Linux on System z Web 2.0

Setting up a Web 2.0 stack on SUSE Linux Enterprise Server 10 SP2 August 2008

Linux on System z Web 2.0

Setting up a Web 2.0 stack on SUSE Linux Enterprise Server 10 SP2 August 2008

Note
Before using this document, be sure to read the information in “Notices” on page 47.

First Edition – August 2008

This edition applies to SUSE Linux Enterprise Server 10 SP2 only.

Contents

Chapter 1. Introduction

In recent years, the traditional way to use the Internet has changed significantly. Web administrators who published information on the Internet invite their community to contribute to the company's Web pages. Applications like Wikis, Blogs, Content Management Systems and a couple of others are providing this new functionality to the Web administrators.

The variety of available open source Web 2.0 applications require a proper setup of server components. These server components and their setup for SUSE Linux Enterprise Server 10 SP2 on Linux® on System z™ are described in this whitepaper.

What is Web 2.0?

The key question which needs to be addressed is: What is Web 2.0?

Tim O'Reilly defines the Web 2.0 as [1]

> Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform.

This new perspective changed the usage of the Internet significantly. In the past, system administrators were required to prevent users to access their private application interfaces from Web applications to avoid security issues. Of course, security issues are still required to be prevented, but APIs have been defined to allow reuse by 3rd party Web applications.

Data that was earlier only available for customers who paid for it is now opened up for public use. Who imagined ten years ago that satellite images of the whole world would be freely available and accessible by a well defined, public API?

Additionally, Web applications are getting more dynamic and more flexible due to new technologies. AJAX is one example to increase the performance of a Web 2.0 application by receiving data asynchronously from different services. Functionality of desktop application like drag and drop are available through JavaScript™ libraries.

All of this created a new spirit around the Internet and this is called Web 2.0.

The Web 2.0 stack components

In general, the Web 2.0 stack is based on the software solution stack called LAMP (acronym for "Linux, Apache, MySQL, PHP (Perl and/or Python)"). Some enhancements to the LAMP stack are getting into the spotlight when the discussion moves to Web 2.0.

The following figure gives an overview about the Web 2.0 stack components

1. see http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html

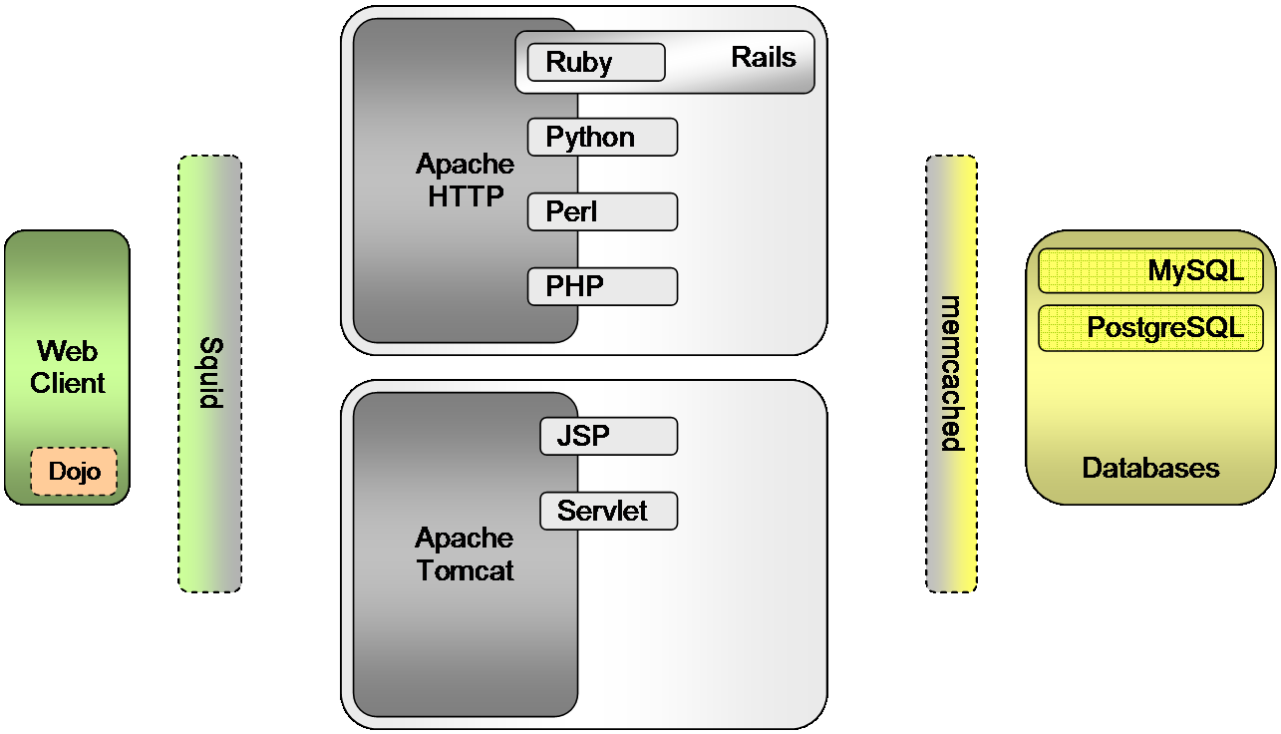

Figure 1. Web 2.0 stack components

Database
MySQL, PostgreSQL

Web server
Apache HTTP server, Apache Tomcat

Programming Languages
PHP, Perl, Python, Ruby, Java™

Frameworks
Dojo (AJAX support), Ruby on Rails

Caches
memcached, Squid

A system administrator does not have to install all Web 2.0 stack components at once. The selection is dependent on the Web 2.0 application and its functionality. i.e. if the Web 2.0 application offers information which is very static, it would make sense to establish a cache, but if the information is updated frequently then a cache is not the best choice.

The appendix gives an overview on all packages which are used in this document.

Applications exploiting the Web 2.0 stack

On the Internet, various open source Web 2.0 applications are available. In a follow-up whitepaper, the setup of some popular Web 2.0 applications running on Linux on System z will be described in detail.

The Web 2.0 stack which is described in this document enables a system administrator to choose the Web 2.0 application which fits best to the requirements. A list of very popular Web 2.0 applications is shown in the table below.

Table 1. Open Source Web 2.0 applications

Application Type	Application name
Wiki	MediaWiki
	MoinMoin
	Confluence
	XWiki
Blogs	Wordpress
	Mephisto
	Movable Type
Content Management Systems (CMS)	Typo3
	Drupal
	OpenCMS
	Apache Lenya
Business Applications	osCommerce

Web 2.0 on Linux on System z

Why run the Web 2.0 stack on Linux on System z?

High availability
A System z enterprise mainframe offers a high available environment by default. This reduces system outages which are related to hardware issues.

Resource utilization
Rarely used Linux on System z virtual server machines running on z/VM® are swapped out and the resources are made available for other virtual machines. The swap in of a Linux on System z virtual server is hardly noticed by the user.

Vertical Scalability
By default, Linux on System z virtual servers running on z/VM are enabled for vertical scalability. This means a Linux on System z virtual server can be empowered with additional memory or CPU power on the fly.

Rapid deployment
If the vertical scalability is not sufficient, a new Linux on System z virtual server can be deployed within a couple of minutes.

Performance
Connections between virtual servers on System z can make use of the HiperSockets™ technology to speed up communication.

Consolidation/TCO
The different Web 2.0 stack components might be running on dedicated server machines. Linux servers can be consolidated to run on one physical System z machine. This saves power, space in the computer center and reduces the administration effort.

System requirements for the Web 2.0 stack

The system requirements for a Web 2.0 stack setup are closely related to the requirements of the Web 2.0 application. An application which is based on a large database requires more system resources like CPU and memory than an application which is running the application logic as client-side JavaScript.

The system requirements are investigated in more detail in a follow-up whitepaper which explains the Web 2.0 applications in more detail.

Assumptions for this whitepaper

System administrators who read this document should be familiar, how to setup YaST to include the SUSE Linux Enterprise Server 10 SP2 DVD image and the related SDK DVD image as install sources.

Where to find this document

You can find the latest version of this document on the developerWorks® Web site at http://www.ibm.com/developerworks/linux/linux390/distribution_hints.html

Chapter 2. Setup of Databases

In general, Web applications deal with a lot of information which is displayed to the user. To get a structured and stable environment in place which guarantees consistency and persistency, databases are used. By default, SUSE Linux Enterprise Server 10 SP2 includes two popular databases: MySQL and PostgreSQL.

MySQL is one of the key components of the software solution stack called LAMP. Many Web 2.0 applications support both databases, so it is up to the system administrator which one to use.

Setup of MySQL

MySQL is a SQL database management system (DBMS), which has become one of the most popular open source database systems.

Some key characteristics - ease of use, reliability, security and performance at close to zero cost - increased the usage of MySQL, especially in Web applications.

