The SME Server Developer’s Guide

Mitel Corporation

The SME Server Developer's Guide

by Mitel Corporation

Copyright © 2002-2006 Mitel Corporation

Last updated: $Date: 2006/05/29 09:02:22 $

Revision: $Id: devguide.sgml,v 1.50 2006/05/29 09:02:22 gordonr Exp $

Table of Contents

List of Tables

List of Figures

I. An overview of the SME Server

Chapter 1. About this manual

Mitel has released this documentation to encourage development on the SME Server platform. This documentation, the code examples herein, and the SME Server itself, are released under free licenses. These licenses permit copying and modification under the terms of those licenses, and are reprinted in the front of this manual.

Note: for Mitel developers

Mitel also maintains a separate, commercial release of the platform, called Mitel Standard Linux which is the basis of the Mitel 6000 Managed Application Server. The Mitel Standard Linux release has additional features, such as Blades and interaction with the Mitel Applications Management Center, which are not documented in this guide.

Please contact the Platforms Development Team for the Mitel Standard Linux developers guide, which should be read in conjunction with this guide. Any issues with Mitel Standard Linux should be raised in Mitel's internal problem tracking system.

Chapter 2. Who should read this manual?

This manual is aimed at developers and provides the information they require to integrate their applications into the SME Server platform. The manual discusses the key concepts of the SME Server such as the configuration database, configuration file templates and the events and actions model which differentiate the SME Server from other Linux distributions.

This manual is not a system administration or system tweaking guide for a particular release. Instead it provides examples of SME Server development best practice. This manual is also useful for SME Server system administrators to explain how the SME Server works "under the covers".

Chapter 3. What is the SME Server?

The SME Server is a software package that can be installed on a standard PC in less than thirty minutes, converting it into a complete, easy-to-use network server and firewall. The SME Server is based on the CentOS Linux server distribution, packaged in such a way that no knowledge of Linux is required to install or operate it. The CentOS packages are used unmodified, and configured automatically to emulate "best practice" from expert system administrators.

The SME Server runs on commodity PC hardware, and supports a range of configurations and devices such as:

- RAID disk mirroring
- Wide variety of network cards
- Tape backup
- Parallel port, USB or network printers
- A variety of Internet connectivity options, including cablemodem, DSL/PPPoE, static IP and dialup

Software for the SME Server is packaged using RPM Package Manager (RPM) system. Existing packages from CentOS and other third-party developers are used, wherever posssible. The SME Server uses the "best of breed" packages from the open source community. The design of the system allows for easy replacement of the packages if better choices become available. The current packages in use are:

Table 3-1. SME Server software

Feature	Software
Web server	Apache
Mail server	qmail
DNS server	djbdns and dnscache
FTP server	ProFTPd
Windows file sharing	Samba
Macintosh file sharing	Netatalk
Remote administration	SSH, PPTP, HTTP over SSL
Tape backups	Flexbackup
Webmail	Horde IMP

Chapter 4. Design philosophy

Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.

—Antoine de Saint-Exupéry

The SME Server automates the best practices of a skilled systems administrator, providing a simple interface for the users and consistent, modular extensibility for the developers.

Principle 1: Automating best practice

A good systems administrator knows what tasks must be done, either regularly or occasionally, to manage an Internet-connected server. Some tasks, such as backups and ensuring system security, are regular and ongoing. Other tasks, such as setting up file sharing or adding a new user, are only performed from time to time. In each case a good system administrator will not only know how to do the task itself, but also how to do it in a secure, maintainable, extensible and efficient manner, in accordance with current industry best practice.

However, not every server has a dedicated, experienced system administrator. This is especially the case in small businesses, where there may be no IT staff at all. Therefore, the goal of the SME Server is to automate the activities performed by a good sysadmin, from simple tasks such as adding users right through to backups and other complex activities, to the point where they can be easily performed by someone with little or no technical knowledge of the system.

Principle 2: Simplicity

The SME Server is characterized by its tight focus on providing network server functions. For the end-user, the SME Server provides simple, extensible web-based management. For developers, the SME Server provides clean, consistent, extensible interfaces to allow features to be added and modified.

Some Linux distributions are aimed at desktop users, general purpose server applications, or "enterprise" server applications. The SME Server is different in that it is targeted towards providing network server functionality for small to medium enterprises. Because of this, the SME Server is much smaller than many other Linux distributions, as software packages which are not needed for this purpose (for example, the X window system) are not included in the distribution.

The SME Server is also simple for a non-technical person to manage. For an end-user administering the server, choices are kept to a minimum. If a decision is very likely to be the same for all small businesses, the answer is assumed and the end user is not required to make a choice. When decisions are required, they are phrased in terms independent of the underlying technology, so that end-users are not required to be intimately familiar with Linux or Linux applications.

For developers, the simplicity is in the architecture of the SME Server system. Features are layered in such a way that additional features can be added without affecting the current services, and often without requiring modifications to the user interface.

Principle 3: Extensibility

SME Server's third design goal is extensibility, which provides a balance to the simplicity previously described. Since the simplest possible server will not suit every need, we make it easy to customize and extend the server in a number of ways.

Firstly, interfaces are provided for experienced users to customize the system from the Linux command line. These include tools to manipulate the configuration database, trigger events, or modify the configuration files for the various software installed on the system.

Secondly, applications allows developers to create additional software modules which can be easily installed and configured by end-users. Applications may provide application software for the server's users, administration tools, network services, or any other type of software or data.

The SME Server architecture explicitly supports developers by making it easy to drop software into place and remove it without needing to modify existing files. For instance, a web application does not need to edit the web server configuration file, but can simply drop a template fragment into the appropriate directory on the system and be assured that it will be expanded into the configuration file as required.

Principle 4: Reliability

The SME Server is designed to run without intervention 24 hours per day, seven days a week. This reliability has been designed from the ground up: stable, well supported versions of the Linux kernel and applications, RAID disk mirroring, automatic firewall, and process supervision. Where applications have been shown to be insecure or unreliable, we use stable, secure replacements. The modular architecture allows this to be done without affecting the system administrator's view of the system and with only localised effect on the developer's view.

Chapter 5. Architecture overview

The SME Server consists of a simplified CentOS installation, together with a number of server applications, and a layer of software that manages those server applications. The management software presents users with a simplified user interface and automatically configures the server applications as necessary.

The applications are not recompiled or modified to work within the SME Server framework. Rather, the framework automates the tasks of an experienced system administrator, and configures each application in a sensible, standard way.

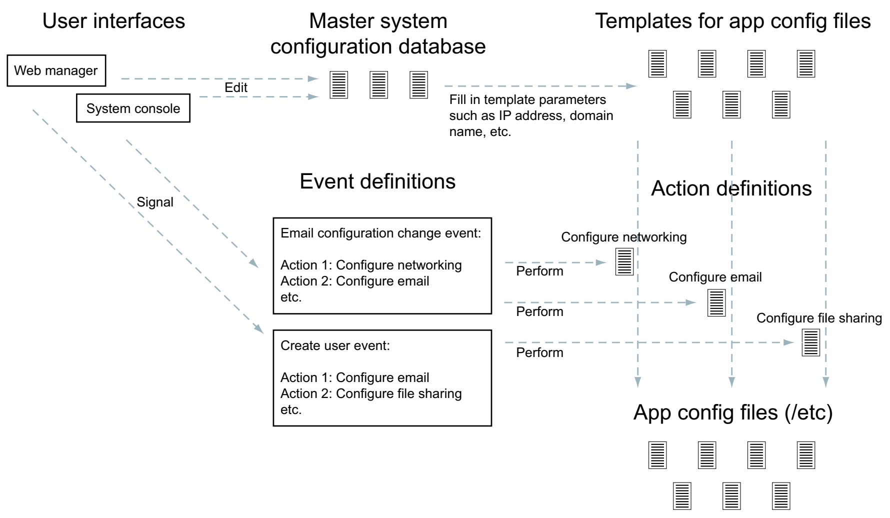
The SME Server framework has four components:

- server-manager and console user interfaces
- configuration databases
- template system, used to generate configuration files
- events and actions

When a user configures an aspect of the server through one of the user interfaces, the SME Server automatically configures the server applications relevant to that change. The SME Server does so using these steps:

- The user interface changes values in the configuration database. This database (actually a collection of databases) contains parameters describing the state of the system (IP address assignments, policy settings, domain names, email server configuration, user accounts, and so on). The user interface does not perform the application reconfiguration, but instead signals an *event* to perform the changes.
- The event relevant to the changes being made to the configuration database is signalled. For example, changes related to email configuration might signal the "email-update" event. These events are collections of scripts and an event can be extended to perform additional functions by adding scripts to the event directory. The *actions* for an event are run in a defined order to produce the desired system state.
- The actions within the event ensure that the configuration files used by the server applications are configured correctly. This is done by combining "templates" for the configuration file with the values in the configuration database.
- The actions then inform the applications that their configuration has been changed and that the application should re-read the file, or restart, as appropriate.

Figure 5-1. SME Server Architecture


User interfaces
Master system configuration database
Templates for app config files
Web manager
System console
Edit
Fill in template parameters such as IP address, domain name, etc.
Signal
Event definitions
Action definitions
Email configuration change event:
Action 1: Configure networking
Action 2: Configure email
etc.
Configure networking
Perform
Configure email
Perform
Configure file sharing
Create user event:
Action 1: Configure email
Action 2: Configure file sharing
etc.
Perform
App config files (/etc)

So, to recap: here are the steps performed when a system parameter is changed in the user interface (the same steps are used both for the console and for the web-based manager):

- The user interface code modifies the settings in the configuration databases to specify the new system configuration.
- The user interface code signals an event to inform the system that the configuration has changed.
- The event triggers a sequence of actions.
- The actions process a set of templates in order to generate new configuration files based on the current settings and reconfigure services where necessary.

II. SME Server internals

Chapter 6. Configuration database

Overview

All user-modifiable configuration parameters on the SME Server are stored in the configuration database. These values are used to generate the system configuration files, such as those found in the `/etc/` directory.

The configuration databases may be modified by various programs on the system, including the SME Server manager, the SME Server console, or scripts run from the command line by a system administrator.

Each entry in the database is either a simple key/value pair or a key and a collection of related property/value pairs.

> **Note:** The section describes the general structure of the configuration database. The actual entries and properties are subject to change between releases.

Simple entries

Simple configuration database entries take the form of a key/value pair:

```
[root@gsxdev1 ~]# config show AccessType
AccessType=dedicated

[root@gsxdev1 ~]# config show ConsoleMode
ConsoleMode=login

[root@gsxdev1 ~]# config show TimeZone
TimeZone=Australia/NSW
```

Complex entries

More complex entries consist of a key, a type, and a collection of property/value pairs:

```
[root@gsxdev1 ~]# config show atalk
atalk=service
    MaxClients=20
    status=enabled

[root@gsxdev1 ~]# config show dhcpd
dhcpd=service
    end=192.168.1.250
    start=192.168.1.65
    status=disabled
```

In most cases, complex entries are used in preference to simple entries. The complex entries allow additional properties to be stored for an entry, which enhances the system's flexibility.

Access from the command line

You can access configuration database entries from the command line using the **config** command, as shown above, or the **db** command. The **config** command provides a shorthand for accessing the *configuration* database. The following commands are equivalent:

```
[root@gsxdev1 ~]# config show LocalIP
LocalIP=192.168.1.100

[root@gsxdev1 ~]# db configuration show LocalIP
LocalIP=192.168.1.100
```

> **Note:** The term *configuration database* is used both to refer to the "master" *configuration* database and to refer collectively to the set of configuration databases, which includes the individual *accounts*, *networks*, and *configuration* databases.

The **db** allows you to access all of the databases. For example to show the details of the `admin` entry from *accounts*

```
[root@gsxdev1 ~]# db accounts show admin
admin=system
    EmailForward=local
    FirstName=Local
    ForwardAddress=
    LastName=Administrator
    Lockable=no
    PasswordSet=yes
    Removable=no
    Shell=/sbin/e-smith/console
    VPNClientAccess=no
```

Documentation for the `db` command is displayed if you run it without providing any arguments:

```
[root@gsxdev1 ~]# db
usage:
    /sbin/e-smith/db dbfile keys
    /sbin/e-smith/db dbfile print [key]
    /sbin/e-smith/db dbfile show [key]
    /sbin/e-smith/db dbfile get key
    /sbin/e-smith/db dbfile set key type [prop1 val1] [prop2 val2] ...
    /sbin/e-smith/db dbfile setdefault key type [prop1 val1] [prop2 val2] ...
    /sbin/e-smith/db dbfile delete key
    /sbin/e-smith/db dbfile printtype [key]
    /sbin/e-smith/db dbfile gettype key
    /sbin/e-smith/db dbfile settype key type
    /sbin/e-smith/db dbfile printprop key [prop1] [prop2] [prop3] ...
    /sbin/e-smith/db dbfile getprop key prop
    /sbin/e-smith/db dbfile setprop key prop1 val1 [prop2 val2] [prop3 val3] ...
    /sbin/e-smith/db dbfile delprop key prop1 [prop2] [prop3] ...
```

Access via the Perl API

You can also access configuration database entries programmatically using the `esmith::ConfigDB` and related Perl modules, which are abstractions for the `esmith::DB` module.

For example, we can retrieve and show the admin account details like this:

```
use esmith::AccountsDB;

my $db = esmith::AccountsDB->open or die "Couldn't open AccountsDB\n";

my $admin = $db->get("admin") or die "admin account missing from AccountsDB\n";

print $admin->show();
```

This code fragment would display the same information as running the **db accounts show admin** command we saw previously.

```
admin
     EmailForward = local
        FirstName = Local
   ForwardAddress =
         LastName = Administrator
         Lockable = no
      PasswordSet = yes
        Removable = no
            Shell = /sbin/e-smith/console
  VPNClientAccess = no
             type = system
```

The Perl API will be covered in more depth in the exercises later in this manual. For documentation on the API, log into the SME Server and browse the documentation using the `perldoc` command:

```
perldoc esmith::ConfigDB
perldoc esmith::AccountsDB
perldoc esmith::HostsDB
perldoc esmith::NetworksDB

perldoc esmith::DB
```

Database initialization

The configuration databases are initialized from files in the `/etc/e-smith/db/` hierarchy. These files can perform one of three actions:

- Create a database entry and set it to a default value, if the entry does not already exist.
- Force a database entry to a specific value, regardless of its current setting.
- Migrate an entry from a previous value to a new value.

This design allows each package to provide part of the system configuration, or migrate the system configuration values as required. Note that a single database property can only be "owned" by one package. Database initialization is run during system install, system upgrade and after new software has been installed.

