[**Editor's Note**: The following excerpt is from Chapter 1 of the free eBook *The Developer Shortcut Guide to SUSE Linux* (Realtimepublishers) written by John Featherly and available at http://cc.realtimepublishers.com/portal.aspx?pubid=339.]

Chapter 1: Using SUSE Linux Professional

Application development has a long and interesting history. From the switchboard plugs that programmed the first computers to today's Web applications, a lot has happened. Academic discussions ebb and flow around topics such as software design, architecture, and methodologies in a quest to determine whether programming is an engineering discipline or an art or neither.

Adding to the many facets of this discussion is the open source movement—one of the most interesting things happening in application development today. Hardly a newcomer, open source had its formal roots planted 30 years ago by Richard Stallman at MIT. Software has a dual nature of being both a representation of an algorithmic idea and a creative product not unlike a musical or theatrical recording. Open source addresses these two aspects in a clear and straightforward manner. The algorithmic ideas are free and "open" in that their representation is the source code and that source code is available to anyone. The creative product is "produced" by a community of contributors that include management as well as technical contributions. This community is almost always virtual, global, and open to any interested person.

This guide is targeted at experienced developers who are looking to get a quick start on writing open source-*based* enterprise applications. These are developers writing custom corporate applications used for unique business solutions in contrast to applications written for general use. This guide is not necessarily targeted at developers looking to get involved and contribute to open source projects (or possibly start an open source project).

Chapter 1 will take a look at the SUSE Linux Professional product features with special attention to application development components. First, we'll look at the desktop environments GNOME and KDE, then we will explore application packages and tools. We will be looking at two major application technologies in parallel throughout the guide: Java and .NET; this chapter introduces the components used for Java and .NET development on Linux. Finally, we'll take a look at configuring an SUSE workstation for application development.

SUSE Linux Professional

SUSE has put together a great distribution with SUSE Linux Professional—an outstanding and extensive collection of components shipped on a single bootable DVD. Both 32-bit and 64-bit versions are included on the 8.5G DVD. Sources are included on a second 4.7G DVD. Five CD's are included containing the 32-bit version if you are not able to install using a DVD drive.

The two main desktop environments; GNOME and KDE are included as well as a selection of simple window managers. More than just operating system (OS) components, the distribution also contains office productivity applications, graphics tools, browsers, email, database systems, and of course development tools.

Desktop Environments

Visual icons for folders and documents that create a representation of a physical desktop on the computer screen are the basis of the user interface (UI) for all operating systems (OSs) that use a GUI. The term *desktop environment* refers to three distinct parts: the UI design, the bundle of user support applications (such as file managers, text editors, and Web browsers), and—of interest to new application developers—the window controls, events, and widgets programming model. Although there are certainly variations on the UI design from one OS to the next, the basis for all of them is the windows, icon, mouse, pointer (WIMP) paradigm.

Linux and most UNIX OSs with a graphical UI (GUI) use *X Window* software as the foundation of their graphics system. X Window provides an *X Server* that runs where the graphics hardware exists (usually considered a "client" machine) and serves graphics capabilities to *X client* programs via the *X protocol*. The X client program typically runs on the local machine where the X Server is running but that is not required. The X protocol can traverse the network and the X client program can run on a remote machine and use display services of the local X Server. In order to be usable, an X Window system needs a program called a *window manager* and a set of user graphics elements traditionally called widgets. The Athena widget set from Project Athena at MIT was the first available (not surprisingly, as X Window was created at Project Athena). The other two significant widget sets are OSF/Motif and Sun Microsystems OpenLook. The two popular Linux open source projects, GNU Network Object Model Environment (GNOME) and Kool Desktop Environment (KDE), provide widget sets and window managers to fill the role of Motif or OpenLook. For example, the GIMP Toolkit (GTK) is the widget set used by the GNOME desktop. GNOME and KDE provide a complete desktop environment comprising UI design, applications, and a programming toolkit.

GNOME

If you selected the predefined system with GNOME during your initial system installation, GNOME will be available as a session type at login. If you did not select GNOME at installation, you can simply add it. There are two GNOME selections in YaST: GNOME System and GNOME Development.

YaST is the SUSE setup tool, which we will explore in more detail later in the chapter.

The combined list of GNOME components is contained in the GNOME YaST Package Group System/GUI/GNOME. Normally you will install the GNOME System selection if you want to use GNOME and the GNOME Development selection if you want to develop GNOME applications.

Although there are many applications, the important applications for everyday use in GNOME are the Nautilus file manager, the Evolution email and calendar application, the GIMP image editing application (a la PhotoShop), and the Epiphany Web browser.

Consult the GNOME Desktop User Guide at http://www.gnome.org/learn/users-guide/latest/ for an extensive description of how to use GNOME.

The current default GNOME desktop user layout has a bit of a Macintosh flavor, as Figure 1.1 shows. The default menu bar is along the top of the screen, and dynamically mounted disk drives such as USB drives create a drive icon on the desktop.

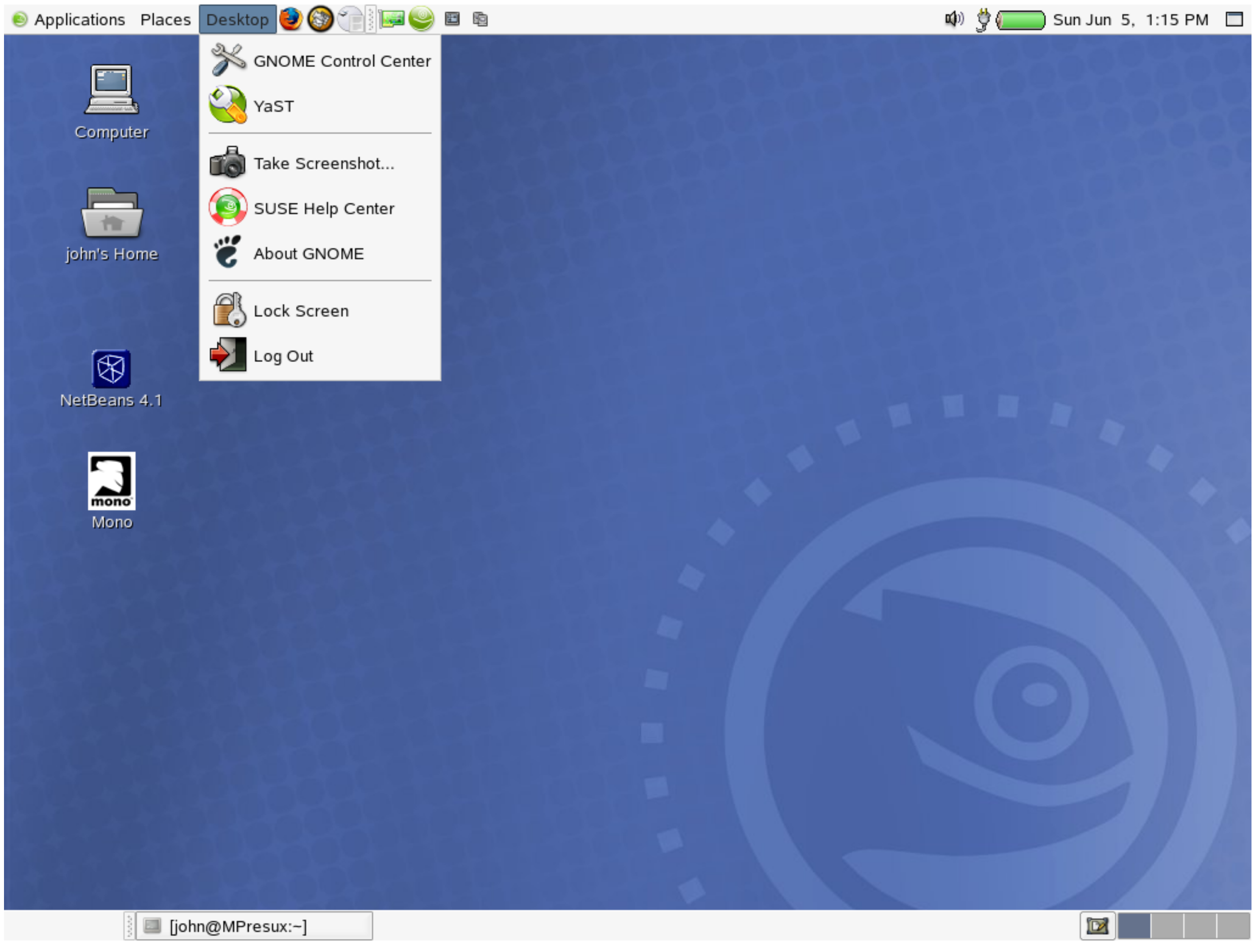


Figure 1.1: The GNOME desktop environment.