Detailed information about MySQL is available at http://www.mysql.org

Installation of MySQL

The installation of the MySQL is straight forward as the packages are included in SUSE Linux Enterprise Server 10 SP2. Install the MySQL server and the MySQL client packages by using the following command:

```
# yast -i mysql mysql-client
```

The mysql-client package includes a command line client for MySQL. This client is required for administration purposes of the database server.

To access the MySQL database from a Web application, a client for the related programming language is required. For the programming languages mentioned in this document, different client implementations are available. The setup of these clients are described in the section where the setup of the programming languages is described.

Verifying the MySQL installation

To verify MySQL has been installed correctly, run the following command:

```
# mysql -V
mysql Ver 14.12 Distrib 5.0.26, for ibm-linux (s390x) using readline 5.1
```

The output should look similar to the above.

Lifecycle of MySQL

Lifecycle operations of Linux services like starting/stopping are basic functionality. In general, the lifecycle functionality is supported by a command line tool named service. The tool allows, dependent on the supported functionality of the service specific script, to start, stop and restart a server. Apart from that, the current status

of a server can be displayed. In the following, a walkthrough is shown for the MySQL server. This walkthrough assumes that this is the initial startup right after the packages have been installed.

1. Verifying the current status of the server

   To get the current status of the MySQL server, run the following command:

```
# service mysql status
Checking for service MySQL:                                    unused
```

   The status unused indicates that the MySQL server is not started.

2. Starting the server for the first time

   To start the MySQL server, run the following command

```
# service mysql start
Creating/Updating MySQL privilege database...
Installing all prepared tables
Fill help tables
(......)
Updating MySQL privilege database...
Fixing privilege tables...
Starting service MySQL                                         done
```

   The output displays some initialization steps which indicate that the MySQL server has been started for the first time. After the startup is completed, verify the status of the MySQL server by running:

```
# service mysql status
Checking for service MySQL:                                    running
```

   As shown, the MySQL server is running as expected.

3. Restarting the server

   Another useful functionality is the restart where it is possible to stop and start the server with one single command. To restart the MySQL server, run the following command:

```
# service mysql restart
Shutting down service MySQL                                    done
Starting service MySQL                                         done
```

   Once the restart completes, verify the status of the MySQL server by running:

```
# service mysql status
Checking for service MySQL:                                    running
```

   As expected, the MySQL server is running again.

4. Stopping the server

   To stop the MySQL server, run the following command:

```
# service mysql stop
Shutting down service MySQL                                    done
```

   Again, the status can be verified by running:

```
# service mysql status
Checking for service MySQL:                                    unused
```

As expected, the MySQL server is not running anymore.

To start the MySQL server at boot time, the command chkconfig is used. Decide first in which runlevel the MySQL server should start. Runlevels are used to coordinate the startup of services during boot time. In the following example, the runlevels 3 and 5 are used as both support networking:

```
# chkconfig --level 35 mysql on
```

To verify the setup use

```
# chkconfig --list mysql
mysql                   0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Now, the MySQL server starts during boot time of runlevels 3 and 5. To deactivate this behavior again, run:

```
# chkconfig mysql off
```

Again, to verify the setup use

```
# chkconfig --list mysql
mysql                   0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

Basic configuration of MySQL

MySQL delivers several example configuration files in its packages. At least four different flavours of the server configuration are available. These additional configuration files are located in the folder /usr/share/mysql and are named:

- my-small.cnf
- my-medium.cnf
- my-large.cnf
- my-huge.cnf

The MySQL documentation offers additional information on which configuration file to choose for a given scenario. For example, the my-small.cnf file can be used as the configuration for the MySQL server. The following steps should be processed to activate the predefined configuration:

```
# cp /usr/share/mysql/my-small.cnf /etc/my.cnf
# chown root:root /etc/my.cnf
# chmod 644 /etc/my.cnf
```

One of the first issues when dealing with any common resource is to ensure its security. At the beginning the password for the database superuser is set properly by running the following commands:

```
# mysqladmin -u root password <'new-password'>

# mysqladmin -u root -h <your_host_name> password <'new-password'>
```

or by using the MySQL console:

```
# mysql -u root
  mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('<new_password>');

  mysql> SET PASSWORD FOR 'root'@'<your_host_name>' = PASSWORD('<new_password>');
  mysql> quit
```

Note: As this is the superuser for all databases, use a strong password

The second recommended security measure is to remove anonymous accounts from the server. Access is granted only to those users specifically enabled.

```
# mysql -u root -p
  mysql> DELETE FROM mysql.user WHERE User = '';
  mysql> FLUSH PRIVILEGES;
  mysql> SELECT Host, User FROM mysql.user;
  mysql> quit
```

To get the list of users which have database access privileges, run the following command:

```
# mysql -u root -p
  mysql> select user,host from mysql.user;
+------+----------------------------------+
| user   | host                           |
+------+----------------------------------+

| root   | localhost                      |
| root   | machine.example.com            |
+------+----------------------------------+
  mysql> quit
```

Note: Applications connecting to the MySQL server's databases should use an account with minimum privileges required for its actions. This helps to prevent malicious code from accessing other databases and data.

The configuration shown in this chapter does not cover all security aspects of MySQL. For more detailed information, refer to the MySQL documentation.

Setup of PostgreSQL

PostgreSQL is an advanced object-relational database management system.

Numerous features such as stored procedures, functions and triggers are included. All of these are blocks of code to be executed by the server and can be written in SQL or some compatible languages like C, C++, Java, PHP, Perl, Python, Ruby, etc. PostgreSQL provides a MVCC (Multi-Version Concurrency Control), eliminating read locks efficiently, different rules to rewrite incoming queries, etc. These features are very advanced, and very similar to the ones offered by commercial alternatives.

Detailed information about PostgreSQL and the supported functionality is available at http://www.postgres.org

Installation of PostgreSQL

PostgreSQL has been included in various Linux distributions for some time. The installation process is straight forward. The database server can be installed by running the following command:

```
# yast -i postgresql postgresql-server
```

Next to the PostgreSQL server, the client which is required to maintain the databases is installed also.

To access the PostgreSQL database from a Web application, a client for the related programming language is required. For the programming languages mentioned in this document, different client implementations are available. The setup of these clients are described in the section where the setup of the programming languages is described.

Verifying the PostgreSQL installation

Checking whether the PostgreSQL package has been installed is done in the same way as with other servers. To retrieve the version of PostgreSQL the following command is performed:

```
# postgres -V
postgres (PostgreSQL) 8.1.11
```

The output should look similar to the above.

Lifecycle of PostgreSQL

Once these basic PostgreSQL packages are installed, lifecycle operations can be performed like starting/stopping the server. In the following, a walkthrough is shown for the PostgreSQL server. This walkthrough assumes that this is the initial startup right after the packages have been installed.

1. Verifying the current status of the server

   To get the current status of the PostgreSQL server, run the following command:

   ```
   # service postgresql status
   Checking for PostgreSQL:                                    unused
   ```

   The status unused indicates that the PostgreSQL server is not started.

2. Starting the server for the first time

   To start the PostgreSQL server, run the following command

   ```
   # service postgresql start
   Initializing the PostgreSQL database at location /var/lib/pgsql/data  done
   Starting PostgreSQL                                                   done
   ```

   The output displays an initialization step which indicates that the PostgreSQL server has been started for the first time. After the startup is completed, verify the status of the PostgreSQL server by running:

   ```
   # service postgresql status
   Checking for PostgreSQL:                                    running
   ```

As shown, the PostgreSQL server is running as expected.

3. Restarting the server

   Another useful functionality is the restart where it is possible to stop and start the server with one single command. To restart the PostgreSQL server, run the following command:

```
# service postgresql restart
Shutting down PostgreSQLpostmaster stopped
                                                                    done
Starting PostgreSQL                                                 done
```

   Once the restart completes, verify the status of the PostgreSQL server by running:

```
# service postgresql status
Checking for PostgreSQL:                                            running
```

   As expected, the PostgreSQL server is running again.

4. Stopping the server

   To stop the PostgreSQL server, run the following command:

```
# service postgresql stop
Shutting down PostgreSQLpostmaster stopped
                                                                    done
```

   Again, the status can be verified by running:

```
# service postgresql status
Checking for PostgreSQL:                                            unused
```

   As expected, the PostgreSQL server is not running anymore.

To start the PostgreSQL server at boot time, the command chkconfig is used. Decide first in which runlevel the PostgreSQL server should start. Runlevels are used to coordinate the startup of services during boot time. In the following example, the runlevels 3 and 5 are used as both support networking:

```
# chkconfig --level 35 postgresql on
```

To verify the setup use

```
# chkconfig --list postgresql
postgresql              0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Now, the PostgreSQL server starts during boot time of runlevels 3 and 5. To deactivate this behavior again, run:

```
# chkconfig postgresql off
```

Again, to verify the setup use

```
# chkconfig --list postgresql
postgresql              0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

Basic configuration of PostgreSQL

PostgreSQL does not allow remote connections by default. To enable remote connections a parameter needs to be set in the file /etc/sysconfig/postgresql. This file governs optional PostgreSQL startup parameters and needs to be created if it does not already exist on the system.