If you examine the `/etc/e-smith/db/configuration/` directory you will see three subdirectories: `defaults/`, `force/` and `migrate/` to match the three options above. A similar structure exists for each of the other databases. A new database can be created by populating a new directory tree under the `/etc/e-smith/db/` directory.

```
[root@gsxdev1 db]# cd /etc/e-smith/db
[root@gsxdev1 db]# ls
accounts       domains       networks       yum_installed
backups        hosts         spamassassin   yum_repositories
configuration  mailpatterns  yum_available  yum_updates

[root@gsxdev1 db]# ls configuration/
defaults  force  migrate
```

Defaults files

Defaults files are simple text files. If the corresponding database key/property already exists, it is skipped. Otherwise, the key/property is created and the value loaded. For example, this file:

```
[root@gsxdev1 db]# cat configuration/defaults/sshd/status
disabled
```

would create the `sshd` database entry if it doesn't already exist, create the `status` property for that entry, again if it doesn't already exist, and finally set the status property to `disabled`.

Force files

Force files are just like defaults files, except they *overwrite* the existing value. So, this file:

```
[root@gsxdev1 db]# cat configuration/force/sysconfig/ReleaseVersion
7.0rc2
```

would create the ReleaseVersion property of the sysconfig entry and unconditionally set its value to `7.0rc2`

Migrate fragments

Migrate fragments are small pieces of Perl text which can be used to perform more complex migrations than is possible with defaults and force files. They would normally be used to replace database keys or properties with new names, or to adjust policy settings during an upgrade.

Each fragment is passed a reference to the current database in the `$DB` variable. This variable is an instance of the appropriate esmith::DB subclass, e.g. `esmith::AccountsDB` when the `accounts` database migrate fragments are being executed. This means that you can use the methods of that subclass, for example `esmith::AccountsDB->users()`.

Here is an example of a migrate fragment, which replaces the outdated `popd` entry with the new name `pop3`:

```
{
    my $popd = $DB->get("popd") or return;

    my $pop3 = $DB->get("pop3") ||
        $DB->new_record("pop3", { type => "service" });

    $pop3->merge_props($popd->props);

    $popd->delete;
}
```

This fragment checks whether the database (the configuration database in this case) has a `popd` entry. If that entry does not exist, the migrate fragment returns immediately. If the `popd` entry exists, we need to convert it, so we retrieve the `pop3` entry (or create it if it doesn't already exist). We then merge the properties from the `popd` entry into the `pop3` entry and finally delete the `popd` entry.

If this migrate fragment is run again, it will return immediately as the `popd` entry has already been deleted.

Important notes about migrate fragments

- Please be careful with migrate fragments. Although they should only modify entries within the current database, there are no restrictions placed on what they can do. The ability to open and even modify other databases may be required to perform a migration.
- Migrate fragments must be safe to run multiple times. They should migrate the value when required and do nothing in other cases.
- Migrate fragments should never call croak or die. This will cause the database migration to stop. If an error is detected, call carp or warn to note the error in the logs.
- Migrate fragments should be owned by the package requiring the migration so that the migration only occurs when that package is installed.
- Migrate fragments should be self-contained and ideally perform only one migration per fragment.
- It is also possible to initialize and migrate database values in action scripts, but creation of migrate fragments is *strongly* preferred. Creating defaults is a simple matter of creating text files and migrate fragments require far less code than action scripts.

Evaluation order: migrate, defaults, force

When a database is loaded:

- migrate scripts are run first
- then defaults are loaded
- and finally any force files are loaded.

This order allows migration of old format entries to occur prior to loading of new default values. Remember, defaults will not change an existing database property.

Forcing database initialization

The database is initialized during a number of events, including **console-save**, so a call to **signal-event console-save** will evaluate all of the database fragments.

> **Note:** The **console-save** event is not a "reconfigure everything" event, and only changes items which can be configured from the text-mode console. It is convenient in this case as it performs database initialization and migration.
>
> It is an SME Server requirement that all database entries and configuration files must be correctly configured after a "reconfiguration reboot". This is available from the console and server manager and performs the **post-upgrade** and **reboot** events. Packages should also provide links in other events (e.g. "email-update" for email related changes) to provide reconfiguration without the reboot.

Important notes about the configuration databases

- The configuration databases should only be modified using the tools and APIs provided.
- The order of the entries and the order of properties is undefined.
- The keys and property names are currently treated in a *case-sensitive* manner, though this may change in the future. *Do not create keys or property names which differ only by their case.*
- Underscores and hyphens are valid in key and property names, but should normally be avoided.
- Do not "overload" an existing property with a new value. If the existing values do not meet your requirements, discuss your implementation with the developers. Values which are not known by the base may cause serious issues on upgrade. If the existing panels have three choices, do not invent new choices without enhancing the panel to support them.
- The `type` pseudo-property is used internally and is *reserved*.
- By convention, database keys are lower case, and property names are stored in mixed case. The `type`, `status` and `access` properties are exceptions to this convention.
- The storage location and internals of the databases is subject to change.
- The configuration databases are currently stored as pipe-delimited flat text files in the `/home/e-smith/db/` directory.

The configuration databases

Configuration

The most important database is the (master) configuration database. This database describes how the system should operate; the type of Internet access to use, how email should be handled, and so on.

The configuration database contains a mix of simple and complex entries, although all new entries are complex entries.

Accounts

Account details are stored in the `accounts` database, as complex entries. We classify accounts into several types, including:

- User accounts: These are accounts created for individual users at the local organization. Each account has a POP/IMAP mailbox and an area for storing files.
- Groups: Groups of users, which can be used for configuring permissions on storage areas and automatically provide a group e-mail address.
- Information bays: These accounts correspond to information bays defined in the system. These storage areas can be accessed via filesharing, FTP and the web.
- System accounts: Linux system accounts which are reserved by installed software packages.
- URL accounts: Portions of the Web namespace which are reserved for system use. For example, the *server-manager* account is reserved as it is used for redirecting web access to the server manager.
- Pseudonyms: Alternate names for existing accounts. For example, `fred.frog` could be a pseudonym for the account `ffrog`, allowing email to be sent to either address.
- Printers: Network shared printers share the same namespace as other accounts so that they can be made visible to the local network.

Domains

The domains database shows the domains handled by this server, including information about how to handle web requests, and the DNS servers for the domain.

Networks

The networks database details the networks which should be treated as local by this server. Local networks have additional access rights which are denied for other networks.

Hosts

The hosts database decribes all hosts/machines known to this server and is used to generate DHCP and DNS configuration.

Other configuration databases

There are several other configuration databases stored with the ones listed above, and the system design allows for additional databases to be created as required.

Namespace issues

All entries in a single database share the same namespace. Users, groups, information bays, printers, and other entries in the accounts database currently all share one namespace. This means that you cannot have a user with the same name as an information bay, group or other entry in the accounts database.

However, it would be possible to have a host named `fredfrog` as well as a user named `fredfrog` as they are stored in separate databases and thus different namespaces.

Chapter 7. Actions and events

Actions

An action is a program, frequently written in a scripting language, which performs a single task. It is typically an encapsulation of a task usually done by a system administrator, such as editing a configuration file or reconfiguring a service. Actions are not called directly; they are always called by signalling an event.

The actions are stored in the `/etc/e-smith/events/actions/` directory. These actions are then linked into the relevant events as the same action may need to be performed in more than one event.. To create a new action called `myaction` you simply create a program to perform the action `myaction` and save it as `/etc/e-smith/events/actions/myaction` . Actions can be written in any programming language, although additional platform support is provided for Perl code.

An example action script is `set-external-ip` which is called when the external IP address changes. Here's the body of that script (at time of writing):

```
package esmith;

use strict;
use Errno;
use esmith::ConfigDB;

my $db = esmith::ConfigDB->open or die "Couldn't open ConfigDB\n";

my $event = $ARGV[0];
my $newip = $ARGV[1];

$db->set_value('ExternalIP', $newip);
$db->set_prop('ExternalInterface', 'IPAddress', $newip);

exit 0;
```

This script sets the `ExternalIP` value and the `IPAddress` property of the `ExternalInterface` record in the configuration database to the value provided as a parameter. The `$event` parameter is not used in this particular script.

> **Note:** The two records exist due to an partial migration from simple to complex entries in the configuration database. Setting both values in this script avoids the need to perform database migration in the ip-change event.

Action script parameters

Action scripts are always called with at least one parameter; the name of the current event. Many action scripts, such as `set-external-ip` , are called with a single additional parameter. This parameter is usually a configuration database key, for example the username being modified or the new IP address.

*Action scripts rarely require more than two parameters.*The details should be stored in the configuration database(s) and only the key should be passed to the action scripts. Events are not meant to be used as function calls. All configuration details must be stored in the configuration databases and the database key passed as the parameter to the action. This allows other scripts to be added to the event.

Since the SME Server passes the name of the current event as the first parameter, it is often beneficial to write action scripts which are polymorphic based on the event name. For example, the code to create a user and the code to modify an existing user may be only slightly different and may well benefit from being in a single script.

Events

Events are a mechanism which allows the system to trigger a set of actions in response to actual events that happen on the system. When one of the users interfaces modifies the configuration databases, it must signal an event to regenerate the various server application configuration files according to the new configuration. *The user interface must never modify configuration files directly.*

Each event is associated with a list of actions which should be performed when that event occurs and is defined as a subdirectory of `/etc/e-smith/events/` containing symbolic links to the appropriate actions, loosely modelled after the *System V init* mechanism for starting servers. For example, if you examine the `/etc/e-smith/events/ip-change` directory:

```
lrwxrwxrwx  1 root root   26  S15set-external-ip -> ../actions/set-external-ip*
lrwxrwxrwx  1 root root   21  S85update-dns -> ../actions/update-dns*
drwxr-xr-x  2 root root 4096  services2adjust/
drwxr-xr-x  5 root root 4096  templates2expand/
```

The symbolic links are given prefixes such as S15, S85, etc. to specify the order in which the actions should be executed in a similar manner to the System V init mechanism.

You can change the actions performed by an event by changing the links in the event directory. You can also create a new event by creating another subdirectory of `/etc/e-smith/events/`.

Implicit actions: services2adjust and templates2expand

Most events contain two common tasks: expanding various templates and adjusting (e.g. restarting) the relevant services. For this reason, two implicit actions are included in all events. These implicit actions mean that additional code does not need to be written to perform these common tasks. The implicit actions are represented by entries in the `services2adjust/` and `templates2expand/` subdirectories.

services2adjust

The `services2adjust/` directory contains links mapping a specific service to the action to perform on that service. For example, if signalling the event in question requires that the `ntpd` service is restarted, you simply include the link `ntpd -> restart` in the `services2adjust` directory. The implicit action services2adjust would then restart the ntpd service. As an example, the `services2adjust/` directory for the `ip-change` event is shown below:

```
lrwxrwxrwx  1 root root 6  masq -> adjust
lrwxrwxrwx  1 root root 7  ntpd -> restart
lrwxrwxrwx  1 root root 7  pptpd -> sigterm
lrwxrwxrwx  1 root root 6  qmail -> sighup
lrwxrwxrwx  1 root root 7  tinydns -> sigusr2
```

templates2expand

The `templates2expand/` directory contains a list of the configuration files which need to be regenerated from their templates. This list consists of a collection of empty files with the same file name as the configuration file to be expanded and in a heirarchy mirroring their location on the system. For example, to expand templates for the `/etc/samba/smb.conf` configuration file, simply include the empty file `etc/samba/smb.conf` in the `templates2expand/` directory of the relevant event(s). For more detail, see the Section called *Mapping templates to events: templates2expand* in Chapter 8.

Order of implicit actions

The implicit actions are implemented by inserting the action script `generic_template_expand` early in the list of actions to be run in an event and the `adjust-services` action near the end of the list.

You should normally link your action scripts in the range `S10` to `S80` so that they occur after `templates2expand` and before `services2adjust`.

> **Note:** The `generic_template_expand` action is currently run at `S05` and `adjust-services` is run at `S90`. The order of action scripts within an event is subject to change between releases.

Signalling events

The `signal-event` program takes an event name as an argument, and executes all of the actions in that event, providing the event name as the first parameter and directing all output to the system log. It works by listing the entries in the event directory and executing them in sequence. So for example, the command:

```
signal-event console-save
```

will perform all the actions associated with the `console-save` event, which is defined by the contents of the `/etc/e-smith/events/console-save/` directory. This is exactly what the console user interface does when you select save at the end of the console configuration wizard.

Events with arguments

So far we have described the following general principle throughout the SME Server; changes are made by altering the configuration files, then signalling events. The actions triggered by each event typically regenerate entire configuration files, taking into account the latest configuration information.

However, some changes are best made incrementally. For example, consider the `user-create` event. One of its actions updates the LDAP directory, which it could do by deleting all of the users and recreating them based on the updated `accounts` database. However, this is inefficient and would lose any additional LDAP attributes which may have been stored. It would be better to simply add the new user incrementally, using the default LDAP schema.

But how is the action code to know which user was just added? The new username is passed as an argument to the user-create event. This way the action programs triggered by the user-create event have a choice. They can either ignore the username argument and regenerate their output based on the updated list of accounts, or they can pay attention to the username argument, retrieve the rest of the information about the new user from the `accounts` database, and perform the incremental work to add the user.

> **Note:** Reminder: action scripts should normally take at most two arguments. The first is always the event name. The second optional argument is a key into one of the databases. Events are not function calls.
>
> Events are not currently serialized. In most cases overlapping events will not cause issues, but caution should be exercised when events are signalled from programs.

Standard events and their arguments

The table below summarises the key SME Server events and their argument if required. Remember, each action script is always called with the event name as the first argument. The arguments listed in this table are provided as the second argument.

> **Note:** Events which are not listed in this table are subject to change and may not appear in future releases of the SME Server.

Table 7-1. SME Server standard events

Event	Argument	Description
bootstrap-console-save	(none)	Expands all templates in the system. It is a *requirement* that all templates are correct after a combination of post-upgrade/reboot. Called after the initial console wizard, after system upgrades, and as part of a reconfiguration reboot.