Check out http://primates.ximian.com/~miguel/gnome-history.html for the history of GNOME from founder Miguel de Icaza.

KDE

KDE is a desktop environment with a strong set of applications as a core part of the project. Figure 1.2 shows the KDE desktop, which has more of a Microsoft Windows–type layout of menu bars, icons, and application menus. The KDE file manager, Konqueror, is also the KDE Web browser and document viewer application. This unified application makes it easy to browse and manage your file system and client Web applications as well as open and view many types of files including archive and package files such as .zip and .rpm files. Another significant application set in KDE is KOffice, which offers the usual suite of office applications—word processor, spreadsheet, presentation, and drawing solutions. KOffice uses XML for the native document file format, which allows general use and ease of interpretation and transformation for KOffice documents.

The KDE User Guide can be found at http://docs.kde.org/en/HEAD/kdebase/userguide/.

Like GNOME, KDE provides a comprehensive development environment for writing KDE applications. Of course, it is also possible to run Java and GTK development tools, such as Eclipse and MonoDevelop, in KDE.

For a description of the origins and evolution of KDE, refer to the history at http://www.kde.org/history/.

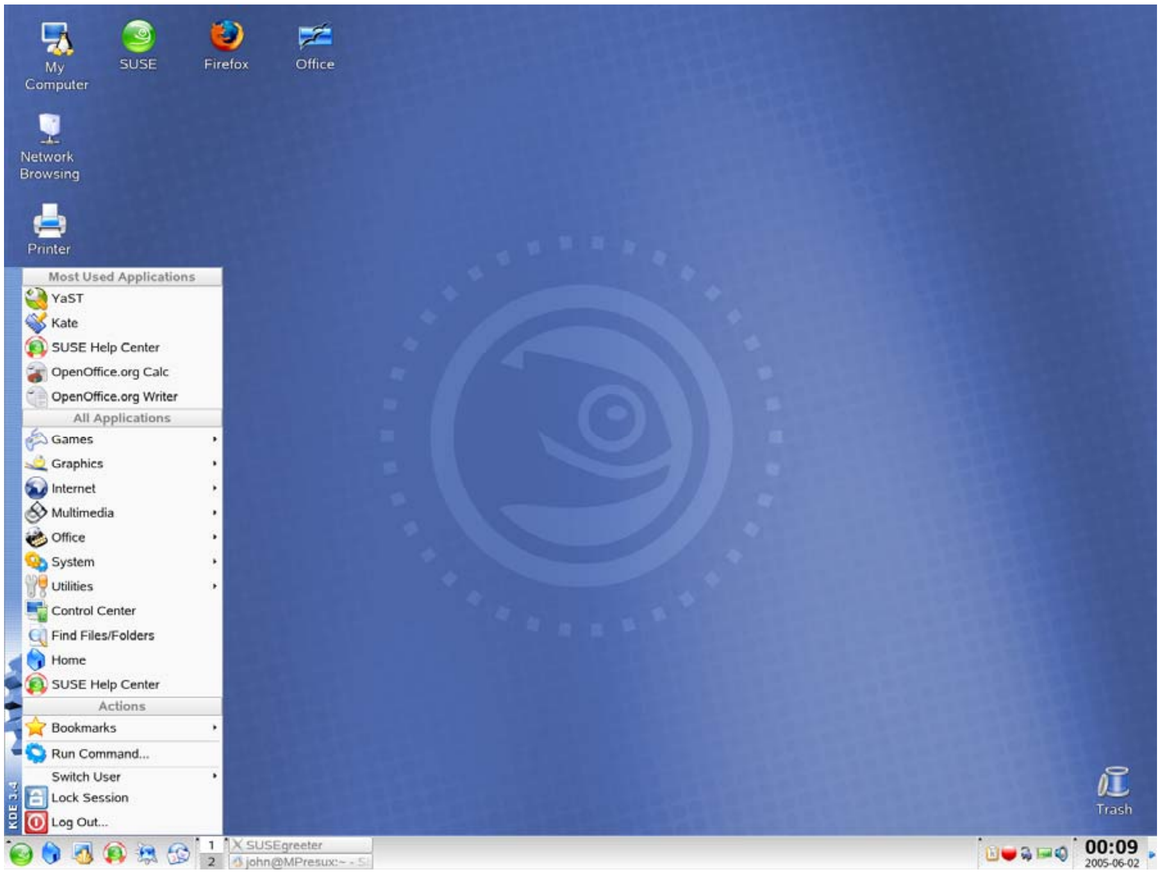


Figure 1.2: The KDE desktop environment.

Application Packages and Development Tools

A desktop such as those we just looked at provides a working environment for a developer. Application packages and tools are the raw materials a developer uses to build new applications. The two application frameworks that this guide will explore are J2EE and .NET via the Mono open source implementation of .NET. Although Java itself is not open source, we will look at development tools and application servers for Java that are open source. There are also interesting open source combinations of Java and .NET such as IKVM and Grasshopper. IKVM is a Java (VM and class libraries) implementation in .NET. Grasshopper cross compiles .NET MSIL to Java bytecode. The Grasshopper Visual Studio plug-in lets you code C# ASP.NET applications in Visual Studio and compile the result to run in a Java VM on Linux (in a server container such as Tomcat).

Application Servers

There are several application servers for J2EE applications such as BEA WebLogic, Sun Application Server, and IBM WebSphere Application Server. This guide is only concerned with the open source application server, JBoss. JBoss began as an open source Enterprise Java Bean (EJB) project and is now a very strong open source community with a number of projects in addition to the JBoss Application Server.

📖 Visit the JBossWiki at http://www.jboss.org/wiki/Wiki.jsp for more background information.