In the file find the reference to POSTGRES_OPTIONS. This should be changed to set the -i parameter at startup. The example below shows an extract of the file with POSTGRES_OPTIONS set to -i:

```
## Path:           Applications/PostgreSQL
## Description:    The PostgreSQL Database System
## Type:           string()
## Default:        ""
## ServiceRestart: postgresql
#
# The options that are given to the PostgreSQL master daemon on startup.
# See the manual pages for postmaster and postgres for valid options.
#
# Don't put "-D datadir" here since it is set by the startup script
# based on the variable POSTGRES_DATADIR above.
#
POSTGRES_OPTIONS="-i"
```

Run the following command to restart the PostgreSQL server:

```
# service postgresql restart
```

The client authentication is controlled by a configuration file, which traditionally is named pg_hba.conf. This file is stored in the database folder /var/lib/pgsql/data.

One possible scenario is to allow connections from local and from a specific subnet. The following example shows how this works for the subnet 192.168.12.0/24. Local users can connect only to their own databases (databases with the same name as their database user name), administrators can connect to all databases and client connections from a subnet of 192.168.12.x can connect to the databases db1 and db2. For all users, authentication is required.

```
# TYPE  DATABASE     USER         CIDR-ADDRESS       METHOD
local    sameuser     all                               md5
local    all          @admins                           md5
host     db1,db2      all           192.168.12.0/24     md5
```

The configuration shown in this chapter does not cover all security aspects of PostgreSQL. For more detailed information, refer to the PostgreSQL documentation.

Chapter 3. Setup of Apache HTTP Server

The Apache HTTP Server, also known as Apache, is a secure, reliable and efficient Web server. It is highly configurable and extensible with lots of third party modules and Apache is available for a wide variety of operating systems including Linux, UNIX®, Mac OS X, Microsoft® Windows® and several others.

Apache became the Web server component of the popular software solution stack called LAMP. Extending this concept, the Apache HTTP Server is a key component of the defined Web 2.0 stack also.

The Apache HTTP server is well documented. A good reference is the Apache Web page at http://httpd.apache.org

Installation of Apache HTTP Server

Apache is one of the key components of the World Wide Web, and is included in SUSE Linux Enterprise Server 10 SP2. The installation is performed using the following command:

```
# yast -i apache2 apache2-doc apache2-example-pages apache2-prefork
```

This installs the Apache HTTP server with some documentation and an example page.

Verification of the Apache HTTP Server installation

To verify the installation of Apache, run the following command:

```
# apache2ctl -v
Server version: Apache/2.2.3
Server built:   Apr 23 2008 22:56:54
```

The output should look similar to the above.

To connect with a Web browser to the Apache HTTP server, the server needs to be started. Use the following command to start the server:

```
# service apache2 start
```

Open a Web browser and enter the URL http://localhost. A default Web page gets displayed indicating that the Apache HTTP server is running properly.

Lifecycle of Apache HTTP Server

Once these basic Apache HTTP server packages are installed, lifecycle actions can be performed like starting/stopping the server. In the following, a walkthrough is shown for the Apache HTTP server. This walkthrough assumes that this is the initial startup right after the packages have been installed and the server is not running.

1. Verifying the current status of the server

To get the current status of the Apache HTTP server, run the following command:

```
# service apache2 status
Checking for httpd2:                                                    unused
```

The status unused indicates that the Apache HTTP server is not started.

2. Starting the server for the first time

   To start the Apache HTTP server, run the following command

```
# service apache2 start
Starting httpd2 (prefork)                                               done
```

   After the startup is completed, verify the status of the Apache HTTP server by running:

```
# service apache2 status
Checking for httpd2:                                                    running
```

   As shown, the Apache HTTP server is running as expected.

3. Restarting the server

   Another useful functionality is the restart where it is possible to stop and start the server with one single command. To restart the Apache HTTP server, run the following command:

```
# service apache2 restart
Shutting down httpd2 (waiting for all children to terminate)            done
Starting httpd2 (prefork)                                               done
```

   Once the restart completes, verify the status of the Apache HTTP server by running:

```
# service apache2 status
Checking for httpd2:                                                    running
```

   As expected, the Apache HTTP server is running again.

4. Stopping the server

   To stop the Apache HTTP server, run the following command:

```
# service apache2 stop
Shutting down httpd2 (waiting for all children to terminate)            done
```

   Again, the status can be verified by running:

```
# service apache2 status
Checking for httpd2:                                                    unused
```

   As expected, the Apache HTTP server is not running anymore.

To start the Apache HTTP server at boot time, the command chkconfig is used. Decide first in which runlevel the Apache HTTP server should start. Runlevels are

used to coordinate the startup of services during boot time. In the following example, the runlevels 3 and 5 are used as both support networking:

```
# chkconfig --level 35 apache2 on
```

To verify the setup use

```
# chkconfig --list apache2
apache2                 0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Now, the Apache HTTP server starts during boot time of runlevels 3 and 5. To deactivate this behavior again, run:

```
# chkconfig apache2 off
```

Again, to verify the setup use

```
# chkconfig --list apache2
apache2                 0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

Overview of Apache HTTP server modules

The Apache server modules are software elements which extend Apache's functionality. There are modules available to add server-side programming language support, authentication schemes and a lot of additional functionality.

By default, the Apache HTTP server has already some modules loaded, statically and dynamically. To display the list of configured modules, run the following command:

```
# apache2ctl -t -D DUMP_MODULES
```

During the installation of Apache HTTP server extensions additional modules are installed, but those might not be enabled. To get a list of all available modules, run:

```
# a2enmod -l
```

A module is enabled by running

```
# a2enmod <name_of_module>
```

A module is disabled by running

```
# a2dismod <name_of_module>
```

Adding support for PHP

One of the general purposes for the development of PHP was to create a scripting language which suites Web development requirements perfectly. For this reason, PHP is supported in most Web server environments running on various operating systems.

The extendible approach for PHP allows to install additional functionality which a Web application can make use of, like clients for databases.

Further information about PHP and additional libraries is available at http://php.net

Installation of mod_php

The Apache HTTP server can be enhanced by installing the package apache2-mod_php5. This is processed by using the following command:

```
# yast -i apache2-mod_php5
```

Apart from the apache2-mod_php5 package other dependent packages such as php5 are installed if they are not available on the system.

To enable the PHP support in the Apache HTTP server, the newly installed module needs to be enabled.

```
# a2enmod mod_php5
# service apache2 restart
```

During the installation process of PHP, one additional configuration file is created in the Apache server configuration folder. This file is located at /etc/apache2/conf.d/php5.conf and includes a definition for the module to configure the interpreter to do its work when a PHP page is requested.

Creation of a PHP 'Hello World!' Web example

To check whether PHP is working correctly within Apache, create a file named /srv/www/htdocs/test.php with the following content:

```
<html>
    <head>
        <title>Hello, World! - PHP</title>
    </head>
    <body>
        <?php
            print 'Hello, World!<br>';
            print 'The time on the server is ';
            print strftime('%H:%M:%S');
        ?>
    </body>
</html>
```

This newly created PHP script can be accessed by opening up a Web browser at http://localhost/test.php. A Web page is shown which displays 'Hello World!' along with the current time.

Database connectors for PHP

As an example for additional libraries, PHP provides database connectors to the MySQL and PostgreSQL databases. These can be installed using the following command:

```
# yast -i php5-mysql php5-pgsql
```

These two PHP extensions can be installed separately also.

Adding support for Perl

One of the most popular scripting languages is Perl. In general, Perl is based on other programming languages such as C, shell scripting (sh), AWK and Lisp. This enables users to make use of a very powerful text manipulation engine.

Another benefit is the amount of additional modules which are available for Perl. These modules add ease of use and interoperability with other tools or software components to the portfolio of the basic Perl functionality.

Further information about Perl is available at http://www.perl.org.

Installation of mod_perl

apache2-mod_perl is an Apache enhancement which integrates the Perl interpreter into the Apache HTTP server.

One benefit of the integration from Perl with Apache is that the startup time for any HTTP request for a Web page returned by a Perl script is reduced. The Perl scripts are compiled once and rerun every time a request occurs. To install the Perl module for Apache, run the following command:

```
# yast -i apache2-mod_perl
```

Apart from the apache2-mod_perl package other dependent packages such as the core Perl package are installed if they are not available on the system.

The installation of apache2-mod_perl integrates two new configuration files into the Apache HTTP setup: /etc/apache2/mod_perl-startup.pl and /etc/apache2/conf.d/mod_perl.conf

The file /etc/apache2/mod_perl-startup.pl includes a list of additional Perl modules which are pre-loaded by Apache. This mechanism speeds up the processing of Perl scripts since basic modules which are required for processing are already ready to use.

The second configuration file /etc/apache2/conf.d/mod_perl.conf includes the setup for the Perl interpreter. In the default configuration, a folder named /srv/www/cgi-bin is enabled to serve Perl scripts. This location can be accessed by a Web browser at two different URLs:

- http://localhost/perl/<script-name>
- http://localhost/cgi-perl/<script-name>

Dependent on the URL used, different handlers are used to run the Perl script. Any modification to the configuration files requires a restart of the Apache HTTP server to implement the new configuration.

