practice

Comments should be made in the English language in order to facilitate other people studying your code.

Class names are usually also in the English language, and the names must indicate what the class does. Good examples include `TopLevel`, `CursorInterface`, and `TaskManager`. Note the uppercase letters, and remember that C++ is case-sensitive.

Function names usually begin with lowercase letters; some examples are given below:

```
void activateRaiseOrIconify();
void toDesktop(int);
void windowAdded(WId);
```

Code indentation can be whatever you like, as long as the code is visible for other developers. Always keep in mind that other people will wish to study your code. Try to make it easy for them to read it.

5.3. Importing your project in KDE CVS

Your program interests people, you feel you have time to really work on it, you need more feedback and help for improving it. You also agree to release it under the GPL license, or an equivalent. You can ask for a CVS account to import it in `kdenonbeta`. The `kdenonbeta` module is quite big and is not distributed with the official KDE. It is neither packaged, nor translated, and feature freeze does not apply to `kdenonbeta`. Its purpose is to allow other developers to work on your application and to test it. You will need a `Qt-copy`, `arts` and `kdelibs` from CVS HEAD. Please visit <http://women.kde.org/articles/tutorials/howtocvs.php> to learn more about how to compile KDE from CVS HEAD.

Note

When your application contains the most important features, when it is totally KDE-compatible (i18n, xml GUI, etc.), you will be able to ask about moving it in a KDE official package.

In order to get a CVS account, send an e-mail to `sysadmin (at) office (dot) kde (dot) org` to justify why you need CVS access. Inform them that you want to import your application (`app_name`) in the `kdenonbeta` module. Make sure to specify your full name and e-mail address. Choose a nickname for your user login. You can currently choose between the standard non-encrypted CVS protocol (`pserver`) and the encrypted CVS-over-ssh. If you choose `pserver`, also send an encrypted password (for instance using `useradd dummy; passwd dummy; grep dummy /etc/passwd /etc/shadow`). If you choose CVS-over-ssh, send a public ssh key (e.g. `~/ssh/id_dsa.pub`).

Wait for a reply from a KDE sysadmin.

Once you have compiled at least a Qt-copy, arts and kdelibs, check out kdenonbeta files. Log into the CVS server with your login user and password:

```
$ cvs co -I kdenonbeta  
$ cd kdenonbeta  
$ cvs co admin (or ln -s ../kde-common/admin .admin)
```

Copy your project's main dir with everything in kdenonbeta and then, in your project's main dir, issue a:

```
$ make disclean
```

All the .o files must be deleted. You can also manually remove all Makefile, Makefile.in (not Makefile.am) and all KDevelop-related files. Remove the admin, autom4te.cache, debug, doc, po and template folders. Keep some files and the src subdir. Type **cd ..** to go back in kdenonbeta and type:

```
$ make -f Makefile.cvs  
$ ./configure --prefix=$KDEDIR  
$ cd your_project_name  
$ make  
$ su -c 'make install'
```

If errors are made during this procedure, rectify them by carefully reading the error message. If you are unable to resolve the problem on your own, go to IRC and ask for help on #kde or #kde-devel.

In the kdenonbeta dir:

```
$ cvs add your_project_name  
$ cvs add your_project_name/*  
$ cvs add your_project_name/src  
$ cvs add your_project_name/src/*  
$ cvs commit
```

You will get the window (vi editor as default) where you can log your message. It is a good practice to note what your commit is about. In your case, you will say (type i first if you are in vi to get into edit mode):

```
First import of your_app_name which does this and that.
```

Check if all the files have been added correctly. If not, CVS add filename and CVS commit.

Each time you want to work on your project, do not forget to log in the KDE server with your user login and password and type:

```
$ cvs up
```

to make sure you have the latest version.

5.4. How do I release my application as a tarball?

Install the *kdesdk* module on your machine. In this module, *kdesdk/scripts/cvs2dist* is a script to extract an application from the KDE source tree, and package it as a stand-alone application.