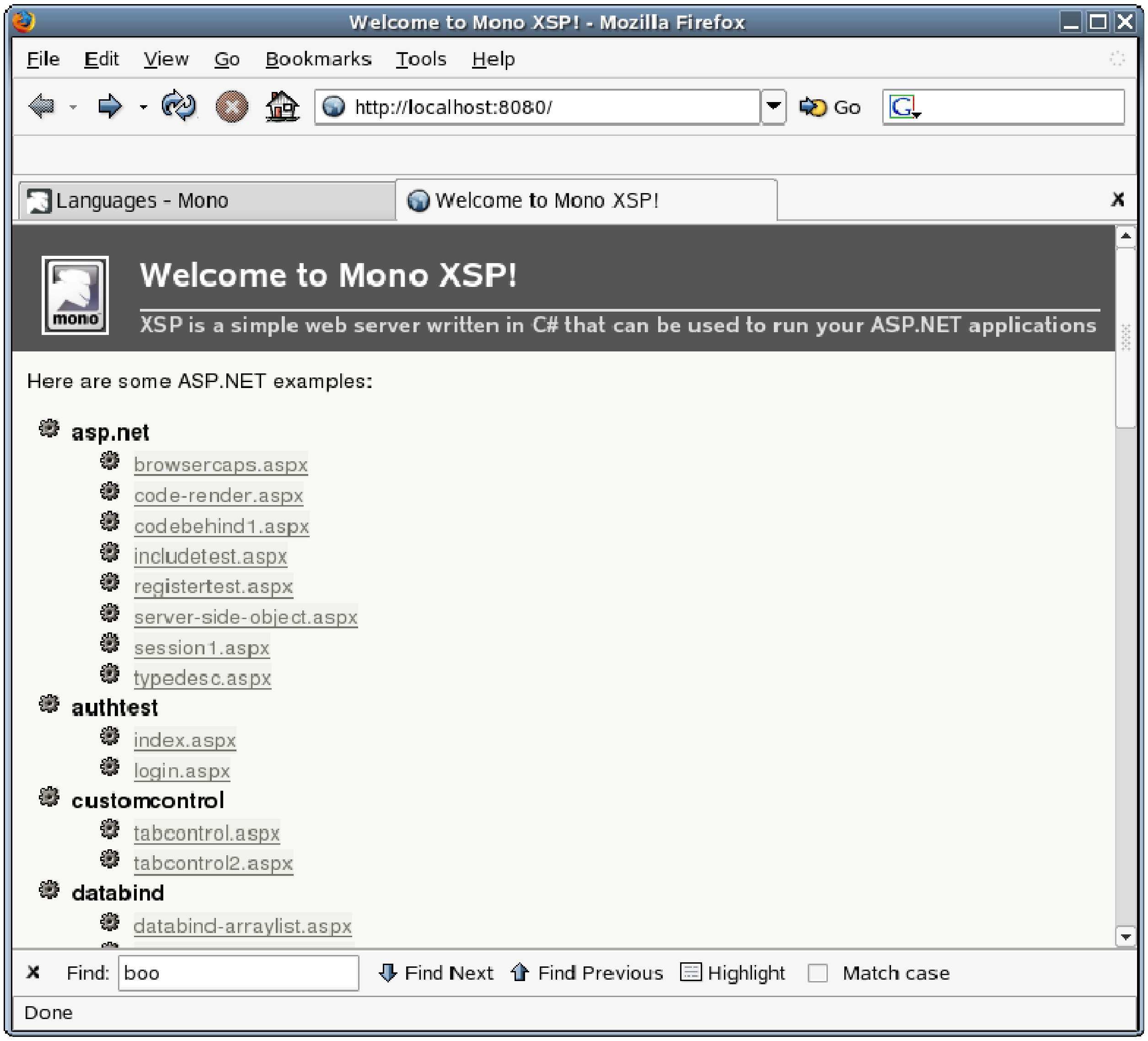
JBoss

The open source Java application server JBoss is in its fourth major release. JBoss has a clean modular architecture built on a microkernel core using Java Management Extensions (JMX). The microkernel core and services layer create a strong service-oriented architecture. An *aspect layer* over the services layer enables object access to the services. The aspect layer also gives a developer the capability to use attribute tags when programming applications. The current version of JBoss is 4.0.2 which is J2EE 1.4 certified. JBoss 4 requires a Java VM 1.4 or higher. The SUSE Linux Professional 9.3 distribution includes JBoss 3.2.5-6 in the Productivity/Networking/Web/Servers YaST package group. There is a YaST dependency on a Sun Java Development Kit (JDK) package.

📖 Chapter 2 will look in detail at installing and configuring JBoss; this chapter will explore how to develop a Java enterprise application to run in JBoss.

Mono XSP

The .NET application server we will look at is the Mono “container” XSP and the corresponding Apache module, mod_mono, to hook Mono into an Apache server. Mono XSP is a server container that provides a .NET environment using Mono to host ASP.NET applications and gain access on a TCP port. It is a basic implementation that will likely grow to include many and will become a full-fledged application server. Figure 1.3 is a sample page with links to sample ASP.NET demos.



Welcome to Mono XSP!

XSP is a simple web server written in C# that can be used to run your ASP.NET applications

Here are some ASP.NET examples:

- asp.net
  - browsercaps.aspx
  - code-render.aspx
  - codebehind1.aspx
  - includetest.aspx
  - registertest.aspx
  - server-side-object.aspx
  - session1.aspx
  - typedesc.aspx
- authtest
  - index.aspx
  - login.aspx
- customcontrol
  - tabcontrol.aspx
  - tabcontrol2.aspx
- databind
  - databind-arraylist.aspx

Figure 1.3: The Mono XSP interface.

IDEs

A considerable evolution from edit-compile-link (e-c-l) programming, IDEs combine e-c-l into a common environment with other tools such as UI designers and schema editors. As with application servers, there are several IDEs available—although they're not all open source—such as BEA Workbench and netBeans. We will focus on the open source plug-in platform Eclipse for Java and MonoDevelop for .NET. In addition to a strong and vibrant community behind Eclipse, the JBoss Eclipse plug-in project is a key reason for the focus on Eclipse in this guide.

Eclipse

Eclipse is an open platform for tool integration that is written in Java. The three main Eclipse projects are the Eclipse Platform, the Eclipse Java development tools (JDTs), and the Eclipse plug-in development environment (PDE). To install Eclipse from the SUSE Linux Professional distribution, pick the Eclipse packages provided in the following list from the YaST Java Selections group (YaST Package Groups Development/Tools/IDE):

- eclipse—base package
- eclipse-platform—common files
- eclipse-gtk2—GTK2 support
- eclipse-jdt—JDT
- eclipse-pde—PDE
- eclipse-scripts—startup scripts
- eclipse-source—source
- libswt3-gtk—GTK2 version of SWT library

YaST will pick up dependencies for packages such as ant and junit if you don't already have them installed. If you need a newer version of Eclipse (version 3.0.2 is the latest released version at the time of this writing) you can download an Eclipse SDK package from http://www.eclipse.org/downloads/index.php. The flavor to pick for SUSE Linux is "Linux (x86/GTK2)." Stable beta releases are also available, for example, version 3.1RC1 at the Eclipse Web site. The SDK downloads are not integrated into your system when you install them. They unpack into a private self-contained directory that provides the program and a full set of plug-ins for all the supporting tools. This setup makes it easy to acquire and test multiple versions of the Eclipse platform simultaneously. SUSE has not made many changes to the directories and files that YaST uses for system-wide installation of Eclipse. You might be tempted to move a private Eclipse installation directory to /usr/share/eclipse, but doing so is not recommended. If you do, you will compromise version and dependency tracking done by YaST and will likely miss files in /usr/bin and /usr/lib.

After installation, you will have Eclipse on the GNOME and KDE application menu. You can also start Eclipse from a command line simply by typing

```
eclipse &
```

at your command line prompt. Figure 1.4 shows the Eclipse welcome window.

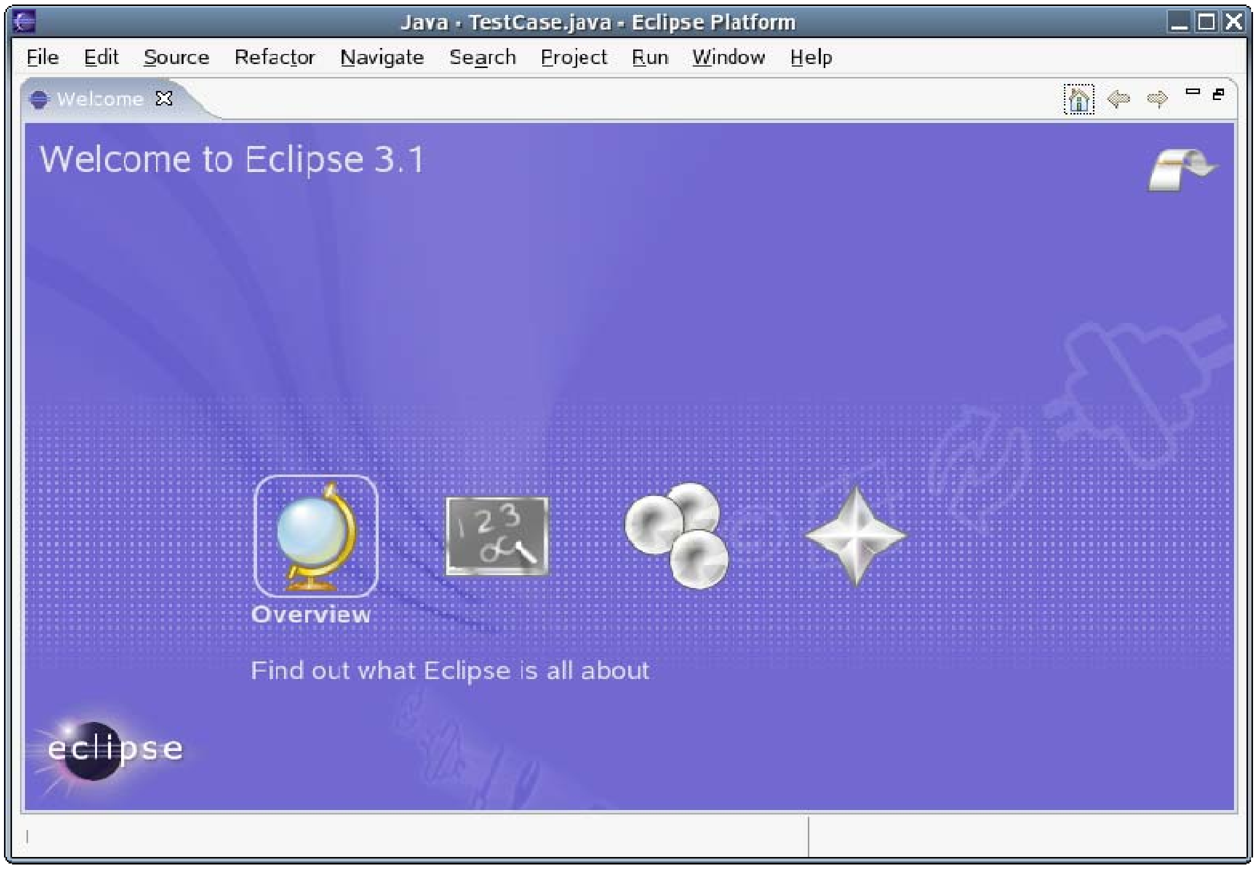
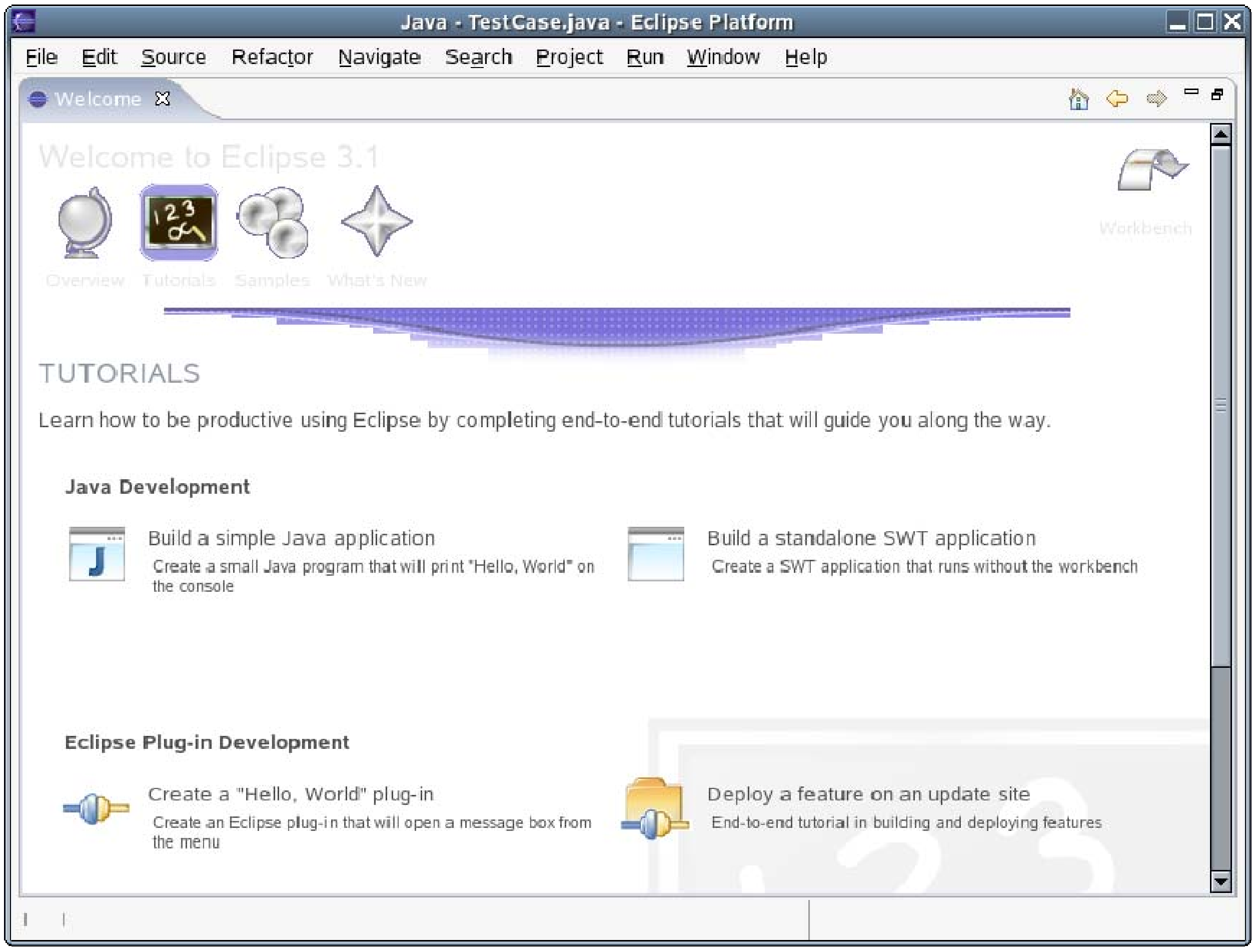