To enable the Perl support in the Apache HTTP server, the new installed module needs to be enabled. Run:

```
# a2enmod mod_perl
# service apache2 restart
```

To get more information about the setup of apache2-mod_perl, refer to the documentation at http://perl.apache.org

Creation of a Perl 'Hello World!' example

Based on the previously explained configuration, setup a 'Hello World!' example. Therefore create a file named /srv/www/cgi-bin/hello.pl with the following content:

```
#!/usr/bin/perl
# Hello.pl
print "Content-type: text/html\n\n";
print "<html>";
print "<head><title>Hello World! - Perl</title></head>";
print "<body>Hello, World!<br>";
($sec,$min,$hr) = localtime();
print "The time on the server is $hr:$min:$sec";
print "</body></html>";
```

Once the file is created, open up a Web browser on the system where the Web server is running and use the following two URLs:

- http://localhost/perl/hello.pl
- http://localhost/cgi-perl/hello.pl

Database connectors for Perl

As an example for additional libraries, Perl provides database connectors to the MySQL and PostgreSQL databases. The connector for PostgreSQL is available on the SDK image only. Therefore add the SDK image to the installation sources. Afterwards run the following command to install the connectors:

```
# yast -i perl-DBI perl-DBD-mysql perl-DBD-Pg
```

For Perl, the generic database interface needs to be installed to be able to use the MySQL or PostgreSQL interfaces of Perl. The MySQL and PostgreSQL interfaces can be installed separately also.

Installing additional Perl modules using CPAN

Web applications might require additional Perl modules to be installed, which are not part of SUSE Linux Enterprise Server 10 SP2. For this case the CPAN interactive shell can be used to add the missing module to the system in a comfortable way. At the first time, CPAN runs through an initialization process. This process analyzes the system and offers default settings which work for most environments.

To install e.g. the Perl-CGI-Session library with CPAN, perform as follows:

```
# perl -MCPAN -e shell
cpan> install CGI::Session
...
cpan> quit
```

This installs the perl-CGI-Session module which is now ready to use.

Adding support for Python

Python is another popular scripting language. Some key characteristics are that the language is very easy to learn and the code is required to be well structured which eases up the maintenance.

Support for Python is available on most of the platforms, including Linux, Unix, Mac OS X, Windows, OS/2®, etc.

Further information about Python is available at http://www.python.org.

Installation of mod_python

The Apache HTTP server can easily be equipped to support Python. The required packages are all included in SUSE Linux Enterprise Server 10 SP2. To install the Python module for Apache, run the following command:

```
# yast -i apache2-mod_python
```

Apart from the apache-mod_python package other dependent packages such as the core Python package are installed if they are not available on the system.

Setting up Python handlers for Apache needs to be done manually. Python offers three different standard handlers to execute Python code within Apache: Publisher handler, PSP handler and CGI handler.

Using the Publisher handler

To enable the Publisher handler, a folder needs to be configured in the Apache HTTP server to which the publisher handler is assigned. Create a file named /etc/apache2/conf.d/mod_python-publisher.conf with the following content:

```
<Directory /srv/www/htdocs/python>
    SetHandler mod_python
    PythonHandler mod_python.publisher
    PythonDebug On
</Directory>
```

This configuration enables the folder /srv/www/htdocs/python to serve Python scripts by making use of the Publisher handler.

Using the PSP handler

To enable the Python Server Pages create a file named /etc/apache2/conf.d/mod_python-psp.conf with the following content:

```
<Directory /srv/www/htdocs/python-psp>
    AddHandler mod_python .psp
    PythonHandler mod_python.psp
    PythonDebug On
</Directory>
```

This configuration enables the folder /srv/www/htdocs/python-psp to deliver Python Server Pages using the PSP handler.

Using the CGI handler

Using the CGI handler is not recommend. It is only intended to migrate legacy code away from CGI.

To enable the Python support in Apache, the new installed module needs to be enabled and the server needs to be restarted. Run:

```
# a2enmod mod_python
# service apache2 restart
```

To get more information about mod_python, refer to the documentation at http://www.modpython.org

Creation of a Python 'Hello World!' example

For the two Python specific handlers separate 'Hello, World!' examples need to be created. The environment for the examples is based on the configuration described above.

For the Publisher handler example, create a file named /srv/www/htdocs/python/index.py with the following content:

```
from mod_python import apache
import time

def index(req):
  req.content_type = 'text/html'
  req.write('<html>')
  req.write('<head><title>Hello, World! - Python</title></head>')
  req.write('<body>Hello, World!<br>')
  req.write('The time on the server is ')
  req.write(str(time.strftime("%H:%M:%S")))
  req.write('</body></html>')
```

The PSP handler is slightly different since it allows embedding Python code into HTML code. A file named /srv/www/htdocs/python-psp/hello.psp is created with the following content:

```
<html>
    <head>
        <title>Hello, World! - Python PSP</title>
    </head>
    <body>
        <% import time %>
        Hello, World!<br>
        The time on the server is <%=time.strftime("%H:%M:%S")%>
    </body>
</html>
```

To give the two 'Hello, World!' examples a try, open a Web browser on the machine where the Web server is running and have a look at the result of the following two URLs:

The Publisher handler 'Hello, World!'

- http://localhost/python
- http://localhost/python/index
- http://localhost/python/index/index

The PSP handler 'Hello, World!'

http://localhost/python-psp/hello.psp

Database connectors for Python

As an example for additional libraries, Python provides database connectors to the MySQL and PostgreSQL databases. These can be installed using the following command:

```
# yast -i python-mysql PyGreSQL
```

These two Python extensions can be installed separately also.

Adding support for Ruby

Ruby is a object-oriented programming language. The language gained large popularity boost in 2004 when the Web application framework “Ruby on Rails” was released. Additionally, implementations for virtual machines also became very popular. Such implementations are for example JRuby for the Java virtual machine or IronRuby for the .NET framework.

Interpreters for Ruby are available for Windows, Mac OS X, Linux and several other operating systems.

To get more information about Ruby, refer to the project Web page at http://ruby-lang.org.

Installation of mod_ruby

mod_ruby enables the execution of Ruby scripts by the Apache HTTP server. It embeds the Ruby interpreter to run the scripts natively in the Apache HTTP server.

The mod_ruby package is not part of SUSE Linux Enterprise Server 10 SP2. To enable the Apache HTTP server with native Ruby support, a tarball needs to be downloaded from the Internet at http://www.modruby.net/en/index.rbx/mod_ruby/download.html. During time of writing the stable version of mod_ruby was 1.2.6.

Prior to compiling mod_ruby some dependencies need to be resolved. Install the Ruby interpreter and the Apache Portable Runtime libraries along with the related development packages. The Ruby interpreter is available on the SUSE Linux Enterprise Server 10 SP2 SDK image. This needs to be available as installation source. Use the following commands to install the dependent packages:

```
# yast -i apache2-devel libapr1 libapr1-devel libapr-util1 libapr-util1-devel
# yast -i ruby ruby-devel
```

To install mod_ruby perform the following steps:

```
# tar xzf mod_ruby-1.2.6.tar.gz
# cd mod_ruby-1.2.6/
# ./configure.rb --with-apr-includes=/usr/include/apr-1
# make
# make install
```

The 'make install' step integrates mod_ruby into the Apache HTTP server infrastructure. During this step the configuration for mod_ruby is not enabled and needs to be done manually.

A file which includes the configuration needs to be created in the Apache HTTP server environment. Create the file /etc/apache2/conf.d/mod_ruby.conf with the following content:

```
# LoadModule ruby_module /usr/lib/apache2/mod_ruby.so
<IfModule mod_ruby.c>
  RubyRequire apache/ruby-run

  <Location /ruby>
    SetHandler ruby-object
    RubyHandler Apache::RubyRun.instance
    Options +ExecCGI
  </Location>

  <Files *.rbx>
    SetHandler ruby-object
    RubyHandler Apache::RubyRun.instance
  </Files>
</IfModule>
```

To enable the configuration for mod_ruby, use the following commands:

```
# a2enmod mod_ruby
# service apache2 restart
```

Installation of eRuby

eRuby allows embedding of Ruby code into HTML files similar to other approaches like ASP, JSP or PSP. It is not part of SUSE Linux Enterprise Server 10 SP2, but it is available on the related SDK image. To install eRuby support for Apache HTTP server, add the SDK image to the Installation Sources for the Software Management tool and use the following command:

```
# yast -i ruby ruby-devel eruby
```

In the previous section, mod_ruby was setup to serve Ruby CGI scripts. To setup eRuby, the configuration file /etc/apache2/conf.d/mod_eruby.conf is created with the following content:

```
AddType application/x-httpd-eruby .rhtml
Action application/x-httpd-eruby "/cgi-bin/eruby"
```

This requires adding the eRuby executable to the /cgi-bin directory. A symbolic link is created by using

```
# ln -s $(which eruby) /srv/www/cgi-bin
```

Once this symbolic link is in place, the folder cgi-bin needs to be allowed to follow symbolic links. In the file /etc/apache2/default-server.conf append the '+FollowSymLinks' value to the Options attribute of the <Directory "/srv/www/cgi-bin"> entry.