From the command **cvs2dist --help**, get:

```
Usage: cvs2dist module directory [options] [addfile1] [addfile2] ...
```

Provided that your application is in the kdenonbeta module, **and** that your application name is KMyApp, the command you will issue is:

```
$ cvs2dist /path/to/your/cvs/source/kdenonbeta/ kmyapp -n kmyapp -v 0.1
```

where you replace kmyapp with your program's name, and 0.1 with your program's current version.

This will create two packages, kmyapp-0.1.tar.gz and kmyapp-0.1.tar.bz2, in the directory where you issued your command.

6. References

- KDevelop: <http://www.kdevelop.org/>
- The KDE Developer FAQ <http://developer.kde.org/documentation/other/developer-faq.html>
- KDE Application Developer's Checklist
- <http://developer.kde.org/documentation/other/checklist.html>
- How to get KDE3.x.x and KDE3.2 (from CVS HEAD) working on the same machine
- <http://women.kde.org/articles/tutorials/howtocvs.php>
- The KDE API reference (for CVS HEAD) <http://developer.kde.org/documentation/library/cvs-api/>
- The KDE User Interface Guidelines
- <http://developer.kde.org/documentation/standards/kde/style/basics/index.html>
- The online Qt reference <http://doc.trolltech.com/3.2/index.html>
- Qt Quartely: newsletters for C++ and Qt developers <http://doc.trolltech.com/qq/>
- The C++ FAQ Lite <http://www.parashift.com/c++-faq-lite/index.html>
- How to Implement "User Defined Settings" using KDE3 and Qt Designer 3
- http://www.kdevelop.org/doc/tutorial_settings/

GNU Compiler Collection (GCC)

1. Introduction

The GNU Compiler Collection is also known as the "GNU C Compiler" and "GCC". It is a complete suite of tools that includes compilers for C, C++, Fortran, and Java, etc. Compilers for languages, other than C, have their own names e.g. the compiler for C++ is g++. Officially, the name can be used for for any language, but it is conventionally used for C. This tutorial will refer to the term "GCC" as GNU C Compiler, unless stated otherwise.

The GCC was first released in 1987 as part of the GNU project. The objective of the GNU project was to develop a free UNIX-like operating system and related software. The GCC project became an important tool to develop other free software.

GCC currently supports three versions of the C standard, details of which can be found in the "References" section. GCC provides many options for normal C programs, in addition to other options to control the compiler, preprocessor, and an assembler, etc. Programmers rarely have to use such low-level options, and mostly deal with the normal options for C programs.

GCC offers a wide range of features:

- GCC is a portable compiler and supports many platforms, including microcontrollers.
- GCC supports *cross compilation* i.e. a program that can be compiled on one platform to produce an executable for another platform. This is particularly useful for embedded systems.

2. How to compile?

GCC compiles a C source code into an executable binary format. The following is a sample C program, *sample_prog.c*.

```
-----  
int main () {  
    printf("Hello World\n");  
    return 0;  
}  
-----
```

The following command can be used to compile the program:

```
$ gcc sample_prog.c
```

This will compile the source code file, *sample_prog.c*, and will produce an executable file *a.out*. It can be run as:

```
$ ./a.out
```

./ here refers to the current directory. This will produce the output:

```
Hello World
```

By default, if you do not specify the output filename, GCC will create *a.out*. **The output file name can be specified by using the *-o* switch:**

```
$ gcc sample_prog.c -o sample
```

This will create an executable file *sample*. It is important to note that if a file exists with the same name as an executable file, then it will be overwritten. Moreover, it is recommended to use the *-Wall* option, which will turn on the common compiler warnings. This will help in locating problems that can crash the program later, or produce incorrect results:

```
$ gcc -Wall sample_prog.c -o sample
```

To compile a C++ program use `g++`:

```
$ g++filename
```

Multiple source code files, for a program, can be compiled at once:

```
$ gcc source_file1.c source_file2.c -o output
```