Java - TestCase.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse 3.1
Overview
Find out what Eclipse is all about
eclipse

Figure 1.4: The Eclipse welcome window.

The built-in tutorials are a good way to become familiar with Eclipse and the JDT. Simply select the tutorial icon (the chalkboard icon) on the welcome window. Figure 1.5 shows the tutorial selection window.


Java - TestCase.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse 3.1
Overview
Tutorials
Samples
What's New
Workbench

TUTORIALS

Learn how to be productive using Eclipse by completing end-to-end tutorials that will guide you along the way.

Java Development

Build a simple Java application
Create a small Java program that will print "Hello, World" on the console

Build a standalone SWT application
Create a SWT application that runs without the workbench

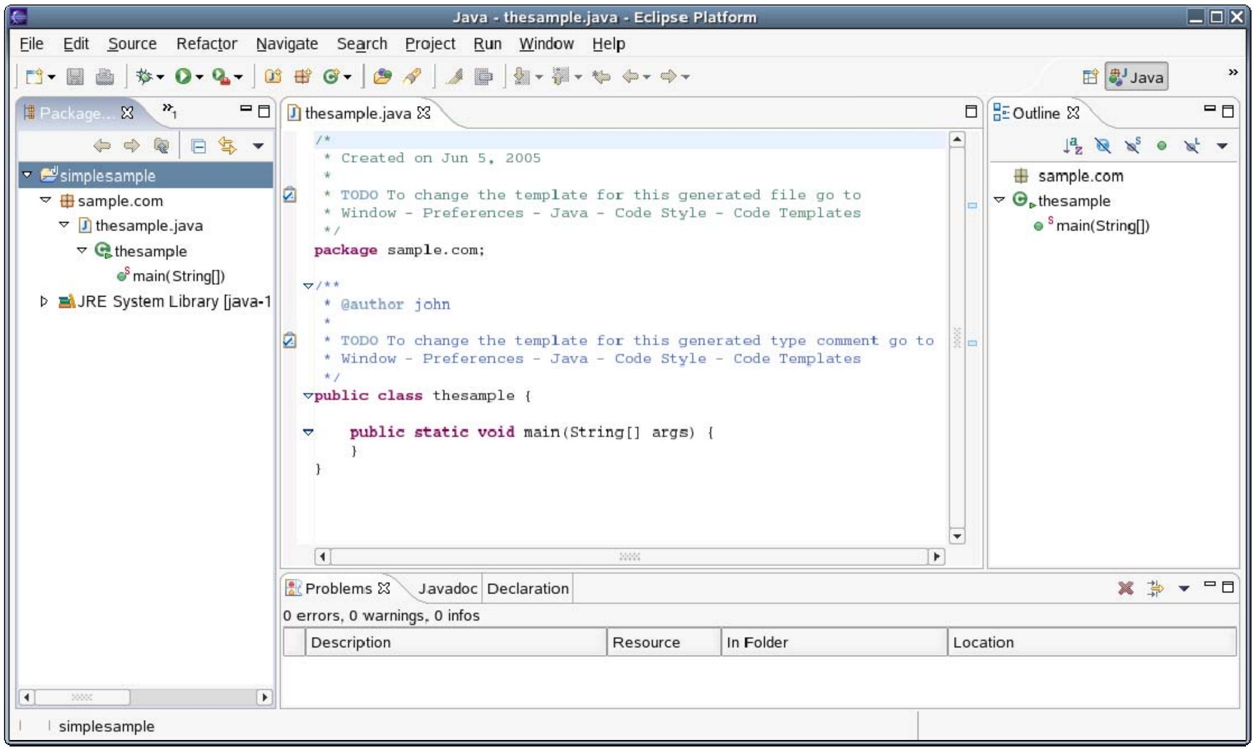
Eclipse Plug-in Development

Create a "Hello, World" plug-in
Create an Eclipse plug-in that will open a message box from the menu

Deploy a feature on an update site
End-to-end tutorial in building and deploying features

Figure 1.5: Eclipse tutorials.

It is also easy to learn Eclipse by creating a project and exploring the UI. The window layout, menus, and toolbars are quite intuitive. Figure 1.6 shows a Java project open in Eclipse.



```
/*
 * Created on Jun 5, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.com;

/**
 * @author john
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class thesample {

    public static void main(String[] args) {
    }
}
```

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location

Figure 1.6: A Java project in Eclipse.

Eclipse JDT also makes it easy to develop Java web applications with servlets and JSPs. A version of the container server Tomcat is integrated in Eclipse to test and debug web applications. There are a number of application frameworks for Java web applications with Eclipse integration or support. For example the Spring Framework, http://www.springframework.org project has a Spring plug-in for Eclipse. Check http://www.eclipse.org/org/index.html for background information on Eclipse.

MonoDevelop

MonoDevelop is a young project (current version is 0.7 at this writing). It is an Integrated Development Environment (IDE) for writing .NET applications in the Mono environment on Linux (Mono also runs on Windows). Although still somewhat bare bones, it is certainly possible to enter code into the editor and compile. MonoDevelop has several project solution templates for C#, VB.NET, Java, and Boo (a .NET CLI scripting language). Figure 1.7 shows the MonoDevelop solution template selection.