This new functionality is activated once the Apache HTTP server is restarted. Use the following command to restart the server:

```
# service apache2 restart
```

Note: SELinux might reject the execution of eRuby because of the symbolic link. One possible solution is to disable SELinux for the Apache HTTP server. In

many environments, disabling SELinux is not acceptable; therefore refer to the documentation at http://www.nsa.gov/selinux/ on how to setup SELinux properly.

Creation of a Ruby CGI 'Hello World!' example

Based on the previously explained environment, setup a 'Hello World!' example for CGI Ruby. Create a folder named /srv/www/htdocs/ruby:

```
# mkdir /srv/www/htdocs/ruby
```

In this folder, create a file named /srv/www/htdocs/ruby/hello.rbx with the following content:

```
require "cgi"
cgi = CGI.new("html4")
cgi.out {
  cgi.html {
    cgi.head { cgi.title {"Hello, World! - Ruby"} } +
    cgi.body {
      "Hello, World! <br>" +
      Time.now.strftime("The time on the server is %H:%M:%S")
    }
  }
}
```

This file requires the executable flag to be set

```
# chmod +x /srv/www/htdocs//var/www/html/ruby/hello.rbx
```

The Web page is reached by a locally started Web browser at the URL http://localhost/ruby/hello.rbx

Creation of a Ruby Server Page 'Hello World!' example

Based on the previously explained environment, setup a 'Hello World!' example for eRuby. Create a file named /srv/www/htdocs/hello.rhtml with the following content:

```
<html>
  <head>
    <title>Hello, World! - eRuby</title>
  </head>
  <body>
    <%= "Hello, World!" %><br>
    <%= Time.now.strftime("The time on the server is %H:%M:%S") %>
  </body>
</html>
```

The Web page is reached by a locally started Web browser at the URL http://localhost/hello.rhtml

Database connectors for Ruby

As an example for additional libraries, Ruby provides database connectors to the MySQL and PostgreSQL databases. These can be installed using the following command:

```
# yast -i ruby-mysql rubygem-ruby-postgres
```

The MySQL and PostgreSQL interfaces can be installed separately also.

Installing additional Ruby libraries using RubyGems

RubyGems is a packaging system for Ruby, similar to the CPAN tool of Perl. RubyGems allows the installation of additional Ruby libraries from a dedicated server. This is a very useful mechanism to increase the functionality of the Ruby installation if a package is missing in the distribution.

RubyGems is available on the SUSE Linux Enterprise Server 10 SP2 SDK image and is installed by using the following command:

```
# yast -i rubygems
```

Once RubyGems is installed on the system it can be used to enhance the Ruby installation with additional libraries such as tzinfo:

```
# gem install tzinfo --remote
```

This installs the Ruby tzinfo package.

Setup of Ruby on Rails

Ruby on Rails is a ruby-based framework for developing Web applications. It emphasizes ″convention over configuration″, is following the MVC (Model-View-Controller) pattern, making Web development faster and more efficient and Web applications easier to maintain.

To get more information on Rails refer to the project Web page at http://rubyonrails.org.

Installation of Ruby on Rails

Ruby on Rails is available from the SUSE Linux Enterprise Server 10 SP2 SDK image and is installed using the following command:

```
# yast -i rubygem-rails rubygem-rake
```

Verifying the Ruby on Rails installation: Check the version number by running the following command:

```
# rails -v
Rails 2.0.2
```

The output should look similar to the above.

Creation of a Ruby on Rails example

One way to run a Rails application in a production environment is to start the default Ruby application server on port 80. This approach - while being sufficient for a small installation - has the drawback of the default HTTP port now being bound exclusively to one single Rails application. The following example session shows how to create and run a Rails application.

```
# service apache2 stop
# cd /srv/www/htdocs/
# rails www.rubytest.example.com
....
# cd www.rubytest.example.com
# RAILS_ENV=production ruby script/server -p 80
```

Note: Running 'RAILS_ENV=production ruby script/server -p 80' requires rubygems of version 0.9.4 to be installed. The SDK actually includes only version 0.9.2. Follow the instruction which is shown in the output of the command 'ruby script/server' to upgrade to the appropriate version of rubygems.

To verify the Rails example, start a Web browser on the machine where the Rails application server is running and direct to the URL http://localhost. A 'Welcome aboard' screen comes up with some information how to proceed further activities on the Web application.

Deployment of Ruby on Rails applications

By default, the Rails application server is bound exclusively to a single port. This indicates that, if a Rails application is bound to port 80, the Apache HTTP server is not able to make use of this port anymore. A number of deployment options have been established to address this issue, including: running the Web application as a FastCGI process, using JRuby to deploy to a Java application server or forwarding requests from Apache or another Web server to a Ruby application server like Mongrel.

The latter method is used in the configuration example given below: requests are forwarded by the Apache HTTP server to a single Mongrel backend server using mod_proxy.

For further information about Mongrel visit the project Web site at http://mongrel.rubyforge.org

The Mongrel Web server package is available on the SUSE Linux Enterprise Server 10 SP2 SDK image and is installed using the following command:

```
# yast -i rubygem-daemons rubygem-mongrel
```

Then, the configuration for the Apache HTTP needs to be modified. Add a new virtual host by creating a file named /etc/apache2/vhosts.d/rubytest.conf with the following content[2]:

```
ProxyRequests off
<VirtualHost *:80>
        ServerName www.rubytest.example.com

        ProxyPass / http://127.0.0.1:8000/
        ProxyPassReverse / http://127.0.0.1:8000
        ProxyPreserveHost on
</VirtualHost>
```

Additionally, the Apache HTTP server needs to be setup to map requests for the new server name to the new virtual host. Therefore, the NameVirtualHost attribute

2. This configuration is taken from http://mongrel.rubyforge.org/wiki/Apache where additional information is available about the integration of Mongrel into Apache

needs to be configured in the file /etc/apache2/listen.conf. Since this is dependent on the individual Network setup, refer to the documentation of Apache HTTP server.

Afterwards make sure mod_proxy and proxy_http are enabled:

```
# a2enmod mod_proxy
# a2enmod proxy_http
```

Start the rails application in production mode on port 8000 by executing the following commands:

```
# cd /srv/www/htdocs/www.rubytest.example.com
# mongrel_rails start -d -p 8000 -e production
```

Note: When multiple Ruby on Rails Web applications are intended to run simultaneously on the same system, each Web application must be assigned to a unique port number. Therefore the configuration of the virtual host given in the rubytest.conf file needs to be duplicated with another ServerName. Additionally the ports of the ProxyPass and ProxyPassReverse attributes have to match the port specified during startup of the Web application.

Finally, restart Apache:

```
# service apache2 restart
```

Make sure the application is running by opening http://localhost in a browser. The default "Welcome aboard" page should now appear.

Chapter 4. Setup of Apache Tomcat

Apache Tomcat is a Servlet container which is compliant with the official specifications for the Java Servlet and JavaServer Pages technologies, providing an environment for Java code to run in cooperation with a Web server. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process. The two Java Specification Requests, JSR53 and JSR154 specify the Servlet API and the Java ServerPages API.

Further information about Apache Tomcat is available at the project Web page http://tomcat.apache.org.

Installation of Apache Tomcat

Apache Tomcat is a Web server which supports the JSP and Servlet standards. Java needs to be installed as a prerequisite to the Apache Tomcat server.

In this document Java 1.5.0 from IBM® is used. This is included in SUSE Linux Enterprise Server 10 SP2. Install the Java packages by using the following command:

```
# yast -i java-1_5_0-ibm java-1_5_0-ibm-devel jpackage-utils
```

Verifying the Java installation

Confirm that Java has been installed successfully by running the following command:

```
# java -version
java version "1.5.0"
Java(TM) 2 Runtime Environment, Standard Edition (build pxz64dev-20080315 (SR7))
IBM J9 VM (build 2.3, J2RE 1.5.0 IBM J9 2.3 Linux s390x-64 j9vmxi6423-20080315 ...
J9VM - 20080314_17962_lHdSMr
JIT  - 20080130_0718ifx2_r8
GC   - 200802_08)
JCL  - 20080314
```

The output should look similar to the above.

Note: If the output reports a different version of Java, but the 1.5.0 version has been installed, it might be the case that the version 1.5.0 is not set to be the one used by the system. The tool update-alternatives is used to switch the Java version used by the system. Use 'update-alternatives --config java' to select the right Java version.

Once the correct Java version has been installed, install Apache Tomcat by running:

```
# yast -i tomcat5 tomcat5-admin-webapps tomcat5-webapps
```

This installs the Apache Tomcat server and some administrative Web applications which allow some basic configuration of the Apache Tomcat server. Additionally, several example Web applications for Apache Tomcat are installed.