3. Compiling Multiple Files and Linking

A source code file can be compiled to produce an *object* file instead of creating an executable directly. This is useful when a program is split into multiple source files, and the modification in one of the source files requires the recompilation of that specific file only. Later, those object files can be linked separately. The following command will create object files `sample_prog1.o` and `sample_prog2.o`.

```
$ gcc -c sample_prog1.c  
$ gcc -c sample_prog2.c
```

These files can be linked together to produce an executable as follows:

```
$ gcc sample_prog1.o sample_prog2.o -o sample
```

Suppose that `sample_prog1.c` has now changed and a `sample` executable needs to be rebuilt. First, recompile `sample_prog1.c` to create a new object file, and then relink it to the `sample_prog2.o` object file to produce a new executable. Do not recompile `sample_prog2.c`, since this file has not changed:

```
$ gcc -c sample_prog1.c  
$ gcc sample_prog1.o sample_prog2.o -o sample
```

4. Linking with External Libraries

External libraries can be either *static* or *dynamic*. The difference between the two is that in the former, an actual executable file contains machine code from both the program object file(s) and the object files of any external library functions used by the program. In the latter, an actual executable file just contains references to external library functions, in addition to machine code of program object file(s). Libraries are linked at run-time in the case of *dynamic* libraries, which result in smaller executable files. Static libraries and dynamic libraries are stored in files with extensions `.a` and `.so` respectively.

The standard system libraries, both static and dynamic, can be found under `"/usr/local/lib/"` and `"/usr/lib/"` and the corresponding standard system header files are located under `"/usr/local/include/"` and `"/usr/include/"`. For example, a standard C library is stored in `"/usr/lib/libc.a"`, a math library can be found in `"/usr/lib/libm.a"` and an SSL library is stored under `"/usr/lib/libssl.a"`. If you want to run a program that contains, for instance, a reference to an external function in the SSL library, then it can be compiled and linked as:

```
$ gcc sample_ssl_prog.c /usr/lib/libssl.so
```

or for a math library:

```
$ gcc sample_math_prog.c /usr/lib/libm.a
```

A better way to link against libraries is to use the `-l` option:

```
$ gcc sample_ssl_prog.c -lssl  
$ gcc sample_math_prog.c -lm
```

The option `-I` will attempt to find a library file `libNAME.a` in the standard library directories. By default, whenever the `-I` option is given, GCC will first attempt to find an alternative shared library with the same name and the `.so` extension and will be given preference over `libNAME.a`.

If the header files and libraries are not in standard paths, then GCC will give a compilation and/or linking error.

```
-----  
Header files  
  
/usr/local/include/  
/usr/include/  
  
Libraries  
  
/usr/local/lib/  
/usr/lib/  
  
-----
```

By default, GCC searches the header files and libraries in the standard directories in a specific order i.e. from first to last. In case a header file is found in both directories i.e. `"/usr/local/include/"` and `"/usr/include/"`, the former path will take precedence. In order to search for header files and libraries in other paths, GCC provides the `-I` and `-L` option.

```
$ gcc sample_prog.c -I/home/osrc/include -L/home/osrc/lib
```

As stated earlier, a program using a shared library creates an executable that contains references to external library functions. On executing the program, the machine code for those functions must be loaded into the memory. This requires that the path to shared libraries must either be a standard system libraries path, or explicitly set in the `LD_LIBRARY_PATH` environmental variable:

```
$ gcc sample_prog.c -I/home/osrc/include -L/home/osrc/lib -o sample  
$ LD_LIBRARY_PATH=/home/osrc/lib  
$ export LD_LIBRARY_PATH  
$ ./sample
```

GCC provides many other options to optimize and customize compilation. A detailed overview of GCC is out of the scope of this document. Further information can be found in the references given below.

5. References

- GCC Home Page – GNU Project – Free Software Foundation: <http://gcc.gnu.org/>
- An Introduction to GCC: <http://www.network-theory.co.uk/docs/gccintro/index.html>