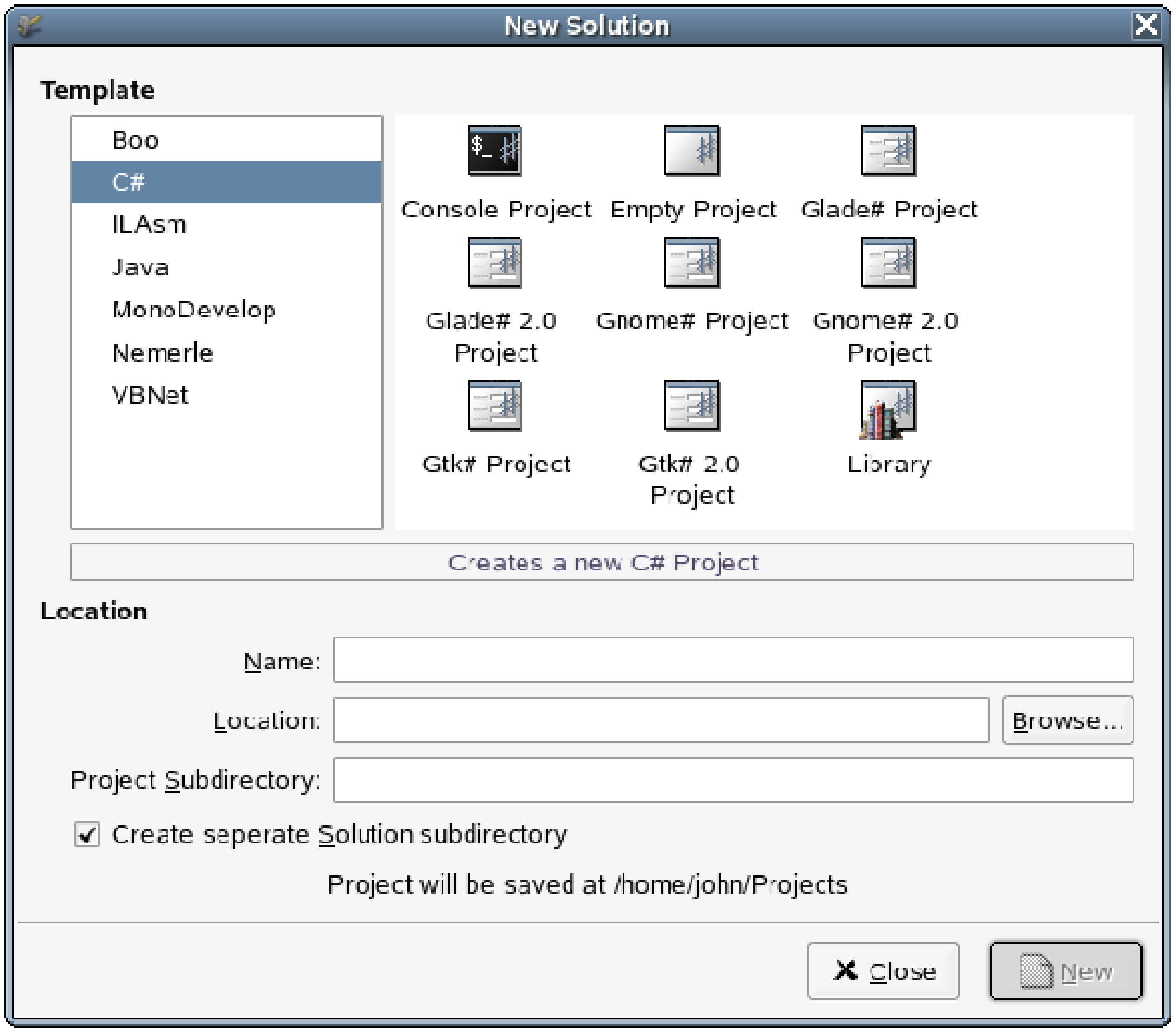


Figure 1.7: MonoDevelop solution templates.

The lack of designers for ASP.NET, Windows Forms, and GTK is a major shortcoming of MonoDevelop. However, if you develop .NET applications using Microsoft Visual Studio, it is certainly possible to bring the result to Mono and run it on Linux.

Libraries and Development Kits

There are many libraries in the Linux and Java open source universe. Apache Jakarta is a good example of a collection of libraries often in multiple forms such as Java class libraries and C/C++ API libraries. Mono provides an implementation of the ECMA .NET class libraries.

Java Libraries

The fundamental Java libraries of course are the class libraries provided by Sun in the Java Runtime Environment (JRE). Although not open source, the Sun JRE is widely implemented on many types of hardware and OSs. There is an open source project to develop an open version of the Java class libraries called *GNU Classpath*. It is currently in beta. The IKVM project is implementing a Java system that runs on .NET. IKVM is using GNU Classpath for the class libraries. Using Mono for the .NET runtime, IKVM provides a completely open implementation of Java.

Beyond the fundamental Java class libraries, the Apache Jakarta Commons project provides a significant set of Java application components. Table 1.1 provides the Jakarta Commons component libraries.

Component Library	**Definition**
Attributes	Provides a runtime API to metadata attributes such as doclet tags, inspired by the Nanning and XRAI projects as well as JSR 175 and C# attributes
BeanUtils	Provides easy-to-use wrappers around the Java reflection and introspection APIs
Betwixt	Provides services for mapping JavaBeans to XML documents and vice versa
Chain	Provides a chain of responsibility pattern implementation for organizing complex processing flows
CLI	Provides a simple API for working with command line arguments, options, option groups, mandatory options, and so forth
Codec	Contains general encoding/decoding algorithms; includes phonetic encoders, Hex, Base64, and a URL encoder
Collections	Provides a suite of classes that extend or augment the Java Collections Framework
Configuration	Offers tools to assist in the reading of configuration/preferences files in various formats
Daemon	Is an alternative invocation mechanism for UNIX-daemon-like java code
DBCP	Provides database connection pooling services
DbUtils	Is a Java Database Connectivity (JDBC) helper library that factors out mundane resource cleanup code for common database tasks
Digester	Is an XML-to-Java-object mapping utility commonly used for parsing XML configuration files
Discovery	Provides tools for locating resources (including classes) by mapping service/reference names to resource names using a variety of schemes
EL	Provides an interpreter for the Expression Language, which is defined by the JavaServer Pages specification version 2.0
Email	Provides a simple library for sending email from Java
FileUpload	Makes it easy to add robust, high-performance, file upload capability to servlets and Web applications

Component Library	Definition
HttpClient	Provides a framework for working with the client-side of the HTTP protocol
IO	Is a collection of input/output utilities
Jelly	Is an XML-based scripting and processing engine that borrows many good ideas from JSP custom tags, Velocity, Cocoon, and the scripting engine inside XDoclet; can be used from the command line, inside Ant, or inside a servlet
Jexl	Is an expression language that extends the Expression Language of the JSTL by bringing in some of the lessons learned by the Velocity community
JXPath	Provides utilities for manipulating Java classes that conform to the JavaBeans naming conventions using the XPath syntax; also supports maps, DOM, and other object models
Lang	Provides a very common set of utility classes that provide extra functionality for classes in java.lang
Latka	Is an HTTP functional testing suite for automated QA, acceptance, and regression testing
Launcher	Is designed to be a cross-platform Java application launcher and eliminates the need for a batch or shell script to launch a Java class; the original Java classes come from the Jakarta Tomcat 4.0 project
Logging	Is a wrapper around a variety of logging API implementations
Math	Is a library of lightweight, self-contained mathematics and statistics components that address the most common practical problems not immediately available in the Java programming language
Modeler	Provides mechanisms to create *Model MBeans* compatible with the Java Management Extensions (JMX) specification
Net	Is a collection of network utilities that is based on the NetComponents codebase, including FTP clients
Pool	Provides a generic object pooling interface, a toolkit for creating modular object pools, and several general purpose pool implementations
Primitives	Provides smaller, faster, and easier-to-work-with types supporting Java primitive types; currently, Primitives is focused on collections of primitives
Resources	Provides a lightweight framework for defining and looking up internationalized message strings keyed by a java.util.Locale and a message key
Transaction	Provides implementations for multi-level locks, transactional collections, transactional file access, and some other utility classes commonly used in transactional Java programming
Validator	Provides a simple, extendable framework to define validators (validation methods) and validation rules in an xml file; offers support for internationalization of validation rules and error messages
VFS	Is a Virtual File System component for treating files, FTP, SMB, ZIP, and such like as a single logical file system

Table 1.1: Jakarta Commons component libraries.

Another significant body of Java class libraries is available for XML processing. These are libraries such as Xalan (XSLT processor), Xerces (XML parser), and XMLBeans. In addition to Java, some have C++ versions. SUSE Linux Professional has an extensive collection of Java library packages. Browse the list in the YaST Package Group Development/Libraries/Java.