Event	Argument	Description
console-save	(none)	Expands templates and reconfigures services which can be changed from the text-mode console and which do not require a reboot. Services which do require a reboot for configuration will be handled by bootstrap-console-save. The console-save event is *not* a general "reconfigure everything" event.
email-update	(none)	Reconfigures services listed on the e-mail panel.
group-create, group-delete, group-modify	Group - key into accounts database	Called when a group is created/deleted/modified.
halt	(none)	Called when the system is being shutdown prior to power off.
host-create, host-delete, host-modify	Host - key into hosts database	Called when a host is created, deleted or modified.
ibay-create, ibay-delete, ibay-modify	Ibay - key into accounts database	Called when an information bay is created/deleted/modified.
ip-change	New external IP address	Called when the external IP address changes, e.g. through a new PPPoE connection or DHCP lease.
local	(none)	Called after each reboot. Customisations which would normally require modification of the `/etc/rc.local` file should instead be installed as individual scripts in the `/etc/e-smith/events/local/` event directory.
network-create, network-delete	Network - key into networks database	Called when a local network is created or deleted.
password-modify	User - key into accounts database	Called when a user password is modified, including when the account is unlocked.
post-upgrade (and post-install)	(none)	Called as final step of the CD upgrade (install). *This event must be immediately followed by a reboot.* The bootstrap-console-save event is then called after the reboot to complete the reconfiguration. The only changes which should occur in this event are ones which must be performed prior to the reboot (e.g. configuring the boot loader). The post-install event is only called once, from the CD installer.

Event	**Argument**	**Description**
pre-backup, post-backup	Cause - type of backup being performed (e.g. "tape")	The pre-backup event creates consistent system state for the backup. For example, it creates an ASCII dump of the MySQL databases. If the pre-backup event fails, the backup is *not* run. The post-backup is called if the backup is successful and removes the state files generated by pre-backup.
pseudonym-create, pseudonym-delete, pseudonym-modify	Pseudonym - key into accounts database	Called when a pseudonym is created/deleted/modified.
reboot	(none)	Called when the system is being shutdown prior to a reboot.
remoteaccess-update	(none)	Reconfigures services listed on the Remote Access panel and updates the firewall rules for all services.
user-create, user-delete, user-modify	User - key into accounts database	Called when a user is created/deleted/modified.
user-lock	User - key into accounts database	Called when a user account is locked.

Handling deletions

When adding a user, the user is created in the `accounts` database, and various actions, such as creating the Linux account, are performed in the `user-create` event. However, when deleting a user, we want to maintain the `accounts` database entry for as long as possible, in case there is information which the actions in the `user-delete` event might need in order to cleanly delete the users.

The SME Server convention for handling deletions is:

- Change the type of the entry to mark it as being in the process of being deleted e.g. a *user* entry becomes a *user-deleted* entry.
- Signal the relevant deletion event - e.g. *user-delete*
- Remove the entry from the database, but only if the event succeeds.

With this approach, the action scripts can decide whether to ignore the *user-deleted* entries when performing their tasks.

Event logs

All events, and all actions run by the event, are logged to the `messages` system log. Here is an example action log, which has been formatted onto multiple lines to enhance readability:

```
Feb  2 13:22:33 gsxdev1 esmith::event[4525]:
  S65sshd-conf=action|
  Event|remoteaccess-update|
  Action|S65sshd-conf|
  Start|1138846952 730480|
  End|1138846953 66768|
  Elapsed|0.336288
```

From this single log, we can see the action script name, which event it was called in, when it started, ended and how long it took (0.34 seconds). Now, let's add an action script which always fails and signal the event again:

```
Feb  2 16:11:54 gsxdev1 esmith::event[4787]:
  S99false=action|
  Event|remoteaccess-update|
  Action|S99false|
  Start|1138857114 58910|
  End|1138857114 81920|
  Elapsed|0.02301|
  Status|256
```

Note that this log has a new field `Status`, which is added if the action script returns a false (non-zero) exit status. Suppressing the Status field when it is zero (success) makes it much easier to find failed actions in the logs.

Failed events

If an action script fails, the entire event fails. The other actions scripts in the event are run, but the whole event is marked as having failed.

By convention, if a delete event fails, the user interface does *not* delete the entry from the relevant database. So, if the user-delete event fails, a "stray" `user-deleted` entry will appear in the accounts database. The event logs with Status properties can be matched with the user-deleted entries to determine which action script failed so it can be corrected in the future. This user-deleted entry will also block the creation of another account with that name until the issue is corrected.

Chapter 8. Configuration file templates

Design of the template system

Every piece of software has its own configuration format, and writing parsers for each one is a complex, time-consuming and error-prone process. The SME Server software avoids the whole issue by using templates which *generate* the correct configuration.

In most cases, SME Server configuration files are *over-written* when templates are expanded. In a few specific cases, the existing configuration file is parsed and rewritten in-place. This is done where the configuration file (e.g. `/etc/fstab`) is also automatically updated by some other process.

Templates are stored under `/etc/e-smith/templates/` in a directory hierarchy which matches the standard filesystem. For example, the template for `/etc/inittab` is stored in the `/etc/e-smith/templates/etc/inittab/` directory. Each template is stored as a directory of template fragments and processed by the Perl `Text::Template` module.

The template fragments are concatenated together in *ASCIIbetical* order (US-ASCII sort order) and the complete file is parsed to generate the appropriate configuration files for the service. The use of fragments is part of the SME Server's modular and extensible architecture; it allows third-party modules to add fragments to the configuration where necessary.

Note: It is also possible to store templates as single files, rather than as a directory of fragments. This method is preserved for backwards compatibility, but does not provide the extensibility of directory based templates. Directory templates should be used for all new templates, even if that directory only contains a single fragment.

The Text::Template module

The `Text::Template` module allows arbitary Perl code to be embedded in a template file by surrounding it in braces (`"{"` and `"}"`). The code inside the braces is interpreted and its return value replaces the section between, and including, the braces. For instance:

```
The answer is { 2 + 2 }
```

becomes

```
The answer is 4
```

Variables can be passed in from the program which is expanding the template, hence:

```
Shopping list:
{
    $OUT = ”;

    for my $item ( qw(bread milk bananas) )
    {
        $OUT .= "* $item\n";
    }
}
```

would expand to:

```
Shopping list:
* bread
* milk
* bananas
```

The SME Server template system uses this mechanism to automatically pass in global configuration variables from the `configuration` database which can then be used to fill out the configuration files.

For example, the /etc/hosts template is fairly simple and composed of two fragments:

```
[gordonr@smebuild hosts]$ pwd
/etc/e-smith/templates/etc/hosts

[gordonr@smebuild hosts]$ ls
10localhost  20hostname
```

Let’s look at those fragments. The first is a piece of static text, which `Text::Template` will include verbatim:

```
127.0.0.1       localhost
```

The second is more complex and relies on values from the configuration database:

```
{
    $OUT .= "$LocalIP\t";
    $OUT .= " ${SystemName}.${DomainName}";
    $OUT .= " ${SystemName}";
}
```

Note that the whole fragment is enclosed in braces. Within those braces is a section of Perl code. When this template is expanded, it results in the following configuration file:

```
#------------------------------------------------------------
#              !!DO NOT MODIFY THIS FILE!!
#
# Manual changes will be lost when this file is regenerated.
#
# Please read the developer’s guide, which is available
# at http://www.contribs.org/development/
#
# Copyright (C) 1999-2006 Mitel Networks Corporation
#------------------------------------------------------------

127.0.0.1       localhost
192.168.10.1    smebuild.gormand.com.au smebuild
```

The header block comes "for free" as part of the template system, courtesy of an optional file `template-begin`, which is always processed as the first fragment. If it isn't provided, the text shown with # comments is included.

The other lines are provided by the two fragments shown above. Note the use of the configuration database variables: `$LocalIP`, `$SystemName` and `$DomainName`. All simple entries in the configuration database are provided as global variables to the templates.

Note that all of the template fragments are concatenated together before evaluation, so it is possible to set values in fragments which are used in later fragments. This is a very useful model for reducing the code in individual template fragments.

The complex entries in the configuration database are also provided as global variables to the templates. However, they are provided as Perl hashes instead of simple scalars. For example, here is how you might configure the Network Time Protocol (NTP) server `/etc/ntp.conf` file:

```
server { $ntpd{NTPServer} }
driftfile /etc/ntp/drift
authenticate no
```

The `NTPServer` setting is stored in the `ntpd` configuration database record, and so can be accessed via the hash accessor `$ntpd{NTPServer}`.

template-begin and template-end

Each template directory can contain two optional files `template-begin` and `template-end` . The template-begin file is always processed as the first file of the template, and the template-end file is always processed as the last file.

If the directory does not contain a `template-begin` file, the contents of `/etc/e-smith/templates-default/template-begin` is used automatically.

If the directory does not contain a `template-end` , nothing is appended to the template output. It is mostly used to provide the closing block for configuration files written in languages such as HTML and PHP, through a link to an entry in the `templates-default/` directory.

/etc/e-smith/templates-default

The `/etc/e-smith/templates-default` directory contains a set of template-begin and template-end files for various languages. For example, if your template generates a perl script, you would link `template-begin` to `/etc/e-smith/templates-default/template-begin-perl` and automatically get the `#!/usr/bin/perl -w` line and a comment containing the contents of the default template-begin file.

```
[gordonr@sevendev1 devguide]$ ls /etc/e-smith/templates-default/
template-begin       template-begin-perl   template-end-php
template-begin-html  template-begin-php
template-begin-pam   template-begin-shell
```

Note: You may also need a `templates.metadata` configuration file if your generated file needs to be executable.

Template fragment ordering

Template fragments are assembled in ASCII-betical order, with two exceptions: template-begin always comes first, and template-end always comes last. Template fragments are often named to start with a two digit number to make the ordering obvious, but this is not required.

Note: The number of fragments and the order of those fragments within a template directory is subject to change between releases.

Templates for user home directories: templates-user

Most of the templates on the system map to single, fixed output files, such as `/etc/hosts`. However, templates are also used to generate configuration files such as mail delivery instructions for users. These templates are stored in the `/etc/e-smith/template-user/` tree.

For example, the template for the `.qmail` file in user home directories (which details how mail is to be handled), is stored under `/etc/e-smith/template-user/.qmail/`. As these templates have a variable output filename, they are expanded using small pieces of Perl code in action scripts.

Local site overrides: templates-custom and templates-user-custom

It is possible that the standard templates are not correct for a particular installation, and so the local system administrator can override the extsing templates by placing files in the `templates-custom` tree. This is a parallel tree to the normal templates hierarchy, and is normally empty. There is also a `template-user-custom` tree for overriding entries in the templates-user tree.

Note: *Never edit the standard templates.* Your changes will be overwritten when packages are upgraded.

Note: The template-custom trees should be reserved for local system overrides. *Software should not install files in this tree.*

If a templates-custom entry exists for a template, it is merged with the standard templates directory during template expansion, using the following rules:

- If a fragment of the same name exists in both templates and templates-custom, the one from templates-custom is used, and the one from the standard templates tree is ignored.
- If the fragments in templates-custom have different names from those in templates, they are merged into the template as if they were in the templates directory.
- If the templates-custom entry is a file, rather than a directory, it completely overrides the standard template.

To make this concrete, let's assume we have the following template structure:

```
/etc/e-smith/templates/etc/book.conf:
10intro
30chapter3
40chapter4
80synopsis
```

and

```
/etc/e-smith/templates-custom/etc/book.conf:
30chapter3
50chapter5
```

The resulting template would be processed in this order:

- template-begin from /etc/e-smith/templates-default
- 10intro from /etc/e-smith/templates/etc/book.conf
- 30chapter3 from /etc/e-smith/templates-custom/etc/book.conf
- 40chapter4 from /etc/e-smith/templates/etc/book.conf
- 50chapter5 from /etc/e-smith/templates-custom/etc/book.conf
- 80synopsis from /etc/e-smith/templates/etc/book.conf
- template-end (empty), nominally from /etc/e-smith/templates-default

How to resolve conflicts with standard templates

It is possible that the standard templates may specify behaviour which is not appropriate for your application. In many cases the templates will be driven by configuration database settings which allow their behaviour to be customized, which should be the first thing to check.

In many cases, your application only needs to *extend* the behaviour of the template by adding one or more fragments. This should be your second option and can be achieved by simply adding your fragment in the correct place in the list of fragments.

In rare cases the standard template specifies a behaviour which conflicts with your application. In these cases, you should do *all* of the following:

- Create a templates-custom directory to match the existing one in the templates hierachy.

- Copy the conflicting fragment, and only that fragment, to the templates-custom directory. The fragment should have the same name in both directories. At this point you have not changed the behaviour of the system as the templates-custom entry will be preferred, but will behave identically.
- Modify the copy in templates-custom to suit your required behaviour.
- Raise a New Feature Request here: http://www.contribs.org/bugzilla/. Please attach your modified template (or even better, a patch file) and provide details of why you think that the standard template should be changed.

> **Note:** You should not release RPMs which install templates in the `templates-custom` directories. If the behaviour of a base template needs to be changed, please raise a bug to discuss the change.

Subdirectory templates

It is also possible to split templates into further subdirectories. This can be very useful for evaluating the same fragments in a loop, for example for each virtual domain in `httpd.conf` or each ibay in `smb.conf`.

Two examples of this can be found in `/etc/e-smith/templates/etc/httpd/conf/httpd.conf/80VirtualHosts` which loops over the `/etc/e-smith/templates/etc/httpd/conf/httpd.conf/VirtualHosts/` directory, and `/etc/e-smith/templates/etc/smb.conf/90ibays` which performs a similar loop over the `/etc/e-smith/templates/etc/smb.conf/ibays/` directory.

Template expansion

Mapping templates to events: templates2expand

The SME Server is designed to ensure consistent and reliable operation, without requiring command-line access. Whenever an event is signalled, the relevant templates for that event are expanded and the services are notified of the configuration changes.