Verifying the Apache Tomcat installation

To verify if the Apache Tomcat server is working properly, the server itself needs to be started. Therefore, run the following command:

```
# service tomcat5 start
```

Open a Web browser on the server where the Apache Tomcat server is running and open the URL http://localhost:8080. The default Apache Tomcat Web page is shown which offers some examples, documentation and administration pages.

Important folders in Apache Tomcat

There is an environment variable with special importance, $CATALINA_HOME. This environment variable is defined in /etc/tomcat5/tomcat5.conf and points to the root of the Apache Tomcat server installation – the default is /usr/share/tomcat5. These are some of the key Apache Tomcat folders, all relative to $CATALINA_HOME:

- $CATALINA_HOME/bin - Startup, shutdown, and other scripts.
- $CATALINA_HOME/conf - Configuration files and related DTDs. The most important file is the server.xml which is the main configuration file for Apache Tomcat.
- $CATALINA_HOME/logs - Default location of log files.
- $CATALINA_HOME/webapps - This is where additional Web applications are installed.

Important folders for storing library files are:

- $CATALINA_HOME/common/lib - Library files used by Apache Tomcat and Web applications.
- $CATALINA_HOME/shared/lib - Library files used across Web applications only.

Lifecycle of Apache Tomcat

Once these basic Apache Tomcat server packages are installed, lifecycle actions can be performed like starting/stopping the server. In the following, a walkthrough is shown for the Apache Tomcat server. This walkthrough assumes that this is the initial startup right after the packages have been installed and the server is not running.

1. Verifying the current status of the server

   To get the current status of the Apache Tomcat server, run the following command:

   ```
   # service tomcat5 status
   Checking for Tomcat (/srv/www/tomcat5/base/)                          unused
   ```

   The status unused indicates that the Apache Tomcat server is not started.

2. Starting the server

   To start the Apache Tomcat server, run the following command

   ```
   # service tomcat5 start
   Starting Tomcat (/srv/www/tomcat5/base/)                              done
   ```

After the startup is completed, verify the status of the Apache Tomcat server by running:

```
# service tomcat5 status
Checking for Tomcat (/srv/www/tomcat5/base/)                              running
```

As shown, the Apache Tomcat server is running as expected.

3. Restarting the server

   Another useful functionality is the restart where it is possible to stop and start the server with one single command. To restart the Apache Tomcat server, run the following command:

```
# service tomcat5 restart
Shutting down tomcat (/srv/www/tomcat5/base/)                             done
Starting Tomcat (/srv/www/tomcat5/base/)                                  done
```

   Once the restart completes, verify the status of the Apache Tomcat server by running:

```
# service tomcat5 status
Checking for Tomcat (/srv/www/tomcat5/base/)                              running
```

   As expected, the Apache Tomcat server is running again.

4. Stopping the server

   To stop the Apache Tomcat server, run the following command:

```
# service tomcat5 stop
Shutting down tomcat (/srv/www/tomcat5/base/)                             done
```

   Again, the status can be verified by running:

```
# service tomcat5 status
Checking for Tomcat (/srv/www/tomcat5/base/)                              unused
```

   As expected, the Apache Tomcat server is not running anymore.

To start the Apache Tomcat server at boot time, the command chkconfig is used. Decide first in which runlevel the Apache Tomcat server should start. Runlevels are used to coordinate the startup of services during boot time. In the following example, the runlevels 3 and 5 are used as both support networking:

```
# chkconfig --level 35 tomcat5 on
```

To verify the setup use

```
# chkconfig --list tomcat5
tomcat5                 0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Now, the Apache Tomcat server starts during boot time of runlevels 3 and 5. To deactivate this behavior again, run:

```
# chkconfig tomcat5 off
```

Again, to verify the setup use

```
# chkconfig --list tomcat5
tomcat5                 0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

The Apache Tomcat administration tools

Two main administration tools are included in the package tomcat5-admin-webapps called admin and manager.

The admin tool is used to configure the Apache Tomcat server itself. It provides functionality for user, group and role management. Additionally the debug level can be modified for an application or the whole server.

The manager tool provides functionality for the Web application lifecycle. Web applications can be deployed, started, stopped and Web application specific options can be modified.

By default, user access to these tools is not configured and has to be granted manually. The installation of Apache Tomcat includes a file called /etc/tomcat5/base/tomcat-users.xml which is used to assign roles to groups or users. For example, the user tomcat is assigned to the roles tomcat, admin and manager. Another user admin is assigned to the role admin and the user manager to the role manager. Such a setup is shown in the following:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="admin"/>
  <role rolename="manager"/>
  <user username="tomcat"  password="tomcat1" roles="tomcat,admin,manager"/>
  <user username="admin"   password="tomcat2" roles="admin"/>
  <user username="manager" password="tomcat3" roles="manager"/>
</tomcat-users>
```

Afterwards the two administration applications can be accessed by using the URL http://localhost:8080/admin and http://localhost:8080/manager/html.

Adding support for JSP and Servlet API

Installation of JSP and Servlet API libraries

The Apache Tomcat installation requires the JSP and Servlet libraries. Therefore an installation of additional packages is not required.

In general the JSP implementation classes and the JSP Standard Tag Library are included in the following packages:

- jakarta-taglibs-standard
- servletapi5

The content of these packages is included into the Apache Tomcat environment during their installation.

Creation of a JSP 'Hello World!' example

The verification of the JSP installation is a bit more complex than the other verification processes. First, create a folder where the JSP file is stored. This example uses the folder named /usr/share/tomcat5/webapps/sample-jsp:

```
# mkdir /usr/share/tomcat5/webapps/sample-jsp
```

Once this folder is created, create a file named /usr/share/tomcat5/webapps/sample-jsp/hello.jsp which contains:

```
<%@page import="java.util.Date" %>
<%@page import="java.text.DateFormat" %>
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
  <head>
    <title>Hello, World! - JSP</title>
  </head>
  <body>
    <% String hwStr = "Hello World!\n"; %>
    <% String timeStr = "The time on the server is "; %>
    <% DateFormat currentTime = DateFormat.getTimeInstance(DateFormat.FULL); %>
    <%= hwStr %><br>
    <%= timeStr + currentTime.format(new Date()) %>
  </body>
</html>
```

As well as the JSP file, create a folder which includes information about the Web application:

```
# mkdir /usr/share/tomcat5/webapps/sample-jsp/WEB-INF
```

This folder includes one additional file - web.xml - which can define various settings for the Web application. To keep this example simple it only includes the application name. The file /usr/share/tomcat5/webapps/sample-jsp/WEB-INF/web.xml needs to be created with the following content:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
     "http://java.sun.com/dtd/web-app_2_2.dtd">

<web-app>

    <display-name>JSP - Hello, World!</display-name>

</web-app>
```

To complete the setup the new Web application needs to be deployed into the Apache Tomcat server. This is done by using the manager application. In the page at http://localhost:8080/manager/html, the application stored in the path sample-jsp needs to be started. Once the application is running, it can be accessed at http://localhost:8080/sample-jsp/hello.jsp.

Note: Depending on the setup of Apache Tomcat the Web application example might be auto-deployed. In this case, the Web application is activated once the Apache Tomcat server is restarted.

Creation of a Servlet 'Hello World!' example

The verification of the Servlet follows the same procedure as the JSP example. First, create a folder where the Servlet and related filers are stored. This example uses the folder named /usr/share/tomcat5/webapps/sample-servlet.

```
# mkdir /usr/share/tomcat5/webapps/sample-servlet
# mkdir /usr/share/tomcat5/webapps/sample-servlet/WEB-INF
# mkdir /usr/share/tomcat5/webapps/sample-servlet/WEB-INF/classes
# mkdir /usr/share/tomcat5/webapps/sample-servlet/WEB-INF/classes/sample
```

The Servlet file is named /usr/share/tomcat5/webapps/sample-servlet/WEB-INF/classes/sample/HelloWorldServlet.java and includes the following content:

```
package sample;

import java.io.*;
import java.util.Date;
import java.text.*;

import javax.servlet.http.*;
import javax.servlet.*;

public class HelloWorldServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head><title>Hello, World! - Servlet</title></head>");
    out.println("<body>");
    out.println("Hello, World!<br>");

    DateFormat currentTime = DateFormat.getTimeInstance(DateFormat.FULL);
    out.println("The time on the server is " + currentTime.format(new Date()));
    out.println("</body></html>");

    out.close();
  }
}
```

This Java source file needs to be compiled by using the following commands:

```
# cd /usr/share/tomcat5/webapps/sample-servlet/WEB-INF/classes
# javac -cp /usr/share/java/servletapi5.jar sample/HelloWorldServlet.java
```

In the folder /usr/share/tomcat5/webapps/sample-servlet/WEB-INF, a web.xml file needs to be created with the following content:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
     "http://java.sun.com/dtd/web-app_2_2.dtd">