.NET Libraries

The base .NET class libraries are defined by the .NET Common Language Infrastructure (CLI) standard. The CLI standard is ECMA-335. It—together with the C# language specification ECMA-334—define .NET Framework that has been implemented by Microsoft and Mono. Application-oriented component libraries for .NET are not as prevalent as they are for Java. There is a substantial list, however, at http://www.gotdotnet.com. I also expect to see a growing number at http://www.gotmono.net.

It is important to understand the Mono release versions, the features they contain, and the roadmap for future releases. Mono version 1.0 was released in June of 2004. The following list highlights the features supported:

- A cross-platform ECMA CLI runtime engine
- A cross-platform Java runtime engine
- C# 1.0 compiler
- Development toolchain
- Class libraries implementing the .NET 1.1 profile
- The GTK# 1.0 GUI programming toolkit
- Mono extension libraries
- Third-party convenience libraries bundled with the release
- GNU Classpath for the CLI
- Visual Basic runtime.

The current Mono release version is 1.1.7-1 at the time of this writing. The 1.1 release stream is intermediate to the next major release which will be Mono 1.2. The Mono 1.2 release is expected in late 2005. The significant new features in Mono 1.2 will be:

- Generic types support: C# compiler, execution system and core class libraries (C# 2.0)
- System.Windows.Forms 1.1 support
- Mono Debugger
- GTK# 2.0 (includes support for GTK 2.6)
- Numerous scalability and performance enhancements

Notice that Windows.Forms is not in Mono 1.0 and is just now showing up in the 1.1 intermediate release stream. Mono 1.2 is also expected to have technology previews of some of the (not yet released by Microsoft) .NET 2.0 features that will be released with Visual Studio 2005. Full implementation of .NET 2.0 is planned for Mono 2.0, which is expected in the second half of 2006.

C/C++ Libraries

Even larger and definitely older than the Java community, the C/C++ community has the largest selection of class libraries (C++) and subroutine libraries (C). The de facto standard Linux C/C++ compiler is GNU Compiler Collection (GCC). In fact, GCC was a key enabler in the birth and growth of Linux. GCC and an extensive collection of libraries are contained in the SUSE Linux distribution. Libraries from Glib and GTK2 through Xerces-c are listed in the YaST Package Group Development/Libraries/C and C++.

Development Kits

The Sun JDK is the primary and most widely used development kit for Java. As we have seen, there are possible permutations and combinations of virtual machines (VMs), class libraries and compilers. In addition to open source Java development kits, some companies—such as IBM, BEA, and Microsoft—have produced proprietary Java implementations. The only two .NET development kits are the Microsoft .NET Framework SDK and the Mono project package. Both include a C# compiler as well as runtime and class libraries. The Sun JDK and Mono kit are available in the SUSE Linux Professional distribution. We will discuss installation and versions of the Sun JDK in more detail later in this chapter.

Programming Languages

The list of programming languages, assembled, compiled, and interpreted is huge. From FORTRAN and Algol to Java and C#, chances are there is some form available on Linux and more than likely multiple forms. The GNU compiler collection provides the bootstrap capability to build Linux kernel and drivers and support packages from source. This guide will use Java in the J2EE examples and C# in the .NET examples.

Building and Configuring a Development Workstation

SUSE Linux Professional includes 32-bit and 64-bit versions, so most Intel or AMD desktop or laptop is supported. Graphics and audio hardware are also widely supported, so chances are your video and sound cards are supported out of the box, as will network adapters. The only hardware that I have found that needed special driver setup is the MultiPort wireless LAN (WLAN) adapter for Compaq laptops.

Installation

You can install by using the boxed DVD/CD or via a network installation through FTP, HTTP, or file share. You might want to install SUSE Linux on a workstation that already has Windows installed, as doing so will preserve the Windows setup and add a boot selection so that either Windows or Linux can be booted from the same drive. When SUSE Linux runs, the Windows NTFS partition is mounted read-only so that you have easy access to any of your Windows files. Full details on the installation methods are in the SUSE Linux Professional documentation.

YaST

During installation, you must decide what to install and how to configure the workstation. Installation uses the same tool, YaST, that is used on a running system to select packages (see Figure 1.8).