Requesting expansion of a template in an event is a simple matter of creating an empty file under the `templates2expand` hierarchy for that event. For example, here are the templates which are expanded during an `ip-change` event:

```
[gordonr@smebuild templates2expand]$ pwd
/etc/e-smith/events/ip-change/templates2expand

[gordonr@smebuild templates2expand]$ find . -type f
./etc/services
./etc/pam.d/passwd
./etc/dhcpd.conf
./etc/pptpd.conf
```

```
./etc/securetty
./etc/hosts.deny
./etc/shells
./etc/proftpd.conf
./etc/fetchmail
./etc/ppp/options.pptpd
./etc/ppp/ip-down.local
./etc/ppp/ip-up.local
./etc/hosts.allow
./etc/startmail
./var/qmail/alias/.qmail-localdelivery-default
./var/qmail/alias/.qmail-default
./var/qmail/control/concurrencylocal
./var/qmail/control/me
./var/qmail/control/virtualdomains
./var/qmail/control/smtproutes
./var/qmail/control/plusdomain
./var/qmail/control/doublebounceto
./var/qmail/control/rcpthosts
./var/qmail/control/badhelo
./var/qmail/control/databytes
./var/qmail/control/mailrules.default
./var/qmail/control/helohost
./var/qmail/control/bouncehost
./var/qmail/control/envnoathost
./var/qmail/control/defaultdomain
./var/qmail/control/locals
./var/qmail/control/bouncefrom
./var/qmail/control/defaulthost
./var/qmail/control/concurrencyremote
./home/e-smith/.qmail
```

It is important to note that any package can request a template expansion for an event. The list shown above has been contributed by a number of packages, and some of those packages have requested expansion of more than one template:

```
[gordonr@smebuild templates2expand]$ find . -type f|xargs rpm -qf | sort | uniq
e-smith-base-4.15.6-01
e-smith-email-4.15.4-01
e-smith-pptpd-1.11.0-18
e-smith-proftpd-1.11.0-25
e-smith-qmail-1.9.0-11
smeserver-qpsmtpd-1.0.1-09
```

Template permissions and ownership: templates.metadata

Templates are normally expanded to be owned by `root` and are not executable, which is a reasonable default for most configuration files. However, templates may need to generate configuration files which are owned by a different user, or which need to be executable or have other special permissions. This can be done by creating a `templates.metadata` file which defines the additional attributes for the expansion.

> **Note:** Configuration files should generally *not* be writable by any user other than root. In particular, configuration files should not normally be writable the *www* user as this poses a significant security risk. Installation advice which says "chmod 777" is almost invariably wrong.

For example, here is the metadata file `/etc/e-smith/templates.metadata/etc/ppp/ip-up.local`:

```
UID="root"
GID="daemon"
PERMS=0755
```

which sets the group to `daemon` and makes the script executable. Note that the file is readable by members of the `daemon` group, but it is not writable by anyone but root. It is also possible to use the same template to generate multiple output files, such as in this example:

```
TEMPLATE_PATH="/etc/sysconfig/network-scripts/route-ethX"
OUTPUT_FILENAME="/etc/sysconfig/network-scripts/route-eth1"
MORE_DATA={ THIS_DEVICE => "eth1" }
FILTER=sub { $_[0] =~ /^#/ ? ” : $_[0] } # Remove comments
```

The templates.metadata file for route-eth0 just uses `eth0` instead of `eth1` on the second and third lines. Note also the `FILTER` setting which allows post-processing of the generated template.

There are many examples under `/etc/e-smith/templates.metadata/` and the full list of options can be seen with:

```
perldoc esmith::templates
```

Manual testing: expand-template

It is sometimes useful to expand templates manually during testing, which can be done with the **expand-template** command. The syntax of this command is simply:

```
expand-template filename
```

where *`filename`* is the name of the configuration file you want to generate, e.g. `/etc/hosts`.

> **Note:** `expand-template` is designed for testing, and not as the standard way to expand templates. The correct way to ensure that a template is expanded is to create the `templates2expand` files in the relevant events, along with any `templates.metadata` files which may be required.

Perl API: processTemplate

In rare circumstances you may need to call **processTemplate** directly. Explicit calls to **processTemplate** are typically only used when the output filename is variable, such as when processing the `.qmail` files for each group:

```
use esmith::templates;

foreach my $group (@groups)
{
    my $groupName = $group->key;

    [...]

    processTemplate(
            {
                CONFREF =>
                    {
                        Members => $members,
                    },

                TEMPLATE_PATH =>
                    "/var/qmail/alias/.qmail-group",

                OUTPUT_FILENAME => "/var/qmail/alias/.qmail-$groupName",
            }
        );

    [...]
}
```

Note: Software which was written for SME Server before release 7 will have a number of scripts which call **processTemplate**. In almost all cases, these can be replaced with simple flag files in the `templates2expand/` directory of the relevant events. The new method is *far* more efficient as a single invocation is perl is used to expand all template files.

Chapter 9. Process startup, supervision and shutdown

Process startup

In typical Linux systems, services (processes) are started at boot time through a mechanism such as *System V init*. When the system administrator needs to change the settings, they modify the configuration files and then restart the service or notify the process that it needs to re-read the configuration.

It is usually assumed that processes which have been started will continue to run, and only require intervention during configuration changes. There are a number of problems with this model, which are addressed by the SME Server:

- Processes do occasionally fail through software errors, memory exhaustion and accidental finger poking by the system administrator.
- Some startup scripts and processes do not gracefully handle server crashes, such as power outages. The startup scripts and processes often use process identifier (PID) files to determine whether the process is running. Reliable handling of PID files is impossible to achieve under all failure cases.
- Many processes do not deal properly with rapid invocation of stop and start requests. This is often, but not always, due to "PID file race" conditions.

Process supervision: runit (and supervise)

The SME Server addresses these issues by running processes under the **runit** process supervision environment, which:

- runs each process under control of its own supervisor process
- imposes process limits
- restarts the process if it fails
- provides a consistent mechanism for controlling the underlying process

Note: Gerrit Pape's **runit** came from previous work by Dan Bernstein on the **supervise** supervision environment. **runit** provides additional features, and has been released under a free software license.

The runit process tree

When a Linux system boots, it starts the **init** process, which then starts all other processes. When **init** enters "run-level 7", it starts **/etc/runit/2** from an entry in `/etc/inittab`.

`/etc/runit/2` starts the **runsvdir** master supervision process, which scans the `/service/` directory for work to do. If the **runsvdir** command happened to fail, it would be restarted by **init**.

The **runsvdir** command looks for subdirectories under the `/service/` directory, and starts a **runsv** process to manage that directory. If any of the **runsv** processes fail, they will be restarted by **runsvdir**.

Each **runsv** process looks for a `run` script under the directory it is managing. **runsv** runs the `run` script and keeps a connection to the process started by that script. If the process dies, it is restarted.

If the directory also has a `log` subdirectory, **runsv** runs `run` script in that directory and connects the output of the main program to the input of the "logger" process.

This produces a process tree which looks something like this:

```
[root@gsxdev1 events]# pstree 1
init-+-acpid
     |-md1_raid1
     |-md2_raid1
     | ...
     |-runsvdir-+-runsv-+-multilog
     |          |       '-ulogd
     |          |-6*[runsv---multilog]
     |          |-runsv-+-multilog
     |          |       '-ntpd
     |          |-runsv-+-multilog
     |          |       '-tinydns
     |          |-runsv-+-cvm-unix
     |          |       '-multilog
     |          |-runsv-+-multilog
     |          |       '-mysqld
     |          |-5*[runsv-+-multilog]
     |          |           '-tcpsvd]
     |          |-runsv-+-multilog
     |          |       '-oidentd
     |          |-runsv-+-multilog
     |          |       '-smtp-auth-proxy
     |          |-runsv-+-multilog
     |          |       '-smbd---smbd
     |          |-runsv---httpd---10*[httpd]
```

This looks like a complex process tree, but is a critical part of the SME Server's design for reliability. Each process is independent, has a consistent management interface, has process limits imposed on it, and will restart if it happens to fail.

> **Note:** For the curious, if init fails, the system reboots.

For further documentation on runit, refer to the runit manual page.

Run-level 7 and the e-smith-service wrapper

The SME Server runs in the normally unused run-level 7. This ensures that the only software running on the SME Server is software that we have chosen to run, and it is started and stopped in a consistent way. If we need to replace a standard startup script with one which runs the process under supervise, we can do so *without modifying the original package*.

In order to run a process under run-level 7, all you need to do is provide a link in the `/etc/rc.d/rc7.d/` directory to your startup script. However, in most cases your process should only start if it is enabled in the configuration database.

If you look at the `/etc/rc.d/rc7.d/` directory. you will see that it contains a large number of links to the `/etc/rc.d/init.d/e-smith-service` script.

```
S00microcode_ctl      -> /etc/rc.d/init.d/e-smith-service
S05syslog             -> /etc/rc.d/init.d/e-smith-service
S06cpuspeed           -> /etc/rc.d/init.d/e-smith-service
S15nut                -> ../init.d/e-smith-service
S15raidmonitor        -> /etc/rc.d/init.d/e-smith-service
S26apmd               -> /etc/rc.d/init.d/e-smith-service
S35bootstrap-console -> /etc/rc.d/init.d/e-smith-service
[...]
```

This script is key to ensuring that services start when they are enabled and do not start when they are disabled, as it:

- Checks the name of the link, e.g. `S05syslog`
- Removes the S05 prefix, leaving `syslog`
- Checks to see whether `syslog` is defined in the configuration database, and whether it has its `status` set to `enabled`.
- If so, it runs the `/etc/init.d/syslog` script with the argument `start`.
- If the service is not enabled, it exits without starting the service.

> **Note:** If a script exists in the `/etc/init.d/supervise/` directory, `e-smith-service` will use that in preference to the one in the `/etc/init.d/` directory. This allows us to install our own supervised startup scripts *without modifying the original package*.

Chapter 10. The server-manager web interface

The user interfaces to the SME Server (the web based server-manager and the text mode console interface) perform their work by modifying the master system configuration database to describe the new system configuration, then regenerating the various application configuration files by *signalling an event*.

This decoupling of the user interfaces from the system configuration allows packages to be added and removed without modifying the user interface code. It also allows all actions performed by the manager to be scripted, if this is desired. For example, if a new package needs to expand a template when users are created, it can just create the appropriate links in the `user-create` event.

The web directory

The primary files which make up the SME Server manager are kept in the `/etc/e-smith/web/` directory. These files define the layout of the web functions and require auxiliary files which provide translations and the implementation of the functions.

Table 10-1. Web interface directories

Name	Description
`/etc/e-smith/web/common/`	Common files such as images and page headers.
`/etc/e-smith/web/functions/`	Screen definitions, written in FormMagick XML. The scripts in this directory are linked into the `cgi-bin` directory of the panels in which they should appear.
`/etc/e-smith/web/panels/`	Top-level directory for panel definitions. Each panel is a collection of screens, presented as a single user interface.
`./manager/{cgi-bin,common,html}/`	Subdirectories for the HTML, CGI and common files for the "manager" panel, which is accessed by the */server-manager/* URL.
`./password/{cgi-bin,common,html}/`	Subdirectories for the "password" panel, which is accessed by the */user-password/* URL.
`/etc/e-smith/locale/`	Top-level directory for all panel localizations.
`./en-us/etc/e-smith/web/functions/`	Subdirectory containing localization into US English.
`./fr/etc/e-smith/web/functions/`	Subdirectory containing localization into French.
`/usr/lib/perl5/site_perl/`	Top-level directory for all Perl modules.
`./esmith/FormMagick/Panel/`	Subdirectory containing Perl modules which provide the implementations to support the panel definitions.

Web function scripts

The `functions` subdirectory contains all of the screen definitions for all panels. Each screen definition is a CGI script which displays the screen and also handles the CGI form submission. The scripts are written using the `CGI::FormMagick` toolkit, which separates the screen layout from the panel implementation code, facilitates form validation and provides full support for localisation of the manager.

An overview of FormMagick

Layout of a FormMagick script

This section describes the FormMagick panel which is used in the Section called *Exercise 5: Adding a user interface screen* in Chapter 12. A typical FormMagick web function starts with the script preamble, which notes it as a perl script and informs the vi editor that the majority of the file is XML, rather than perl.

```
#!/usr/bin/perl -wT
# vim: ft=xml:
```

This is followed by the navigation settings metadata, which determine where the script should appear in the manager menu bar.

```
#----------------------------------------------------------------------
# heading     : Demo
# description : Logger
# navigation  : 1000 1000
#----------------------------------------------------------------------
```

Next is a small number of lines of perl which create a FormMagick object and then call the `display` method to draw the page.

```
use strict;
use warnings;

use esmith::FormMagick::Panel::loggerdemo;

my $f = esmith::FormMagick::Panel::loggerdemo->new();
$f->display();
```

And finally there is the FormMagick XML page description, which starts at the `__DATA__` marker and continues to the end of file. We will examine that in the next section.

The FormMagick XML description

The FormMagick XML is divided into a preamble and then a set of pages. The preamble contains references to the title, header and footer of the page. These are usually the same on all pages so that a consistent header and footer is displayed.

```
<form
    title="FORM_TITLE"
    header="/etc/e-smith/web/common/head.tmpl"
    footer="/etc/e-smith/web/common/foot.tmpl">
```

The upper-case word `FORM_TITLE` is a placemarker token for a phrase which needs to be localised. There is an associated lexicon file which provides the translation of this token into the appropriate language for the user accessing the panel, as specified by their browser settings. For example, here is the English lexicon entry for that token:

```
<entry>
    <base>FORM_TITLE</base>
    <trans>Logger demo</trans>
</entry>
```

If the user browses the panel with English as their chosen language, the panel will display in English. If they choose French, French will be displayed. If an unsupported language is chosen, FormMagick will fall back to US English. Adding another language is basically a matter of providing the lexicon for that language.

The rest of the XML description is a series of pages. In this example there is a single page. Each page starts with a `page` tag, which gives the page a name for later reference and can optionally specify a `pre-event` and `post-event`.

```
<page name="First" pre-event="print_status_message()"
    post-event="change_settings">
```

The `pre-event` is a reference to a function in the panel implementation (described later) and called before the page is loaded. The `post-event` is called after the user submits the information on the page, for example by pressing the `Save` button.

Each page is then composed of a number of fields

```
    <field
        type="select"
        id="loggerdemo_Interval"
        options="10,20,30,40,50"
        value="get_interval()">
        <label>LABEL_LOGGERDEMO_INTERVAL</label>
    </field>

    <field
        type="select"
        id="loggerdemo_status"
        options="'disabled' => 'DISABLED', 'enabled' => 'ENABLED'"
        value="get_status()">
        <label>LABEL_LOGGERDEMO_STATUS</label>
    </field>
```

Each field describes a user interface widget (e.g. a select box) and provides the data required for that widget. These data may be static lists (the options of the first field above), a set of key/value pairs (the options of the second field above) or dyanamic data returned from a subroutine (the value parameters in each of the fields).

The command **perldoc CGI::FormMagick** provides detailed documentation about the supported field types.

It is also possible to call subroutines which generate the required HTML for a section of a page. For example, buttons are often added by calling the `print_button` routine:

```
        <subroutine src="print_button('SAVE')" />
```

Note: Buttons should be part of the FormMagick XML description, and hopefully will be in the future. The `print_button` routine is a workaround for the lack of a button widget.

Each page must finish with a closing page tag:

```
    </page>
```

After all of the pages have been described there is a single XML tag to close the form.

```
</form>
```

Navigation metadata

The web manager's navigation frame is generated automatically by examining the contents of the `/etc/e-smith/web/functions/` directory.