<web-app>
    <display-name>Servlet - Hello, World!</display-name>

    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>sample.HelloWorldServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>hello-servlet</url-pattern>
    </servlet-mapping>
</web-app>
```

To complete the setup, the new Web application needs to be deployed into the Apache Tomcat server. This is done by using the manager application. In the page at http://localhost:8080/manager/html, the application stored in the path sample-servlet needs to be started. Once the application is running, it can be accessed at http://localhost:8080/sample-servlet/hello-servlet.

Note: Dependent on the setup of Apache Tomcat, the Web application example might be auto-deployed. In this case, the Web application is activated once the Apache Tomcat server is restarted.

Adding Database connectors

Web applications which are deployed into the Apache Tomcat server may need to connect to databases. This section explains how to add database connectors for MySQL and PostgreSQL into the Apache Tomcat environment.

Setup of MySQL connector

The MySQL JDBC driver which is called 'MySQL Connector/J' is not included in SUSE Linux Enterprise Server 10 SP2. An archive is available at the MySQL development download page http://dev.mysql.com/downloads/connector/j/5.1.html .

To allow system-wide access to the MySQL Connector/J, the JAR file included in the archive needs to be integrated into the Java system environment. This file needs to be copied into the /usr/share/java folder which is a well known folder for Java libraries, created during the installation of Java and JPackage. The following example shows how to setup the connector for system-wide usage:

```
# tar zxf mysql-connector-java-5.1.6.tar.gz
# cd mysql-connector-java-5.1.6
# cp mysql-connector-java-5.1.6-bin.jar /usr/share/java/
# cd /usr/share/java
# chmod 644 mysql-connector-java-5.1.6-bin.jar
# ln -s mysql-connector-java-5.1.6-bin.jar mysql-connector-java.jar
```

Note: Version 5.1.5 is used in the examples throughout this document

To integrate MySQL Connector/J integrated into the Apache Tomcat environment, create a link from the JAR file into the folder $CATALINA_HOME/common/lib of the Apache Tomcat server. Create the link as follows:

```
# ln -s /usr/share/java/mysql-connector-java.jar /usr/share/tomcat5/common/lib/
```

The JDBC driver is available for usage after a restart of the Apache Tomcat server has been performed.

```
# service tomcat5 restart
```

Setup of PostgreSQL connector

The PostgreSQL JDBC driver is included in SUSE Linux Enterprise Server 10 SP2 and can be installed using the following command:

```
# yast -i postgresql-jdbc
```

The installed package includes different versions of the JDBC driver for PostgreSQL. The different versions belong to the different version of the Java installation. This document is based on Java 1.5.0 which requires making use of the JDBC3 driver for PostgreSQL. To add this specific JDBC driver to the Apache Tomcat configuration, a link from the JDBC driver into the folder $CATALINA_HOME/common/lib of the Apache Tomcat server needs to be created. Create the link as follows:

```
# ln -s /usr/share/pgsql/postgresql-8.1-404.jdbc3.jar /usr/share/tomcat5/common/lib/
```

The JDBC driver is available for usage after restarting the Apache Tomcat server.

```
# service tomcat5 restart
```

Chapter 5. Setup of Caches

Setup of Squid

Squid is a caching proxy for the Web supporting various protocols like HTTP, HTTPS, FTP and more. It is mainly used for two different purposes, Web caching reducing bandwidth and response times on the client side, and speeding up the delivery of Web elements by caching frequently-repeated requests on the server side. Nevertheless, it fits perfectly with roles such as proxying Secure Sockets Layer (SSL) requests and caching of Domain Name Server (DNS) lookups, and perform transparent caching. Squid also supports a wide variety of caching protocols, such as Internet Cache Protocol, (ICP) the Hyper Text Caching Protocol, (HTCP) the Cache Array Routing Protocol (CARP), and the Web Cache Coordination Protocol (WCCP).

Squid has extensive granular access control mechanisms and allows monitoring of critical parameters via the Simple Network Management Protocol (SNMP).

To get more information on Squid refer to the documentation at http://www.squid-cache.org

Installation of Squid

The Squid server is also part of SUSE Linux Enterprise Server 10 SP2. Use the following command to install the package:

```
# yast -i squid
```

Note: The Squid server requires the network setup to provide a fully qualified domain name (FQDN). Review /etc/hosts to make sure that a FQDN is provided.

Verification of the Squid installation

To verify the installation of Squid, use the following command:

```
# squid -v
Squid Cache: Version 2.5.STABLE12
(...)
```

The output in this document shows the version number. From the output of the command, several configuration parameters and values are displayed also.

Lifecycle of Squid

Once these basic Squid server packages are installed, lifecycle actions can be performed like starting/stopping the server. In the following, a walkthrough is shown for the Squid server. This walkthrough assumes that this is the initial startup right after the packages have been installed.

1. Verifying the current status of the server

   To get the current status of the Squid server, run the following command:

   ```
   # service squid status
   Checking for WWW-proxy squid                                    unused
   ```

The status unused indicates that the Squid server is not started.

2. Starting the server for the first time

   To start the Squid server, run the following command

```
# service squid start
Starting WWW-proxy squid  (/var/cache/squid)                           done
```

   After the startup is completed, verify the status of the Squid server by running:

```
# service squid status
Checking for WWW-proxy squid                                           running
```

   As shown, the Squid server is running as expected.

3. Restarting the server

   Another useful functionality is the restart where it is possible to stop and start the server with one single command. To restart the Squid server, run the following command:

```
# service squid restart
Shutting down WWW-proxy squid                                          done
Starting WWW-proxy squid  (/var/cache/squid)                           done
```

   Once the restart completes, verify the status of the Squid server by running:

```
# service squid status
Checking for WWW-proxy squid                                           unused
```

   As expected, the Squid server is running again.

4. Stopping the server

   To stop the Squid server, run the following command:

```
# service squid stop
Shutting down WWW-proxy squid                                          done
```

   Again, the status can be verified by running:

```
# service squid status
Checking for WWW-proxy squid                                           unused
```

   As expected, the Squid server is not running anymore.

To start the Squid server at boot time, the command chkconfig is used. Decide first in which runlevel the Squid server should start. Runlevels are used to coordinate the startup of services during boot time. In the following example, the runlevels 3 and 5 are used as both support networking:

```
# chkconfig --level 35 squid on
```

To verify the setup use

```
# chkconfig --list squid
squid           0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

Now, the Squid server starts during boot time of runlevels 3 and 5. To deactivate this behavior again, run:

```
# chkconfig squid off
```

Again, to verify the setup use

```
# chkconfig --list squid
squid           0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

Basic configuration of Squid

In SUSE Linux Enterprise Server 10 SP2, a version of Squid 2.5 is included. The documentation for this version is delivered within the Squid RPM package. It includes several examples of possible configurations and is stored in /usr/share/doc/packages/squid.

To cover all the configuration scenarios is beyond the scope of this whitepaper. As an example for a Squid configuration, the setup for a reverse proxy is given. Refer to the documentation for additional information.

Configuring the Squid server as a reverse proxy

Reverse proxy cache, also known as Web server acceleration, is a method of reducing the load on a busy Web server by using a Web cache between the server and the Internet. Another benefit is improved security. Additionally it is one of many ways to improve the scalability without increasing the maintenance of the server too much. A good use case of a reverse proxy is to reduce the workload on a Web server that provides both static and dynamic content. The static content can be cached on the reverse proxy while the Web server is freed up to better handle the dynamic content.

In the scenario where the Web server is running on a different machine, the configuration of the Squid server in /etc/squid/squid.conf looks like the following:

```
http_port 80                            # Port of Squid proxy
httpd_accel_host <your_webservers_ip>   # IP address of Web server
httpd_accel_port 80                     # Port of Web server
httpd_accel_single_host on              # Forward uncached requests to single host
httpd_accel_with_proxy on
httpd_accel_uses_host_header off
```

If the Web server runs on the same machine as the Squid server is running, the Web server must be re-configured to run on a different port than 80, e.g. 81. The reason is that clients connect to the Squid server which acts between the clients and the Web server. Therefore, the configuration in /etc/squid/squid.conf needs to be modified to redirect requests to port 81 of the local machine:

```
http_port 80                            # Port of Squid proxy
httpd_accel_host localhost              # IP address of Web server
httpd_accel_port 81                     # Port of Web server
httpd_accel_single_host on              # Forward uncached requests to single host
httpd_accel_with_proxy on
httpd_accel_uses_host_header off
```

Setup of memcached

Memcached is a distributed memory system for caching purposes. In general it is used to speed up communication between a Web application and a database. The result is to reduce response time for highly frequented Web pages and lesser load of the database server.

There are several client APIs available to access the memcached server. All of the programming languages described in this book, Java, PHP, Perl, Python and Ruby, are supported with a memcached client API.

There are various public Web sites which make use of memcached such as SourceForge, Wikipedia, YouTube, Facebook and many others.

More information is available on the memcached project Web page at http://www.danga.com/memcached.

Installation of memcached

Memcached is not part of SUSE Linux Enterprise Server 10 SP2 itself. At the moment, packages for memcached are only available from the Internet. The official Web site for memcached is http://www.danga.com/memcached/download.bml. At time of writing, the latest version is 1.2.5.

Prior to compiling the sources, other dependent packages such as gcc must be installed:

```
# yast -i gcc
```

Process the following steps to compile and install memcached:

```
# tar xzf memcached-1.2.5.tar.gz
# cd memcached-1.2.5
# ./configure --prefix=""
# make
# make install
```

For the configure command, the parameter prefix is set to an empty string to avoid the installation moving files to a location other than that required by memcached.

Verification of the memcached installation

To verify the build and installation of memcached, run the following command:

```
# make test
```

This runs several tests for the memcached and report success or any kind of failure.

Lifecycle of memcached

The memcached service can be started up with several command line options.

Memcached daemons can be started on as many spare machines as required. The memcached daemon has no configuration file, just a few command line options. For example to start memcached as a daemon process using 2GB of memory and

listening on IP 10.0.0.40 on port 11211. The user name needs to be submitted only for the case when running as root. Start memcached by running the following command:

```
# memcached -d -m 2048 -l 10.0.0.40 -p 11211 -u root
```

To get an overview for all command line options, refer to the memcached documentation.

Note: Memcached lacks authentication and security features, meaning it should only be used on servers with an appropriate firewall set up. By default, memcached uses the port 11211.

The memcached daemon process can be stopped by running the following command:

```
# killproc memcached
```

Chapter 6. Setup of AJAX support

Web 2.0 applications are getting more flexible and more dynamic due to the enablement of functionalities like "Asynchronous JavaScript and XML" (AJAX) or "Drag and Drop". Several frameworks have been created to offer a set of these functionality in a bundled package. The Dojo toolkit is one of the most popular frameworks which offers a lot of support to the developer of Web 2.0 applications.

Installation of the Dojo Toolkit

The installation of the Dojo toolkit is easy to handle. Right now there are no RPM packages available, but a tarball is available from the Dojo project Web site at http://dojotoolkit.org.

Extract the tarball and move the extracted files into a folder where the Web server has access to. The example below shows how to extract the tarball and move the content into the folder /srv/www/htdocs.

```
# tar xzf dojo-release-1.1.1.tar.gz
# mv dojo-release-1.1.1 /srv/www/htdocs
```

Example for using Dojo

This section shows how to create an example which displays the current time, updated every second.

The Apache HTTP server needs to be setup to support PHP and Dojo as previously described. PHP scripts need to have the executable flag set.

To give an example, two files need to be created. The first, time.html, should be placed in the /srv/www/htdocs folder. It should include the following source code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Dojo example: AJAX clock</title>

 <script type="text/javascript" src="dojo-release-1.1.1/dojo/dojo.js"
          djConfig="parseOnLoad: true"></script>

 <style type="text/css">
     @import "dojo-release-1.1.1/dijit/themes/tundra/tundra.css";
     @import "dojo-release-1.1.1/dojo/resources/dojo.css";
 </style>

 <style type="text/css">
  #error, #main {
   margin: auto;
   margin-top: 120px;
   text-align: center
  }
  #error {
   font-size: 120%;
  }
  #main input,button {
   font-size: 400%; width: 250px
  }


 </style>

 <script type="text/javascript">
  dojo.require("dojo.parser");
  dojo.require("dojox.timing");
  dojo.require("dijit.form.TextBox");
  dojo.require("dijit.form.Button");

  var timer;

  var startClock = function() {
   timer.start();
  }
  var stopClock = function() {
   timer.stop();
  }

  var getCurrentTime = function() {
   console.debug("Timer ticked: Requesting time from server")

   // Performs the AJAX request to the URL specified;
   // After the request is sent and the response is received,
   // the load event is triggered, which in turn sets the new
   // value of the clock widget.
   dojo.xhrGet( {
    url: "http://localhost/time.php",
    handleAs: "text",
    timeout: 4000,

    load: function(response) {
     dojo.byId("error").innerHTML = "";
     dijit.byId("clock").setValue(response);
    },
    error: function(response) {
     dojo.byId("error").innerHTML = response;
    }
   } );

  }
```

```
  dojo.addOnLoad(function() {
   // Create a new timer that fires a tick every second
   timer = new dojox.timing.Timer(1000);
   // Every tick will lead to updating the text box with the current
       //time value from the server
   timer.onTick = getCurrentTime;

   dojo.connect(dojo.byId("clock-start"), 'onclick', startClock);
   dojo.connect(dojo.byId("clock-stop"), 'onclick', stopClock);

  });
 </script>
</head>

<body class="tundra">
 <div id="error"></div>

 <div id="main">
  <input id="clock" dojoType="dijit.form.TextBox" style="" value="" /><br/>
  <button id="clock-start" dojoType="dijit.form.Button">Start the clock</button><br/>
  <button id="clock-stop" dojoType="dijit.form.Button">Stop the clock</button>
 </div>

</body>
```

The second file, time.php, is stored in the /srv/www/htdocs folder also. It should include the following source code:

```
<?php
 // Returns the server time as HH:MM:SS
 echo (date ("H").":".date("i").":".date("s"));
?>
```

To run the example, a Web browser needs to be opened on the machine where the Web server is running and pointed to the URL http://localhost/time.html

Appendix. Packages for the Web 2.0 stack

The following table includes an overview of all packages used in this document

Table 2. Package overview of Web 2.0 stack in SUSE Linux Enterprise Server 10 SP2

Component	Package Name	Version	Source
MySQL	mysql	5.0.26-12	SLES10 SP2
	mysql-client	5.0.26-12	SLES10 SP2
PostgreSQL	postgresql	8.1.11-0.2	SLES10 SP2
	postgresql-server	8.1.11-0.2	SLES10 SP2
Apache HTTP server	apache2	2.2.3-16.18	SLES10 SP2
	apache2-devel	2.2.3-16.18	SLES10 SP2
	apache2-doc	2.2.3-16.18	SLES10 SP2
	apache2-example-pages	2.2.3-16.18	SLES10 SP2
	apache2-prefork	2.2.3-16.18	SLES10 SP2
	apache2-mod_php5	5.2.5-9.5	SLES10 SP2
	apache2-mod_perl	2.0.2-14.2	SLES10 SP2
	apache2-mod_python	3.1.3-60.9	SLES10 SP2
	mod_ruby	1.2.6	Internet Download at http://www.modruby.net/en/index.rbx/mod_ruby/download.html
	eruby	1.0.5-13.3	SLES10 SDK
	libpar1	1.2.2-13.2	SLES10 SP2
	libapr1–devel	1.2.2-13.2	SLES10 SP2
	libapr-util1	1.2.2-13.2	SLES10 SP2
	libapr-util1-devel	1.2.2-13.2	SLES10 SP2
Apache Tomcat	tomcat5	5.0.30-27.26	SLES10 SP2
	tomcat5-webapps	5.0.30-27.26	SLES10 SP2
	tomcat5-admin-webapps	5.0.30-27.26	SLES10 SP2
PHP	php5-mysql	5.2.5-9.5	SLES10 SP2
	php5-pgsql	5.2.5-9.5	SLES10 SP2
Perl	perl-DBI	1.50-13.2	SLES10 SP2
	perl-DBD-mysql	3.0002-15.2	SLES10 SP2

Table 2. Package overview of Web 2.0 stack in SUSE Linux Enterprise Server 10 SP2 (continued)

Component	Package Name	Version	Source
	perl-DBD-Pg	1.43-13.4	SLES10 SDK
Python	python-mysql	1.2.0-17.2	SLES10 SP2
	PyGreSQL	3.7-14.2	SLES10 SP2
Java	java-1_5_0-ibm	1.5.0_sr7-0.2	SLES10 SP2
	java-1_5_0-ibm-devel	1.5.0_sr7-0.2	SLES10 SP2
	jpackage-utils	1.6.3-18.4	SLES10 SP2
	mysql-connector-java	5.1.6	Internet Download at http://dev.mysql.com/downloads/connector/j/5.1.html
	postgresql-jdbc	8.1.11-0.2	SLES10 SP2
Ruby	ruby	1.8.4-17.16	SLES10 SDK
	ruby-devel	1.8.4-17.16	SLES10 SDK
	rubygems	0.9.2-4.3	SLES10 SDK includes 0.9.2 - Version upgrade from Internet
	rubygem-rails	2.0.2-0.3	SLES10 SDK
	rubygem-rake	0.8.1-0.3	SLES10SDK
	rubygem-daemons	0.4.4-1.4	SLES10SDK
	rubygem-mongrel	0.3.13-2.4	SLES10 SDK
	ruby-mysql	2.7.1-1.4	SLES10 SDK
	rubygem-ruby-postgres	0.7.1–2006.04.06-2.4	SLES10 SDK
Squid	squid	2.5STABLE12-18.9	SLES10 SP2
memcached	memcached	1.2.5-1	Internet Download at http://www.danga.com/memcached/download.bml
Dojo	dojo-release	1.1.1	Internet Download at http://dojotoolkit.org/downloads

Notices

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

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

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

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

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

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

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

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

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

developerWorks, HiperSockets, IBM, OS/2, System z, z/VM

The following terms are trademarks of other companies:

Java, JavaScript, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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