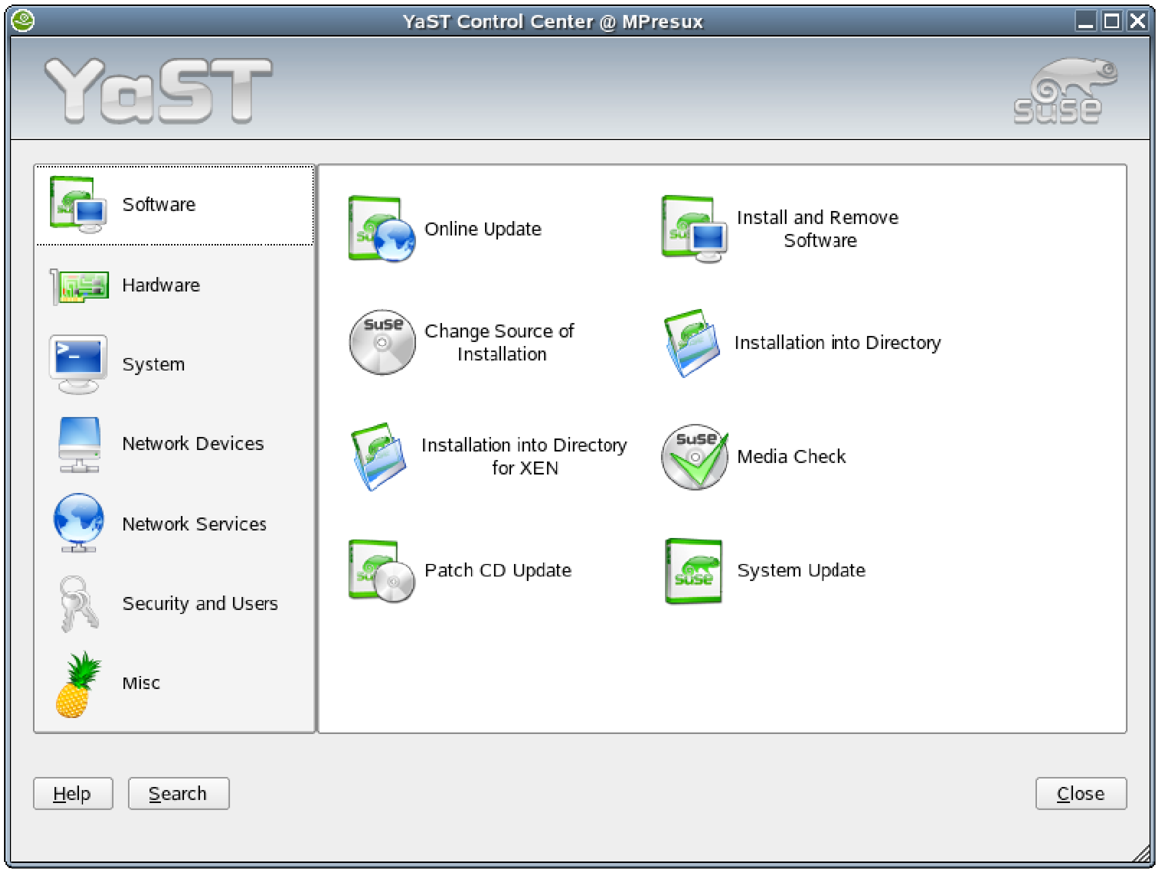


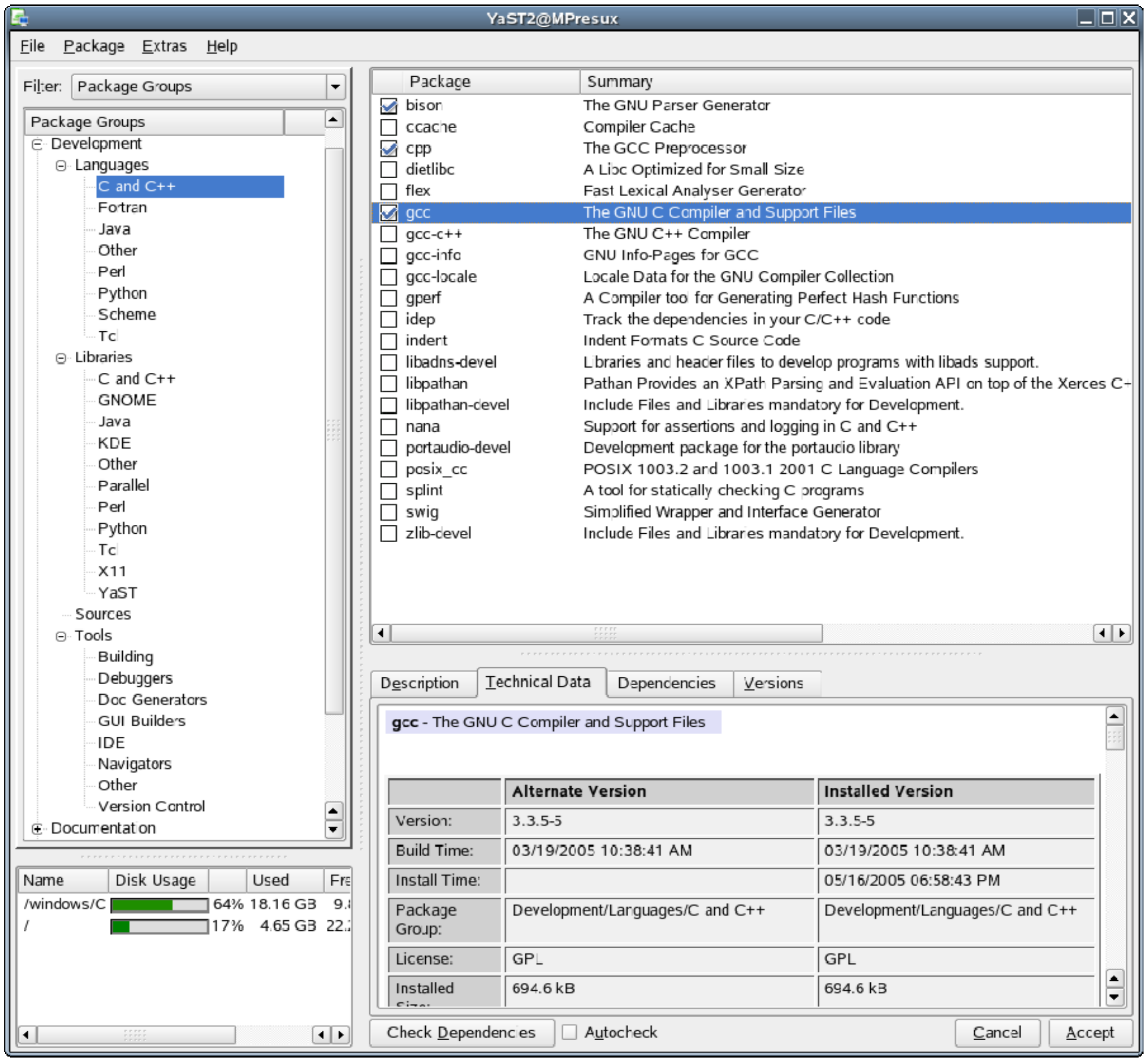
Figure 1.8: The main YaST screen.

YaST does a lot more than just package selection, but that is its main function during installation. The two important organization hierarchies are Package Groups and Selections. The Package Groups organize packages into groups of related components from which you make individual selections. Selections are larger groupings that allow you to select the entire group as a whole. Predefined system configurations are available at install time for minimal system, minimal graphics system, or default system with GNOME or KDE. I recommend you pick a default system during installation and add development tools and other packages after installation. See the *SUSE Linux Professional Administration Guide* for complete details about installation.

Package Groups

Packages for development tools, compilers libraries, and so on are the next thing to install after setting up a default system. SUSE Linux Professional ships with a wide selection of development related packages using the RPM Package Manager (RPM) system. There is a lot of version dependency and conflict detail to keep track of between packages, all of which is handled by YaST. SUSE has adapted all the included packages to fit into the SUSE system model in terms of file locations, environment variables, and PATH directories. Not all Linux/UNIX systems are the same when it comes to executables and library locations in /sbin or /usr/sbin or /bin or /opt (or a number of other places). You might need to install packages that are not available from SUSE in SUSE-specific layout, in which case you must be aware of which details may need adjusting in terms of file locations, links, and environment variables.

The place to find most of the development-related packages is, not surprisingly, in the Development branch of the YaST package groups list (see Figure 1.9).



Name	Disk Usage	Used	Fre
/windows/C	64%	18.16 GB	9.
/	17%	4.65 GB	22.

	Package	Summary
✓	bison	The GNU Parser Generator
	ccache	Compiler Cache
✓	cpp	The GCC Preprocessor
	dietlibc	A Libc Optimized for Small Size
	flex	Fast Lexical Analyser Generator
✓	gcc	The GNU C Compiler and Support Files
	gcc-c++	The GNU C++ Compiler
	gcc-info	GNU Info-Pages for GCC
	gcc-locale	Locale Data for the GNU Compiler Collection
	gperf	A Compiler tool for Generating Perfect Hash Functions
	idep	Track the dependencies in your C/C++ code
	indent	Indent Formats C Source Code
	libadns-devel	Libraries and header files to develop programs with libads support.
	libpathan	Pathan Provides an XPath Parsing and Evaluation API on top of the Xerces C-
	libpathan-devel	Include Files and Libraries mandatory for Development.
	nana	Support for assertions and logging in C and C++
	portaudio-devel	Development package for the portaudio library
	posix_cc	POSIX 1003.2 and 1003.1 2001 C Language Compilers
	splint	A tool for statically checking C programs
	swig	Simplified Wrapper and Interface Generator
	zlib-devel	Include Files and Libraries mandatory for Development.

gcc - The GNU C Compiler and Support Files

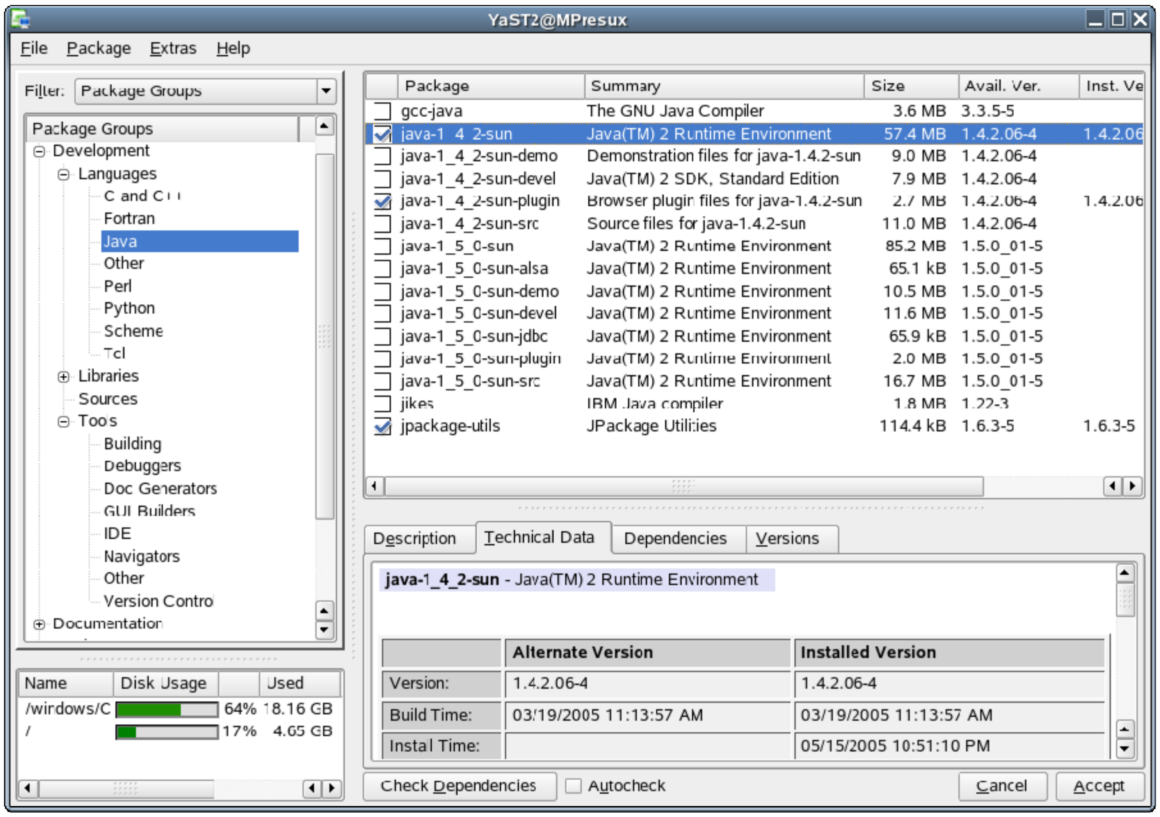
	Alternate Version	Installed Version
Version:	3.3.5-5	3.3.5-5
Build Time:	03/19/2005 10:38:41 AM	03/19/2005 10:38:41 AM
Install Time:		05/16/2005 06:58:43 PM
Package Group:	Development/Languages/C and C++	Development/Languages/C and C++
License:	GPL	GPL
Installed Size:	694.6 kB	694.6 kB

Figure 1.9: Selecting YaST Package Groups.