In order to be listed in the navigation frame, your CGI script must contain `heading`, `description` and `navigation` lines, usually at the top of the script:

```
# heading     : Configuration
# description : E-mail
# navigation  : 6000 6700
```

These define the category heading under which your add-on's admin interface should be listed, the title it should have, and the priority it should have in the listing order. The first number gives the priority of the heading (usually a multiple of 1000) and the second number gives the priority of this particular item within that heading group. In other words, a heading with a priority of 1000 will come before one with 6000 in the navigation panel, and within that heading category the individual items are listed in order from highest to lowest.

To figure out what numbers to give your own script, figure out where you want it to appear in the navigation panel then check source code for the scripts which appear before and after where you want to be. For instance, if you want your item to appear before "Remote Access" and after "Local Networks" in the navigation menu, you would look at `/etc/e-smith/web/functions/remoteaccess` and `/etc/e-smith/web/functions/localnetworks` and find the following:

```
# heading     : Security
# description : Remote access
# navigation  : 5000 5200

# heading     : Security
# description : Local networks
# navigation  : 5000 5300
```

You might then put something these lines in your own script:

```
# heading     : Security
# description : Advanced security
# navigation  : 5000 5250
```

Tip: When naming your script, use a name which closely resembles the description (and hence the name in the navigation panel). This makes it easier to correlate menu items to Perl scripts. Just take the descriptive name and remove capital letters, punctuation and spaces. For instance, "Advanced security" might become `/etc/e-smith/web/functions/advancedsecurity`

Permissions and security

The CGI scripts must have elevated permissions (setuid `root`) in order to write to the configuration database, since they will be run by the web server (which runs as user `www`). To ensure that these scripts can only be run by system administrators, the permissions on the parent directory and the scripts are set so that only the members of the `admin` group can run them. These panels are also restricted in the web server configuration so that only the `admin` user can access them.

Common files

The `common` subdirectory contains any static files (such as images) which are used by multiple panels.

Panel definitions

The `panels` directory contains the panel definitions. There is one subdirectory for each panel. Each panel must have html and cgi-bin subdirectories. The cgi-bin subdirectory should contain only symbolic links to the actual CGI scripts in the functions directory, and the html directory should contain the main index.html file for the panel, as well as any required navigation links.

Note: The word panel is also sometimes used to refer to an individual web manager web function.

Keeping the CGI scripts for all panels in a shared directory makes it much easier to create auxiliary panels with slightly different options and permissions. You can just copy the entire panel directory, then customize the access permissions and navigation links. For example, it would be very straightforward to create a password-protected panel which only allowed the creation and deletion of user accounts. That task could be delegated to administrative staff.

III. How to create an SME Server package - step by step

Chapter 11. Getting started

The best way to get started is to install an SME Server and start experimenting with it. Download a copy from http://mirror.contribs.org/pub/smeserver/releases/7/iso/ and burn your own CD.

If you (or any developers at your organization) have multiple computers on a home network, a cablemodem, DSL, or dialup connection, and an old Pentium machine that you don't need, we recommend installing the SME Server software on the old Pentium machine, and using it as a home gateway and firewall.

> **Note:** The SME Server software erases all data from the PC on which it is installed, to turn it into a dedicated server that can run 24x7. Do not install it on a PC unless you are prepared to erase all of its data!

Alternatively, you can install the SME Server on a corporate LAN in *server/gateway* mode (creating a small private network behind a firewall that occupies a single IP address on the Internet) or in *server-only* mode - in which the SME Server provides network services to other computers as a peer on the network.

> **Note:** Warning! The server-only mode is designed for LAN environments that already have a firewall/gateway.

In addition to feeling comfortable installing and using the SME Server software, you should also have a working knowledge of Linux, including use of the command line tools.

You should also be familiar with the perl programming language. Most of the SME Server software is written in perl, and the configuration template mechanism is based on perl.

It is strongly recommended that you obtain and read a copy of the book Maximum RPM (ISBN 067231 1054) or study the on-line version available at http://www.rpm.org/max-rpm/.

You also need to know how to use one of the Linux text editors such as `vi`, `nano` or `pico`. It is also possible to edit files on a remote machine and copy them to the server. However, *it is important that the files are converted to Unix text format*.

Creating a development environment

Packages which do not require compilation, for example shell and perl scripts, can be built on the SME Server platform. All of the examples in this documentation can be performed on a standard SME Server installation.

Before attempting to compile any software, you should check whether the package is available from one of the many well-maintained RPM repositories. Using these RPMs will ensure compatibility with the other RPMs on the SME Server. You are likely to find the package you want in either the CentOS (http://mirror.centos.org/centos/4) or Dag Wieers (http://apt.sw.be/redhat/el4/en) repositories.

If an RPM does not already exist, you should install a CentOS developer workstation or server for SME Server development. You will also need to install the `e-smith-devtools` packages which can be found on the SME Server CD.

> **Note:** We strongly recommend against installing development tools, such as compilers, on any production servers, especially those which are accessible from the Internet.

Chapter 12. Getting to know how to customize the SME Server

Once you have studied the architecture of the SME Server, it is best to try to make some small customizations to become comfortable with the concepts. The number one rule to remember is: customizations always involve *adding* files to the server, rather than modifying existing files. This is very important, as it enables customizations to be easily packaged, and mixed and matched. The unique architecture of the SME Server enables virtually anything to be customized by adding a file in the correct location.

Exercise 1: Changing a configuration template

Let us say that you wish to customize your server so that it runs a specified program every twenty minutes. To simplify the problem, let us assume that this program simply adds a line of dots to the log file (`/var/log/messages`), i.e.:

```
/usr/bin/logger -t "Demo" "......"
```

Normally you would accomplish this by adding a line to the `/etc/crontab` file, which is the standard Linux mechanism for running scheduled jobs. However, the default `/etc/crontab` file looks something like this on an SME Server:

```
#------------------------------------------------------------
#              !!DO NOT MODIFY THIS FILE!!
#
# Manual changes will be lost when this file is regenerated.
#
# Please read the developer's guide, which is available
# at http://www.contribs.org/development/
#
# Copyright (C) 1999-2006 Mitel Networks Corporation
#------------------------------------------------------------

SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# run-parts

01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly

# logrotate
12 1 */7 * * root       /sbin/e-smith/signal-event logrotate
```

Note the auto-generated comment block which reminds you not to edit the file. If you do, *your changes will be overwritten* when the template is next expanded by a system event. We want to append a new line that looks like this (read the Linux crontab documentation to understand the format of crontab entries):

```
*/20  *  *  *  *   root   /usr/bin/logger -t "Demo" "......"
```

Remember that we cannot simply edit the `/etc/crontab` file. The rule is that we must perform this customization by adding a new file to the system. To get an idea how to do this, have a look at the contents of the template for `/etc/crontab`:

```
[gordonr@smebuild crontab]$ pwd
/etc/e-smith/templates/etc/crontab

[gordonr@smebuild crontab]$ ls
00setup  10runparts  20statusreport  65_logrotate  email
```

Each of the files in that directory is a *template fragment*. The SME Server builds the `/etc/crontab` file by assembling those fragments and running them through the template processor.

To make your customization, create your own additional fragment by creating a file in this directory called `25templatedemo` with the following contents:

```
# Template demo crontab entry:
*/20  *  *  *  *   root   /usr/bin/logger -t "Demo" "......"
```

Next time the SME Server regenerates the `/etc/crontab` file, it will contain your additional fragment. Starting the name with the prefix "25" forces the template fragment to go between the "20statusreport" and "65_logrotate" fragments. Force the `/etc/crontab` file to be generated immediately by typing the command:

```
expand-template /etc/crontab
```

If you look at the `/etc/crontab` file now, you should see your new fragment at the appropriate place, and your customization will take effect immediately (as cron notices when its configuration file has been changed). Check `/var/log/messages` to see the results.

To package this customization, you will need to create an RPM package that contains this single file, and ensures that the `/etc/crontab` template is expanded in the relevant events. You should also call one of these events in the RPM post-install section to ensure that the template is expanded without further action. Installing that RPM on any SME Server will cause the customization to occur, and will start printing the line of dots to the `/var/log/messages` logfile every 20 minutes.

The final point to note here is that if you *remove* your new file `25templatedemo` and re-expand the `/etc/crontab` template, the crontab will go back to the way it was, and your customization will disappear cleanly. Therefore you should put a post-uninstallation script into your RPM package that runs the appropriate events to expand the templates once more. That will result in a package that installs and uninstalls cleanly.

Remember that for testing you can call `expand-template` directly in the post-install and post-uninstall sections, but released software should use the `templates2expand` mechanism to request template expansion in the relevant events.

Exercise 2: The magic of templates

For the next exercise, let us build on the first one. You have already created an exciting (o.k. not that exciting) new capability - the ability of the server to write dots into the log file. Let us now take advantage of the fact that the template processor can fill in values from the configuration database.

Edit the `/etc/e-smith/templates/etc/crontab/25templatedemo` file again, this time with the following contents:

```
# Template demo crontab entry:
*/20 * * * * root /usr/bin/logger -t "Demo3" "... {
  use esmith::AccountsDB;

  $adb = esmith::AccountsDB->open_ro;
  $adb->get_prop('admin', 'ForwardAddress') || 'admin';
}
```

Once again, regenerate the template by typing:

```
expand-template /etc/crontab
```

If you look at the new /etc/crontab file, you will see that the template processor has replaced the block between the braces with the actual email address of the administrator, which is defined in the accounts database.

> **Note:** Be careful with the placement of the braces in the example. We want two lines of output - the comment and the line starting with the asterisk. If you move the opening brace onto a new line, you will end up with three lines of output.
>
> Whitespace is not signifcant and the code within braces should be formatted "nicely". However, *whitespace outside braces is significant* and will be copied literally to the output file.

You could check that value with the **db accounts show admin** command. With this change to the template, the message which will go to the `/var/log/messages` log file every 20 minutes will contain the actual current administrative email address, rather than a hardcoded text message.

Now here is the exciting part. Click on the *E-mail* function in the server manager user interface, and change the administrative email address. If you look at `/etc/crontab`, you will see that it has been updated with the new email address! Every 20 minutes you will receive a new entry in the messages log which automatically reflects the new setting of the administrator email address.

So by simply creating a single file `25templatedemo`, you have created quite a sophisticated customization that changes behaviour based on the system settings. And you have done so without affecting the rest of the system or requiring additional changes to the user interface.

The reason this works is subtle, but follows from the overall architecture. It is critical to study this example until you understand it thoroughly. If you understand exactly how this example works, you understand a substantial part of the SME Server architecture.

To understand this example in detail, let's start from the top by studying the user interface. The implementation section of the *E-mail* function can be found in the file `/usr/lib/perl5/site_perl/esmith/FormMagick/Panel/emailsettings.pm`. If you study the part of this script that gets executed when the user clicks *Save*, you will find some Perl code that saves the administrator e-mail address into the configuration database and signals the `email-update` event - thus informing the system that the email settings have changed:

```
sub change_settings_delivery
{
    my ($fm) = @_;
    my $q = $fm->{'cgi'};

    [...]

    my $admin_email = $q->param('AdminEmail');
    my $admin = $accounts->get('admin');
    if ($admin_email)
    {
        $admin->merge_props(
                EmailForward => 'forward',
                ForwardAddress => $admin_email,
                );
    }
    else
    {
        $admin->merge_props(
                EmailForward => 'local',
                ForwardAddress => '',
                );
    }

    [...]

    unless ( system( "/sbin/e-smith/signal-event", "email-update" ) == 0 )
    {
        $fm->error('ERROR_UPDATING');
        return undef;
    }
    $fm->success('SUCCESS');
}
```

When the `email-update` event is signalled, the SME Server executes all of the action scripts in the `/etc/e-smith/events/email-update/` directory. The event also expands all templates as noted in the `templates2expand/` hierarchy, including the `/etc/crontab` template.

That may all seem rather complicated, but it boils down to this: changing the administrator email address automatically rebuilds the `/etc/crontab` template - and if you have customized that template, your customization will automatically pick up the current administrator email address.

Note that if you wanted additional programs to execute whenever the email settings were changed, you would put all of those programs in the `/etc/e-smith/events/actions/` directory, then create symbolic links to them from the `/etc/e-smith/events/email-update/` directory. We use symbolic links because you may want your program triggered by other events as well as `email-update`, and so you create links from all of the relevant event directories back to your action program.

This system is, by design, extensible. For example, you could use this exact same type of customization to send an email message every hour containing the current IP address. This is left as an exercise to the reader.

Exercise 3: Using events and actions

In the SME Server, *events* are like *callbacks* in a programming language. The system signals an event whenever something interesting happens (e.g. a user is added, the IP address changes, etc.), which automatically executes all programs in the event directory. Therefore, any applications which need to know when a certain event is happening simply create a symbolic link from the event directory to a *handler* program, which will get executed whenever the event occurs.

In the previous exercise, we relied on the predefined events and actions in the SME Server to keep the `/etc/crontab` file up to date. In this example, we will create a new action script that will track the user accounts in the system. This script can be used as a template for any type of application that has its own notion of user accounts and needs to be driven by the existing user interface for adding, deleting, or modifying users.

Start by creating a new file called `/etc/e-smith/events/actions/demo-user-tracking` with the following contents:

```
#!/usr/bin/perl -w

# Set up Perl environment and libraries

package esmith;
use strict;
use warnings;
use esmith::ConfigDB;
use esmith::AccountsDB;

# Prepare to access configuration databases
my $db = esmith::ConfigDB->open_ro
    or die "Couldn’t open ConfigDB\n";

my $accounts = esmith::AccountsDB->open_ro
    or die "Couldn’t open AccountsDB\n";

# Read domain name from configuration database
my $domain = $db->get_value(’DomainName’);

# Read command line arguments
my $event = $ARGV [0];
my $id    = $ARGV [1];

# If no command line arguments, assume this is the initial setup
 # of all users. Process all accounts of type "user" (ignore groups,
# information bays, printers, etc.)

unless ($event and $id)
{
    for my $user ($accounts->users)
    {
        my $key     = $user->key;
        my $first   = $user->prop('FirstName');
        my $last    = $user->prop('LastName');

        system ("/usr/bin/logger", "-t", "Demo3",
                "Initializing user $key ($first $last) in domain $domain");
    }

    exit 0;
}

# If command line arguments are present, then this is a create, modify,
# or delete event, signalled by the SME Server event/action system.

my $user  = $accounts->get($id)
    or die "User $id does not exist\n";

my $first = $user->prop('FirstName');
my $last  = $user->prop('LastName');

if ($event eq 'user-create')
{
    system ("/usr/bin/logger", "-t", "Demo3",
            "Creating user $id ($first $last) in domain $domain");
}
elsif ($event eq 'user-modify')
{
    system ("/usr/bin/logger", "-t", "Demo3",
            "Changing user $id to ($first $last) in domain $domain");
}
elsif ($event eq 'user-delete')
{
    system ("/usr/bin/logger", "-t", "Demo3",
            "Deleting user $id in domain $domain");
}
else
{
    system ("/usr/bin/logger", "-t", "Demo3", "Ignoring $event event");
}

exit 0;
```

Make sure the permissions are correct:

```
chmod +x /etc/e-smith/events/actions/demo-user-tracking
```

Now create symbolic links so that this program is executed whenever a user is created, modified, or deleted. Make three symbolic links; one for each event directory:

```
cd /etc/e-smith/events
ln -s ../actions/demo-user-tracking user-create/S90demo-user-tracking
ln -s ../actions/demo-user-tracking user-modify/S90demo-user-tracking
ln -s ../actions/demo-user-tracking user-delete/S90demo-user-tracking
```

The `S90` prefix ensures that the program will be executed after the standard actions (which typically have prefixes ranging from `S15` to `S80`).

Study this program carefully. It uses many different SME Server capabilities. If invoked from the command line with no arguments, it will read all user accounts from the user database, fetch all the data fields associated with each user, and print this information to the log file. If invoked as an event, the SME Server will automatically pass it the event name and user id as command line arguments; in this case the program will print messages to the log file explaining that it is adding, modifying, or dropping the user.

Trying watching the log file by running the command:

```
tail -F /var/log/messages
```

or with the "View log files" panel in the server-manager. Use the standard SME Server user interface to add, modify, or remove users. You should see a stream of comments in the log file.

If you were creating an application that had its own database of user information, you would replace the logfile-writing code with your own code that initialized new users, modified them, etc. Then you would arrange for the RPM post-install script to execute:

```
/etc/e-smith/events/actions/demo-user-tracking
```

with no command line arguments. When the application is initially installed, it will immediately read all users from the database and set them up in your application. Then, as users are added, modified, or dropped over time - your code will be invoked each time, to update the application's private user database.

Tip: It is almost always better to extend the existing accounts database with additional properties for your application than to maintain another database.

Exercise 4: Adding new configuration database parameters

New system configuration parameters can be spontaneously invented and added to the configuration database at any time. For example, let us return to our earlier exercise and parameterize the time interval for the log messages by introducing a new parameter called `LogInterval`.

You can write that parameter into the configuration database, as though it had always existed. For example, type this on the command line:

```
config set LogInterval 30
```

You can use `config show LogInterval` to show that it was set as intended. You can now edit the `25templatedemo` file and replace the hardcoded number 20 with the `LogInterval` parameter. The resulting file will read:

```
# Template demo crontab entry:
*/{ $LogInterval } * * * * root /usr/bin/logger -t "Demo4" "... {
  use esmith::AccountsDB;

  $adb = esmith::AccountsDB->open_ro;
  $adb->get_prop('admin', 'ForwardAddress');

} ..."
```

Now you can change the logger interval at any time by typing the following (replace the number 20 with whatever logger interval you want):

```
config set LogInterval 20
expand-template /etc/crontab
```

This ability to spontaneously introduce new configuration parameters is very important in the SME Server architecture. The configuration database is a high-level specification of how the overall system is supposed to behave for the end user. Configuration settings are like knobs on a stereo system. The templates, events, and actions, are the underlying machinery to carry out the user's wishes. When adding a new application to the system, it is important to be able to add new knobs on demand.

Now let us say that you want to introduce a parameter to enable or disable this logging function. At this point, you might start thinking of this logging activity as a *service* that you should be able to enable or disable. In this case the convention is to create a single *service* entry in the configuration database to manage both parameters.

To implement this, first delete the `LogInterval` parameter, which will no longer be needed:

```
db configuration delete LogInterval
```

Now create a service entry:

```
db configuration set loggerdemo service status enabled Interval 20
```

If you examine the configuration database you will see the new entry looks like this:

```
[root@gsxdev1 ~]# config show loggerdemo
loggerdemo=service
    Interval=20
    status=enabled
```

Note: By convention, database keys are lower case, and property names are mixed case (e.g. Interval). The status property is an exception here, and is stored lower case.

Now edit the `25templatedemo` file to look like this:

```
# Template demo crontab entry
{
    my $status = $loggerdemo{status} || "disabled";

    return "# loggerdemo service is disabled."
        unless ($status eq "enabled");
    use esmith::AccountsDB;

    $adb = esmith::AccountsDB->open_ro;
    my $admin_email = $adb->get_prop('admin', 'ForwardAddress')
                        || 'admin';

    my $interval = $loggerdemo{Interval} || 10;

    $OUT = "*/$interval * * * * root /usr/bin/logger";
    $OUT .= " -t \"Demo4\" \"... $admin_email ...\"\n";
}
```

> **Note:** The variable `$OUT` is a special variable used for output from a template. It is documented in the `Text::Template` documentation. For now, just think about it as the return value from the template, so set it to the value you want to print from this fragment. Note also that a template fragment can return static text without setting `$OUT` directly, as shown on the `return` line above.

This is more complicated than the original template, but it is also more flexible. Note that the initial comment (`# Template demo crontab entry`) is hardcoded, but the line that follows is generated from the configuration database parameters. The code in the template retrieves the *loggerdemo* service entry, retrieves the required properties, and returns the appropriate output. To experiment, try different combinations of parameters:

```
db configuration setprop loggerdemo Interval 10
expand-template /etc/crontab
```

and

```
config setprop loggerdemo status disabled
expand-template /etc/crontab
```

and so on.

Exercise 5: Adding a user interface screen

Let us add a nice user interface screen to adjust the logger interval. Create a new file called `/etc/e-smith/web/functions/loggerdemo`, with the following contents:

```
#!/usr/bin/perl -wT
# vim: ft=xml:

#----------------------------------------------------------------------
# heading     : Demo
# description : Logger
# navigation  : 1000 1000
#----------------------------------------------------------------------

use strict;
use warnings;

use esmith::FormMagick::Panel::loggerdemo;

my $f = esmith::FormMagick::Panel::loggerdemo->new();
$f->display();

__DATA__
<form
    title="FORM_TITLE"
    header="/etc/e-smith/web/common/head.tmpl"
    footer="/etc/e-smith/web/common/foot.tmpl">

    <page name="First" pre-event="print_status_message()"
        post-event="change_settings">

        <field
            type="select"
            id="loggerdemo_Interval"
            options="10,20,30,40,50"
            value="get_interval()">
            <label>LABEL_LOGGERDEMO_INTERVAL</label>
        </field>

        <field
            type="select"
            id="loggerdemo_status"
            options="'disabled' => 'DISABLED', 'enabled' => 'ENABLED'"
            value="get_status()">
            <label>LABEL_LOGGERDEMO_STATUS</label>
        </field>


        <subroutine src="print_button('SAVE')" />
    </page>
</form>
```

The file above describes the panel layout, which is written in `FormMagick` XML. Further details about FormMagick can be fiound in the Section called *An overview of FormMagick* in Chapter 10.

Another file provides the implementation, which goes into `/usr/lib/perl5/site_perl/esmith/FormMagick/Panel/loggerdemo.pm`:

```
#!/usr/bin/perl -w
package esmith::FormMagick::Panel::loggerdemo;

use strict;
use warnings;
use esmith::ConfigDB;
use esmith::FormMagick;

our @ISA = qw(esmith::FormMagick Exporter);
our @EXPORT = qw();

our $VERSION = sprintf '%d.%03d', q$Revision: 1.1 $ =~ /: (\d+).(\d+)/;

our $db = esmith::ConfigDB->open or die "Couldn't open ConfigDB\n";

sub get_status
{
    return $db->get_prop("loggerdemo", "status");
}

sub get_interval
{
    return $db->get_prop("loggerdemo", "Interval");
}

sub change_settings
{
    my $fm = shift;
    my $q = $fm->{'cgi'};

    $db->set_prop('loggerdemo', 'status', $q->param("loggerdemo_status"));
    $db->set_prop('loggerdemo', 'Interval', $q->param("loggerdemo_Interval"));

    unless ( system ("/sbin/e-smith/expand-template", "/etc/crontab") == 0 )
    {
        $fm->error('ERROR_UPDATING');
        return undef;
    }

    $fm->success('SUCCESS');
}

1;
```

> **Note:** This code example calls expand-template. This is used for illustrative purposes only. All templates should be expanded by signalling events.

Similarly to events and actions, the `/etc/e-smith/web/functions/` directory is a repository of potentially available functions. To make the new function actually show up in the user interface, create a symbolic link to it from the web manager cgi-bin directory, as follows:

```
    cd /etc/e-smith/web/panels/manager/cgi-bin
    ln -s ../../../functions/loggerdemo loggerdemo
```

Now, make sure the permissions and ownership are correct so that the web server can run your new function:

```
cd /etc/e-smith/web/functions
chown root:admin loggerdemo
chmod 4750 loggerdemo
```

We also need to run a program to rebuild the navigation bar. This is only required when adding or removing functions from the manager, and is normally handled automatically. Let's do it manually for now:

```
/etc/e-smith/events/actions/navigation-conf
```

Try the server manager now, reloading the navigation bar in your browser, if necessary. You will see a new category called *Demo* and a new function within it called *Logger*. It will look a bit bare with some upper-case words which signify phrases which have not been localised. We'll fix that in a moment.

Try experimenting with it - it is a nice little user interface for playing with the logger customization. Every time you change the settings, notice that the `/etc/crontab` file is updated appropriately.

Adding localizations

The SME Server is designed to support localization into any language. This is done by small files which describe the mapping from the upper case tags (seen above) to the appropriate words for the local language. Enter the following into `/etc/e-smith/locale/en-us/etc/e-smith/web/functions/loggerdemo`

```
<lexicon lang="en-us">
    <!-- vim: ft=xml:
    -->
    <entry>
        <base>FORM_TITLE</base>
        <trans>Logger demo</trans>
    </entry>

    <entry>
            <base>Demo</base>
            <trans>Demo</trans>
    </entry>

    <entry>
            <base>Logger</base>
            <trans>Logger</trans>
    </entry>

    <entry>
            <base>LABEL_LOGGERDEMO_STATUS</base>
            <trans>Status</trans>
    </entry>

    <entry>
            <base>ENABLED</base>
            <trans>Enabled</trans>
    </entry>

    <entry>
            <base>DISABLED</base>
            <trans>Disabled</trans>
    </entry>
    <entry>
            <base>SAVE</base>
            <trans>Save</trans>
    </entry>

</lexicon>
```

Now, re-select the *Logger* function and the tags should now be replaced by English phrases. We can very easily add translations for other languages by adding new locale files in the same hierarchy.

> **Note:** The `LABEL_LOGGERDEMO_INTERVAL` tags has intentionally been left untranslated. Why don't you fix it now?

Exercise 6: Adding a new event type

Let us continue building on this example. Let us say that you want to add a hook to the logger demo, enabling other third party applications to receive a notification whenever the logger settings are changed. We need a new event type for this. Let us create a new event called `loggerdemo-update`:

```
mkdir -p /etc/e-smith/events/loggerdemo-update
```

The idea is that we will arrange for our user interface function to *signal* this new event whenever the settings are changed, instead of directly expanding the `/etc/crontab` file. Edit the `/usr/lib/perl5/site_perl/esmith/FormMagick/Panels/loggerdemo.pm` file and replace the line:

```
unless (system ("/sbin/e-smith/expand-template", "/etc/crontab") == 0)
```

with the line:

```
unless (system ("/sbin/e-smith/signal-event", "loggerdemo-update") == 0)
```

Now the loggerdemo user interface signals the new event whenever it saves the new settings to the configuration database. Next, we have to make sure that the event does what we need it to do - rebuild the `/etc/crontab` file.

```
cd /etc/e-smith/events/loggerdemo-update
mkdir -p templates2expand/etc
touch templates2expand/etc/crontab
```

Now the example should work just as before. You can edit the loggerdemo settings in the web manager, and see that the /etc/crontab file changes. But now other applications can also receive notifications of the loggerdemo-update event, by creating symbolic links from the `/etc/e-smith/events/loggerdemo-update` directory.

Note: Panel implementation code should *always* signal events, and should *never* expand templates or modify files directly. These modifications should only be peformed in events.

Exercise 7: Thought experiment - adding a new server application

You have now learned most of the machinery required for integrating a new server application into the SME Server. Consider a hypothetical chat server, with a configuration file called `/etc/chatserv.conf`.

You would first decide if the chat server needs any new settings. If so, you can create a new service entry in the configuration database to hold those settings.

You would then create a templated version of `/etc/chatserv.conf`, by creating the directory `/etc/e-smith/templates/etc/chatserv.conf` with at least one template fragment. The template should generate the correct version of `/etc/chatserv.conf` based on the configuration database settings. Experiment with different combinations of settings and manually running `expand-template /etc/chatserv.conf`.

Then create a web manager function (if one is required) enabling users to edit the settings. An event, for example `chatserv-update` should be signalled whenever the settings are changed. You would normally link into an existing event, but this is a thought experiment.

Then create the file `/etc/e-smith/events/chatserv-update/templates2expand/etc/chatserv.conf` to ensure that the configuration file is updated. You may also need to restart the chatserv program after changing its configuration, so you'll need a `/etc/e-smith/events/chatserver-update/services2adjust/chatserv` link which contains the word `restart`.

Finally, figure out when you'll need these scripts to be run. At a minimum, you'll want to run them whenever the event is signalled indicating that the user has saved new settings from the web manager. A symbolic link should be created, so that the new event triggers your new action scripts. Let's say you also need to reconfigure and restart your server each time a user is added. In that case, you would also create symlinks from the `user-create` event to your action scripts.

We are done - the new server has been integrated into the system. Note that every single one of these steps involved creating new files and directories, and reading/writing from the configuration database. *No existing files were edited!* You can easily imagine packaging these new files and directories into an RPM.

Customization guidelines

When creating applications:

- You can create new configuration parameters.