You will likely, at minimum, want to install GCC and basic tools such as make and bison. We will be using IDE packages such as Eclipse and MonoDevelop in Chapters 2 and 3 when we look at developing applications. You saw earlier in this chapter how they can be installed using YaST.

Let’s take a look at what is involved in installing a package that is not available in YaST. For example, suppose you needed to install the 5.0 version of the Sun JDK. As Figure 1.10 shows, YaST has the 1.4 JDK (listed as Java 2 SDK, Standard Edition in YaST) and the 1.5 JRE (Java Runtime Environment, 1.5 is the JRE version in J2SE 5.0) but the current JDK which is 1.5.0_03 is not in YaST.

There is possible confusion in that YaST has an entry for JDK 1.5.0_01 but erroneously lists the description as Java Runtime Environment (JRE). Although you could install JDK 1.5.0_01 via YaST, the following example shows what is involved in the system setup of a JDK not available in YaST.



	Package	Summary	Size	Avail. Ver.	Inst. Ve
	gcc-java	The GNU Java Compiler	3.6 MB	3.3.5-5	
✓	java-1_4_2-sun	Java(TM) 2 Runtime Environment	57.4 MB	1.4.2.06-4	1.4.2.06
	java-1_4_2-sun-demo	Demonstration files for java-1.4.2-sun	9.0 MB	1.4.2.06-4	
	java-1_4_2-sun-devel	Java(TM) 2 SDK, Standard Edition	7.9 MB	1.4.2.06-4	
✓	java-1_4_2-sun-plugin	Browser plugin files for java-1.4.2-sun	2.7 MB	1.4.2.06-4	1.4.2.06
	java-1_4_2-sun-src	Source files for java-1.4.2-sun	11.0 MB	1.4.2.06-4	
	java-1_5_0-sun	Java(TM) 2 Runtime Environment	85.2 MB	1.5.0_01-5	
	java-1_5_0-sun-alsa	Java(TM) 2 Runtime Environment	65.1 kB	1.5.0_01-5	
	java-1_5_0-sun-demo	Java(TM) 2 Runtime Environment	10.5 MB	1.5.0_01-5	
	java-1_5_0-sun-devel	Java(TM) 2 Runtime Environment	11.6 MB	1.5.0_01-5	
	java-1_5_0-sun-jdbc	Java(TM) 2 Runtime Environment	65.9 kB	1.5.0_01-5	
	java-1_5_0-sun-plugin	Java(TM) 2 Runtime Environment	2.0 MB	1.5.0_01-5	
	java-1_5_0-sun-src	Java(TM) 2 Runtime Environment	16.7 MB	1.5.0_01-5	
	jikes	IBM Java compiler	1.8 MB	1.22-3	
✓	jpackage-utils	JPackage Utilities	114.4 kB	1.6.3-5	1.6.3-5

Name	Disk Usage	Used
/windows/C	64%	18.16 GB
/	17%	4.65 GB

java-1_4_2-sun - Java(TM) 2 Runtime Environment

	Alternate Version	Installed Version
Version:	1.4.2.06-4	1.4.2.06-4
Build Time:	03/19/2005 11:13:57 AM	03/19/2005 11:13:57 AM
Install Time:		05/15/2005 10:51:10 PM

Figure 1.10: Java packages in YaST.

To access this version, you would start by downloading the 1.5.0_03 JDK from Sun at http://java.sun.com/j2se/1.5.0/download.jsp. There are two Linux packages available, an RPM package and a "user" private setup. The user setup is intended for people that want a setup local or private to their login rather than a system-wide setup. Download the RPM package that has a .bin extension, and execute it to create the jdk-1_5_0_03-linux-i586.rpm RPM package file. If you are not logged in as root (you normally shouldn't be), create a root shell using su, and run rpm –iv on the package. For a more detailed reference of using RPM from the command line, see the *SUSE Linux Professional Administration Guide*.

SUSE includes a viewer tool, KRPMView, that you can use to view the contents of an RPM file. Simply click on the package icon in a file manager window to access the viewer. It includes a button to install the RPM package that is open in the viewer using YaST.

Once the package is installed, notice that the base or home directory for the JDK is /usr/java/jdk-1_5_0_03. This is not at all integrated with where SUSE puts JDK and JRE installations. There are no changes to environment variables such as JAVA_HOME to use the new JDK. The Java-related environment variables are set using the shell profile script in /etc/profile.d/alljava.sh (or .csh for C shells). As you can see from Listing 1.1, the environment variables JAVA_HOME and JAVA_BINDIR are set using this script.

```
#
#    /etc/profile.d/alljava.sh
#
# send feedback to http://www.suse.de/feedback

#
# This script sets some environment variables for default java.
# Affected variables: PATH, JAVA_BINDIR, JAVA_HOME, JRE_HOME,
#                     JDK_HOME, SDK_HOME
#

if [ -x /usr/lib/jvm/java/bin/java ] || [ -x /usr/lib/jvm/java/bin/jre
] ; then
  export JAVA_BINDIR=/usr/lib/jvm/java/bin
  export JAVA_ROOT=/usr/lib/jvm/java
  export JAVA_HOME=/usr/lib/jvm/java
  if [ -x /usr/lib/jvm/java/jre/bin/java ] ; then
    export JRE_HOME=/usr/lib/jvm/java/jre
  else
    export JRE_HOME=/usr/lib/jvm/java
  fi
  unset JDK_HOME
  unset SDK_HOME
  if [ -x /usr/lib/jvm/java/bin/javac ] ; then
    # it is development kit
    if [ -x /usr/lib/jvm/java/bin/jre ] ; then
      export JDK_HOME=/usr/lib/jvm/java
    else
      export JDK_HOME=/usr/lib/jvm/java
      export SDK_HOME=/usr/lib/jvm/java
    fi
  fi
else
  if [ -x /usr/lib/jvm/jre/bin/java ] ; then
    # it is IBMJava2-JRE or SunJava2-JRE
    export PATH=$PATH:/usr/lib/jvm/jre/bin
    export JAVA_BINDIR=/usr/lib/jvm/jre/bin
    export JAVA_ROOT=/usr/lib/jvm/jre
    export JAVA_HOME=/usr/lib/jvm/jre
    export JRE_HOME=/usr/lib/jvm/jre
    unset JDK_HOME
    unset SDK_HOME
  fi
fi
```

Listing 1.1: The alljava.sh script.

Instead of changing the alljava.sh script, you can create a new script to clear and set environment variables appropriately. Listing 1.2 shows a potential script jdkjava.sh.

PATH is also adjusted to include the /bin directory of the JDK.

```
unset JDK_HOME
unset JAVA_BINDIR
unset JAVA_HOME
unset JRE_HOME
unset SDK_HOME
unset JAVA_ROOT

export JAVA_HOME=/usr/java/jdk1.5.0_03
export PATH=$PATH:$JAVA_HOME/bin
```

Listing 1.2:The jdkjava.sh script.

This example is just a skeleton that you will likely change and augment. You need to “source” the script, not just execute it, in order to change the environment variables of your current shell. That is, execute at the command line

```
prompt> source ./jdkjava.sh
```

In addition, be aware that SUSE creates a link in /usr/bin for the java executable. You should delete the link so that the executable in the JDK /bin directory is used.

Different packages will require you to modify different details so that they are adjusted to your system environment. The JDK example is representative of what needs to be considered and can be used as an example of what to look for when working with other packages. Depending on your needs and particular development project, your final setup will likely be a combination of packages from the SUSE distribution and direct downloads. You might even find situations in which you are working with the source for a package that you modify and re-build for use in your project—that is, after all, one of the benefits of open source application development.

Requirements for Debug and Test

Developing server-based enterprise applications, you will need a server environment to debug and test the application. Although you might want to do so using separate server hardware, it is also possible to run application servers for Java and Mono .NET on the development workstation. If you are building unit test capability into your application, install the junit package available in the SUSE distribution. Java container servers such as Tomcat are easy to install on your development workstation and can be used for debugging and testing. IDEs such as Eclipse and netBeans often bundle major supporting pieces such as Tomcat and ant as part of their installation packages.

Summary

The development of applications based on open source components has come a long way in the past 20 years—with major growth occurring in the past 5 years. In the past, developers had to invest a lot of time and effort gathering components and writing code for parts that were not available. The open source community has created many new fundamental components and enhanced and re-engineered the early components. For example, SUSE has built enhancements such as YaST and time-saving package collections. SUSE Linux Professional makes a great platform for application development. A comprehensive distribution with easy-to-use setup and configuration tools, this solution enables you to be up and coding much quicker than if you had to assemble the pieces on your own.

[**Editor's Note**: This content was excerpted from the free eBook *The Developer Shortcut Guide to SUSE Linux* (Realtimepublishers) written by John Featherly and available at http://cc.realtimepublishers.com/portal.aspx?pubid=339.]