- You can add new configuration templates under `/etc/e-smith/templates/`.
- You can add fragments to any of the existing templates under `/etc/e-smith/templates/`.
- You can arrange for actions to be triggered upon package pre-install, post-install, pre-uninstall, or post-uninstall.
- You can link new actions into the events listed in the Section called *Standard events and their arguments* in Chapter 7.
- You can create new events to allow your feature set to be expanded. Do *not* create events for any other reason. They are not a replacement for function calls.
- You can arrange for new server programs to be started up at boot time.
- Typically you would expand a template at application post-install time, application post-uninstall time, and one or more of the other events.
- You can add new web functions to the navigation bar. Remember, panels should only retrieve and modify database values, perform validation, and signal events.

That is all! Applications should not make any extensions to the system other than these. For example, an application should not:

- Change the kernel or add new kernel modules.
- Edit configuration files directly - templates must be used.
- Link actions into any events other than the ones listed above or new events that you create. All built-in events other than the ones listed above are subject to change without notice in new SME Server releases.
- Directly access the per-user email store (i.e. the Maildir and related subdirectories within each user's home directory). This access should be performed via the IMAP server as the location and format of a user's home directory may change between releases.
- Take over the function of existing servers (i.e. shut down qmail and Apache and take over ports 25 and 80). The SME Server has features for proxying email and web requests to other servers on the system.

Do not expand templates at boot time. The only thing that should be happening during a normal system startup is to start servers. Templates should be expanded when it is necessary to change the system configuration (i.e. when a setting is changed, when the IP address changes, etc.) A normal shutdown or reboot should not trigger configuration changes. The bootstrap-console-save event will be run after a system reconfiguration, but will not run if the system does not require reconfiguration.

Chapter 13. Packaging your application

Once you have created a customization for your SME Server by adding new files, directories, and symbolic links (for your actions, events, etc.) - and perhaps also triggering an action to initialize your customization - you are ready to package your customization into an RPM.

A quick introduction to RPMs

All SME Server software packages are distributed as RPM packages. This is the format used by CentOS and other major Linux distributions for distributing applications and other collections of files. The RPM system provides the ability to install, upgrade, remove and (importantly) verify the contents of installed packages.

An RPM essentially consists of an archive of all the files required by a piece of software. Additionally, it includes meta-information describing the software, and scripts which must be run to install or uninstall the software.

Meta-information stored in an RPM includes:

1. summary and description of the software
2. package name
3. version number
4. copyright information
5. category/group to which the software belongs
6. name and email address of the packager
7. pre-requisites to installing this package
8. ... and more

Selecting and creating RPMs for your application

Your application will typically depend on several components:

1. Software packages that are shipped as a standard part of the SME Server. You do not need to include any of these packages; they are always present in the runtime environment.
2. Software packages that are *not* a standard part of the SME Server, but that are required by your software, and would also be of general use in the runtime environment. For example: a Java runtime environment, libraries that enable communication with devices, etc. If possible, these packages should be made into separate packages, rather than being included in your application. This makes it easier to share them with other applications and enforces version compatiblity.
3. Software packages that are *not* a standard part of the SME Server, and that are specific to your software application (i.e. not generally useful in the runtime environment). This is the raw Linux version of your application without any specific SME Server integration code. For example, if your

application is already available for Linux in the form of RPMs - these RPMs are what we are referring to. These are referred to as the *application RPMs*.

4. Any new files that you have created specifically in order to integrate your application into the SME Server runtime environment - should be packaged into a single RPM, as explained in the next section. This is referred to as the *integration RPM*.

So, if your application is based on Linux software that has already been packaged into RPMs, then you will need to create one new RPM:the *integration* RPM.

If, on the other hand, your application is based on Linux software that has *not* yet been packaged into RPMs, then you will probably need to create at least two RPMs: one or more *application* RPMs, and the *integration* RPM.

Finally, for simple customizations (such as the *loggerdemo* example earlier in this manual) there may be no application RPM at all. This would be typical if the point of the application is to change the server configuration without really adding a new software package. In this case you need only the *integration* RPM which contains the new template fragments, user interface screens, etc..

All files on the system, except for user data, must be installed by RPMs.

Setting up your RPM development environment

If you haven't done so already, set up an RPM development environment. If you are using an SME Server as your development environment, you will need to alter your user account to enable regular login. If you want to enable account "joe", then you would type the following commands from the root account:

```
chsh -s /bin/bash joe

db accounts setprop joe Shell /bin/bash
```

Note: Shell/login access is disabled by default to enhance the security of the SME Server. Shell access should only be provided to users who require it and who can be trusted to maintain system security.

Then you should be able to log in to the server as user "joe", and get a Linux command line prompt. Log in, then type the following commands to set up your RPM work area:

```
cd ~/
mkdir -p rpms/{SRPMS,BUILD,SOURCES,SPECS,RPMS,lib}
mkdir -p rpms/RPMS/{i386,noarch}
echo "%_topdir $HOME/rpms" > ~/.rpmmacros
```

You will now find that you have a directory called `rpms` in which you will do your work. Under this are the following subdirectories:

`SOURCES`

The base material from which RPMs are built -- source code, tarballs, etc.

`BUILD`

Working area used by the **rpmbuild** program during RPM creation

`SPECS`

Specification files for building RPMs

`SRPMS`

Source RPMS (created by build process)

`RPMS`

Binary RPMS (created by build process). Has subdirectories `noarch` and `i386` for architecture independent and x86 platforms respectively.

As you prepare software to turn into RPMs, you will place files in these directories as appropriate. The following sections will describe what goes where as each item is covered.

Tip: As you start work on an RPM for version x.y.z of a package, create a subdirectory `rpms/SOURCES/yourpackage-x.y.z/` to work in.

```
mkdir rpms/SOURCES/yourpackage-x.y.z
```

Under this directory there should be a subdirectory called `root`, under which is an image of the file hierarchy that will be installed by the RPM.

```
mkdir rpms/SOURCES/yourpackage-x.y.z/root
```

Building an RPM

This section describes the process for building an RPM - step by step.

1. Choose a name and version number for your package. We are going to package the complete loggerdemo example and will use `loggerdemo` and `1.0.0` as the name and version number.
2. Collect all of the files which have been created in the previous sections into the `/tmp/` directory. There is one additional file `createlinks` which looks like this:

```
#!/usr/bin/perl -w

use esmith::Build::CreateLinks qw(:all);
use File::Basename;

my $panel = "manager";

panel_link("loggerdemo", $panel);
```

3. Create the directory hierarchy required for building the RPM. This is very close to the hierarchy on the installed system.

   ```
   # Change to the SOURCES directory
   cd ~/rpms/SOURCES

   # Remove old files (check that you don't need anything here!)
   rm -rf loggerdemo-1.0.0

   # Create new directory
   mkdir loggerdemo-1.0.0
   cd loggerdemo-1.0.0

   # The crontab template fragment
   mkdir -p root/etc/e-smith/templates/etc/crontab
   cp /tmp/25templatedemo root/etc/e-smith/templates/etc/crontab

   # The web panel description
   mkdir -p root/etc/e-smith/web/functions
   cp -p /tmp/loggerdemo !$

   # The web panel implementation
   mkdir -p root/usr/lib/perl5/site_perl/esmith/FormMagick/Panel
   cp -p /tmp/loggerdemo.pm !$

   # The web panel English localisation
   mkdir -p root/etc/e-smith/locale/en-us/etc/e-smith/web/functions
   cp -p /tmp/loggerdemo-en !$

   # The createlinks auxiliary file
   cp -p /tmp/createlinks .
   ```

   Your directory structure should now look like this:

   ```
   [gordonr@sevendev1 loggerdemo-1.0.0]$ find . -type f
   ./root/etc/e-smith/templates/etc/crontab/25templatedemo
   ./root/etc/e-smith/locale/en-us/etc/e-smith/web/functions/loggerdemo
   ./root/etc/e-smith/web/functions/loggerdemo
   ./root/usr/lib/perl5/site_perl/esmith/FormMagick/Panel/loggerdemo.pm
   ./createlinks
   ```

4. Package the directory into a tarball:

   ```
   cd ~/rpms/SOURCES
   tar zcvf loggerdemo-1.0.0.tar.gz loggerdemo-1.0.0
   ```

5. Create the RPM specification "SPEC" file `~/rpms/SPECS/loggerdemo.spec` which looks like this:

   ```
   %define name loggerdemo
   %define version 1.0.0
   %define release 01
   ```

```
Summary: SME Server logger demo
Name: %{name}
Version: %{version}
Release: %{release}
License: GPL
Group: Networking/Daemons
Source: %{name}-%{version}.tar.gz
Packager: Fred Frog <red@example.com>
BuildRoot: /var/tmp/%{name}-%{version}-%{release}-buildroot
BuildArchitectures: noarch

%description
Logger Demo sample application.

%changelog
* Thu Feb 2 2006 Fred Frog <fred@example.com>
- 1.0.0-01
- Original version

%prep
%setup

%build
perl createlinks

DEFAULTS=root/etc/e-smith/db/configuration/defaults/loggerdemo
mkdir -p $DEFAULTS

echo "service"  > $DEFAULTS/type
echo "enabled"  > $DEFAULTS/status
echo "10"       > $DEFAULTS/Interval


%install
rm -rf $RPM_BUILD_ROOT
(cd root ; find . -depth -print | cpio -dump $RPM_BUILD_ROOT)
rm -f %{name}-%{version}-filelist
/sbin/e-smith/genfilelist $RPM_BUILD_ROOT > %{name}-%{version}-filelist

%clean
rm -rf $RPM_BUILD_ROOT

%post
/etc/e-smith/events/actions/initialize-default-databases
/etc/e-smith/events/actions/navigation-conf
/sbin/e-smith/expand-template /etc/crontab
true

%postun
/sbin/e-smith/expand-template /etc/crontab
true

%files -f %{name}-%{version}-filelist
```

```
%defattr(-,root,root)
```

Note the `%post` (post-installation) and `%postun` (post-uninstallation) statements which expand the `/etc/crontab` template after installing or uninstalling the RPM.

6. Check that your RPM will build OK with "build prepare":

   ```
   cd ~/rpms/SPECS
   rpmbuild -bp loggerdemo.spec
   ```

   The last line of output should be `+ exit 0` if rpmbuild is successful.

7. Run the **rpmbuild** command again to actually create your RPM with the "build all" options:

   ```
   rpmbuild -ba loggerdemo.spec
   ```

   If everything was successful, the last line of output should again be `+ exit 0`.

8. The RPMs should have been generated and put into `~/rpms/RPMS/noarch/` as this program can run equally well on any platform. A source RPM should also exist in `~/rpms/SRPMS/`.
9. Test your RPM by installing it on an SME Server test box.

   > **Note:** RPMs need to be installed as `root`, but you should not log in as the root user. Instead, you should create a normal user and provide them with 'root' privileges via the **sudo** command. To provide full *sudo* rights to the user `joe`, use the **su - -c /usr/sbin/visudo** and add the following line to the end of the file:
   >
   > ```
   > joe    ALL=(ALL) ALL
   > ```
   >
   > You can then use the **sudo** to run commands as `root` when required. You will be prompted for your password (not the root password) which ensures that someone else isn't using your terminal.

   ```
   [joe@sevendev1 SPECS]$ sudo rpm -Uvh \
       ~/rpms/RPMS/noarch/loggerdemo-1.0.0-01.noarch.rpm
   Preparing...                ########################################### [100%]
      1:loggerdemo             ########################################### [100%]
   Migrating existing database mailpatterns
   Migrating existing database hosts
   Migrating existing database configuration
   Migrating existing database yum_repositories
   Migrating existing database networks
   Migrating existing database yum_updates
   Migrating existing database yum_installed
   Migrating existing database spamassassin
   Migrating existing database accounts
   Migrating existing database backups
   Migrating existing database yum_available
   Migrating existing database domains
   ```

Note: In almost all instances you should use the `-U` (upgrade) option to rpm instead of the `-i` (install) option. The `-i` option should be reserved for special operations, such as installing kernels.

The customization should be fully installed, and the `/etc/crontab` file should show the customization. Then remove the customization:

```
# remove the RPM
sudo rpm -e loggerdemo
```

The customization should be completely gone, and the `/etc/crontab` file should look the way it did before.

The createlinks script

The source tarballs of an RPM should not include symbolic links as they are difficult to store under many version control systems and cause issues when generating patches. Since the SME Server uses many symbolic links, there are simple methods for creating the ones required. This is done through the **createlinks** script which is called from the `%build` section of the SPEC file. Let's examine one. It starts with the standard Perl script header and an import of the required module:

```
#!/usr/bin/perl -w

use esmith::Build::CreateLinks qw(:all);
```

The **templates2events** function can be used to create the appropriate **templates2expand** links in various events:

```
my $imap  = "/var/service/imap";
my $imaps = "/var/service/imaps";

templates2events("/etc/dovecot.conf", qw(bootstrap-console-save console-save));
templates2events("$imap/config",      qw(bootstrap-console-save email-update));
templates2events("$imaps/config",     qw(bootstrap-console-save email-update));
```

Note that the first argument is a filename and the second argument is a list of events in which to create the link. The **safe_symlink** function can be used to create a generic symbolic link, as well as the directory hierarchy enclosing that link:

```
for my $event (qw(
    email-update
    ldap-update
    network-create
    network-delete
    ))
{
    safe_symlink("sigusr1", "root/etc/e-smith/events/$event/services2adjust/imap");
}
```

```
safe_symlink("daemontools", "root/etc/rc.d/init.d/imap");
```

The **event_link** function is used to create the links from the event directories to the generic actions directory. For example:

```
for my $event (qw(post-upgrade))
{
    event_link("imap-relocate-maildirs", $event, "05");
}
```

creates a symbolic link **S05imap-relocate-maildirs** in the **post-upgrade** event. The target of the symbolic link will be the **imap-relocate-maildirs** script in the `/etc/e-smith/events/actions/` directory.

Finally, the **service_link_enhanced** function makes it simple to create the `/etc/rc.d/rc7.d` and similar startup symlinks:

```
service_link_enhanced("imap", "S55", "7");
service_link_enhanced("imap", "K45", "6");
service_link_enhanced("imap", "K45", "0");
service_link_enhanced("imap", "K45", "1");
```

More documentation on this module can be seen with the command **perldoc esmith::Build::CreateLinks**.

Chapter 14. The SME Server development environment

Configuring your development environment

The SME Server source code is checked into CVS at SourceForge. The project has a SourceForge page at https://sourceforge.net/projects/smeserver/. All packages are built using **mezzanine**, and an overview of mezzanine is provided here: https://wiki.caosity.org/tiki-index.php?page=Package+Maintenance

Reminder: The SME Server is released under the GPL

The SME Server source code is released under the GPL. You must release the source code to all modifications. If you make improvements, please raise a bug and attach a patch so the change can be discussed and pulled back into the base for everyone to share.

Do I need a SourceForge account?

Not really. Only developers who are going to put patches back into CVS and build new packages need SourceForge CVS access. The sources are freely available and patches are gratefully received. Just follow the instructions in this section and attach the patch(es) to the Bugzilla entry, explaining why the change should be made.

Add some useful aliases

Mezzanine needs to know to use the SourceForge CVS repository in order to checkout/get or import a package. Once a package is checked out from SourceForge, mezzanine/CVS will remember where it came from and check it back into the same CVS repository. You should not set the CVSROOT environment variable as it makes it too easy to accidentally import code into the wrong CVS repository when working with multiple repositories.

Add the following lines to `~/.bashrc` (or a file such as `/etc/profile.d/smebuild.sh` to configure it for all users on this machine):

```
export CVS_RSH=ssh # tell CVS to use ssh

# DO NOT set CVSROOT
sf_cvsroot=smeserver.cvs.sourceforge.net:/cvsroot/smeserver
sf_cvsroot_anon=:pserver:anonymous@smeserver.cvs.sourceforge.net:/cvsroot/smeserver

alias mzgetsf="mzget --dir $sf_cvsroot"
alias mzimportsf="mzimport --dir $sf_cvsroot"

alias mzgetsf_anon="mzget --dir $sf_cvsroot_anon"
```

You can now use the **mzgetsf_anon** (or **msgetsf** if you have a developer account) to retrieve a package from SourceForge. If you use **mzget**, you will get an error message to remind you that CVSROOT has not been set, and should not be set. Note that you can use the other mezzanine commands, such as **mzclean** and **mzsync** normally as the CVS repository is already known.

```
[gordonr@smebuild tmp]$ mzget smeserver-yum
cvs checkout: No CVSROOT specified!  Please use the '-d' option
cvs [checkout aborted]: or set the CVSROOT environment variable.
An unknown error must have occured, because the command returned 256
```

Create your work area

It is a good idea to perform all of your checkouts in a standard directory hierarchy. This makes it easier to move around between the packages. On a shared build server, it also makes it easier for people to recover your work if required (for example when you are on holidays). If you want to use filesharing to edit files from a networked PC, you might like to use `~/home/smeserver/`. On a shared build server, you might even use something like `/builds/users/$LOGNAME/smeserver/` so the files are not in your home directory and are accessible to all developers.

> **Note:** Reminder: All files must be checked in using Unix text format.

```
[gordonr@smebuild ~]$ mkdir ~/smeserver

[gordonr@smebuild ~]$ cd ~/smeserver
```

Modifying an SME Server package

Raise a Bugzilla entry

Before you make any changes to a package, you need to have a Bugzilla entry which specifies the problem and preferably proposes a fix. Raising the bug *before* you do the work allows others to comment on the proposed approach and can save significant time when you go to submit the changes. The change should also be approved by the Development Manager if it is meant for near-term release. You will need the Bugzilla bugid when you check in the changes.

All changes must have an associated Bugzilla entry. The bug tracker is here: http://www.contribs.org/bugzilla/

If a relevant bug does not exist, raise one. If the bug exists, assign it to yourself to show that you are working on it:

For this exercise, let's look at bug 1174 "yum-import-keys should not import duplicates" http://www.contribs.org/bugzilla/show_bug.cgi?id=1174 (http://www.contribs.org/bugzilla/show_bug.cgi?id=1174/).

Choose the package(s) to modify

If you are modifying an existing file, the simplest way to determine the package is to install the relevant version and run **rpm -qf** on the file to be modified:

```
[gordonr@smebuild actions]$ rpm -qf /etc/e-smith/events/actions/yum-import-keys
smeserver-yum-1.1.2-05
```

and so, we want to modify the `smeserver-yum` package.

> **Note:** You can view a complete list of the packages checked into SourceForge CVS http://smeserver.cvs.sourceforge.net/smeserver/. On rare occasions, the sources of particular packages may be slightly out of date as there is a small delay (typically only a few hours) between changes to the developer CVS and the anonymous CVS. In practice, this rarely matters and at worst it requires a merging of your changes with any other recent changes to the package.

All packages on the SME Server ISO/CD must be checked into SourceForge CVS. The only exceptions are packages which come from the following upstream repositories: CentOS and dag.

mzgetsf_anon: SourceForge anonymous CVS checkout

You can now retrieve one of the packages from SourceForge. In this case, we want to modify the `smeserver-yum` package, so let's retrieve it from SourceForge:

```
[gordonr@smebuild smeserver]$ mzgetsf_anon smeserver-yum
U smeserver-yum/ChangeLog
U smeserver-yum/F/smeserver-yum.spec
U smeserver-yum/P/smeserver-yum-1.2.0-DisplayStatus.patch
U smeserver-yum/P/smeserver-yum-1.2.0-DisplayStatus.patch2
U smeserver-yum/P/smeserver-yum-1.2.0-ModifyUpdateDBs.patch
U smeserver-yum/S/smeserver-yum-1.2.0.tar.gz
```

> **Note:** Use **mzgetsf** if you have a SME Server developer account.

Mezzanine package hierarchy

```
[gordonr@smebuild smeserver]$ cd smeserver-yum/

[gordonr@smebuild smeserver-yum]$ ls
ChangeLog  CVS  F  P  S  smeserver-yum
```

If you examine the checked out package, you will see the following directories:

- The SME Server project does not use the `ChangeLog` file as we maintain all of our changes in the `%changelog` section of the `SPEC` file.

- The `CVS` directory contains CVS state information, such as the location of the CVS repository:

  ```
  [gordonr@smebuild smeserver-yum]$ ls CVS
  Entries  Entries.Log  Repository  Root
  ```

- The `F` directory contains the `SPEC` for this package:

  ```
  [gordonr@smebuild smeserver-yum]$ ls F
  CVS  smeserver-yum.spec
  ```

- The `P` directory contains any patches which are required to update the source in the tarball(s) to the current version:

  ```
  [gordonr@smebuild smeserver-yum]$ ls P
  CVS
  smeserver-yum-1.2.0-DisplayStatus.patch
  smeserver-yum-1.2.0-DisplayStatus.patch2
  smeserver-yum-1.2.0-ModifyUpdateDBs.patch
  ```

- The `S` directory contains the source tarball(s):

  ```
  [gordonr@smebuild smeserver-yum]$ ls S
  CVS  smeserver-yum-1.2.0.tar.gz
  ```

Note: The `smeserver-yum` subdirectory is a leftover from an incorrect package import. It can be ignored and will be removed when maintenance is next performed on the CVS repository.

mzclean: Clean out modified files

Before making changes to a package, you should ensure that you have a clean copy of the latest sources. Remember - other people may also be working on the package and making changes in SourceForge. You can perform a "merge" when you check in your changes, but it is easiest to start with an up-to-date copy. It is even possible that someone else has already fixed the bug you are working on!

```
[gordonr@smebuild smeserver-yum]$ mzclean
Cleaning and resyncing smeserver-yum, please wait....
Removing work....
Cleanup of smeserver-yum complete.

[gordonr@smebuild smeserver-yum]$ ls
ChangeLog  CVS  F  P  S
```

Tip: Ensure that you have copied any modified files out of your work tree prior to runing **mzclean**.

Note that the stray `smeserver-yum` subdirectory has been removed.

mzprep: Build working tree

The **mzprep** command runs **rpmbuild -bp** on your SPEC file, sources and patches to generate a working tree. You can make changes in the working tree and use mezzanine to generate the patches between the previous version and your new files.

```
[gordonr@smebuild smeserver-yum]$ mzprep
Creating working directory /home/e-smith/files/users/gordonr/tmp/sf/smeserver-yum/work...
You may now chdir to work to make changes.
Use "mzpatch -n <patch_name>" to generate a patch when done.

[gordonr@smebuild smeserver-yum]$ ls
ChangeLog  CVS  F  P  S  work
```

The only obvious change is the creation of a `work/` subdirectory. Let's have a look in there. Under the `work/` directory is `smeserver-yum-1.2.0/` which contains the *patched* sources for this version of `smeserver-yum`.

```
[gordonr@smebuild smeserver-yum]$ ls work/
smeserver-yum-1.2.0
```

Under `smeserver-yum-1.2.0/` are the `createlinks` script, the `root` hierarchy and a few generated files, explained below.

```
[gordonr@smebuild smeserver-yum]$ ls work/smeserver-yum-1.2.0/
createlinks      debugsources.list  smeserver-yum-1.1.2-26-filelist
debugfiles.list  root
```

SME Server package directory layout

Table 14-1. SME Server package directory layout

Directory/File	Description
createlinks	Builds action, panel and initscript links.
po	Package translations, if gettext is used.
root	The top-level directory of the package, which will be the `/` directory of the installed system. So, `root/usr/lib/perl5/site_perl` becomes `/usr/lib/perl5/site_perl` on the installed system. Note that the somewhat confusing `root/root` is the `/root` directory of the installed system.
update-po	Helper script to generate translation binary files, if gettext is used.

Directory/File	Description
debugsources.list debugfiles.list	Generated by RPM build - lists this files which are built with debug flags
smeserver-yum-1.1.2-26-filelist	Generated by RPM build - lists the files to be packaged, and their permissions and ownership

The root directory hierarchy

The `work/smeserver-yum-1.2.0/root/` hierarchy contains the files which will be installed on the target machine.

```
[gordonr@smebuild smeserver-yum]$ ls work/smeserver-yum-1.2.0/root/
etc  sbin  service  usr  var
```

Modifying a file

OK - now we can actually make a change. We want to modify the file which will be installed as `/etc/e-smith/events/actions/yum-import-keys`, so the actual file we need to modify is: `work/smeserver-yum-1.2.0/root/etc/e-smith/events/actions/yum-import-keys`.

Use your favourite editor (in Unix text mode) to modify the file, adding a comment where the change needs to be made. Here's a **diff** showing such a comment:

```
[gordonr@smebuild tmp]$ diff -u yum-import-keys.orig yum-import-keys
--- yum-import-keys.orig        2006-05-22 16:28:51.734609534 +1000
+++ yum-import-keys     2006-05-22 16:28:39.006814950 +1000
@@ -30,6 +30,7 @@
 for my $key ( grep {!/^\./} readdir(DIR) )
 {
     warn "Importing key $key\n";
+    # TODO: Skip keys which have previously been imported

     system("rpm", "--import", $key) == 0 or
        warn "Couldn't rpm --import $key\n";
```

mzpatch: Create a patch

Mezzanine automates the creation of patches through the **mzpatch** command.

> **Note:** Always run **mzpatch** from the top-level directory of your package (e.g. `~/smeserver/smeserver-yum/`).

```
[gordonr@smebuild actions]$ cd ~/smeserver/smeserver-yum/
```

```
[gordonr@smebuild smeserver-yum]$ ls P
CVS
smeserver-yum-1.2.0-DisplayStatus.patch
smeserver-yum-1.2.0-DisplayStatus.patch2
smeserver-yum-1.2.0-ModifyUpdateDBs.patch
```

You will see from the listing above that the patches should be named *package-version-patchname.patch*. You should also ensure that the patch name you are about to choose doesn't already exist, or you may overwrite an older version of the patch.

```
[gordonr@smebuild smeserver-yum]$ mzpatch -n smeserver-yum-1.2.0-ImportKeysComment.patch
Creating working directory /home/e-smith/files/users/gordonr/tmp/sf/smeserver-yum/work...
Created P/smeserver-yum-1.2.0-ImportKeysComment.patch (11 lines).
cvs [server aborted]: "add" requires write access to the repository
You do not have write access to the master repository.
srctool:  Error:  Addition of P/smeserver-yum-1.2.0-ImportKeysComment.patch failed.
[gordonr@smebuild smeserver-yum]$ ls P
CVS
smeserver-yum-1.2.0-DisplayStatus.patch
smeserver-yum-1.2.0-DisplayStatus.patch2
smeserver-yum-1.2.0-ImportKeysComment.patch
smeserver-yum-1.2.0-ModifyUpdateDBs.patch
```

Mezzanine has created the patch, but it has not committed it to CVS. This allows you to revise the patch until it works, prior to committing it. If you do not have a developer account you will get an error when mezzanine attempts to reserve the patch name in CVS, but the patch will still be created in the `P` directory.

The patch looks quite similar to the one we saw previously, except for the pathname to the modified file:

```
[gordonr@smebuild smeserver-yum]$ cat P/smeserver-yum-1.2.0-ImportKeysComment.patch
diff -Nur -x '*.orig' -x '*.rej' smeserver-yum-1.2.0/root/etc/e-smith/events/actions/yum-
--- smeserver-yum-1.2.0/root/etc/e-smith/events/actions/yum-import-keys 2005-09-26 12:30:
+++ mezzanine_patched_smeserver-yum-1.2.0/root/etc/e-smith/events/actions/yum-import-keys
@@ -30,6 +30,7 @@
 for my $key ( grep {!/^\./} readdir(DIR) )
 {
     warn "Importing key $key\n";
+    # TODO: Skip keys which have previously been imported

     system("rpm", "--import", $key) == 0 or
        warn "Couldn't rpm --import $key\n";
```

Building a new package, with the patch

The next step is to change the package `SPEC` file so that the patch is applied.

Update the Release tag

The *first* change you should make is to update the "Release" tag in the `SPEC` file. You should do this by appending your initials and sequence number to the `Release` number. In most SME Server `SPEC` files, the `Release` number is set via the `release` macro, and so you should set it there.

```
Summary: YUM, an rpm updater
%define name smeserver-yum
Name: %{name}
%define version 1.2.0
- %define release 05
+ %define release 05ff01
```

Please just append your initials and a sequence number to the release tag. Do not change the package name (e.g. to smeserver-yum-fred) or the version number. This policy makes it obvious that your update is a patch based on `smeserver-yum-1.2.0-05`.

If a change has been approved by the Development Manager, it is given an official version/release number (e.g. smeserver-yum-1.2.0-06). *Always tag unofficial changes with your initials and sequence number.*

If an upstream (e.g. CentOS) package needs to be patched and it is not possible to wait for the patch to come from the upstream source, the release number should be modified with the special tag `sme` and a sequence number (e.g. clamav-0.88.2-1sme01). *The sme tag is reserved for changes approved by the Development Manager.*

Add a changelog entry

The SME Server project maintains history of packages changes in the `%changelog` section of the `SPEC` file. The changelog entry should be a simple description of the change of behaviour and must include a reference to an SME Server project Bugzilla entry (1174) and the new version number (1.2.0-05ff01):

```
%changelog
+ * Mon May 22 2006 Fred Frog <fred@example.com> 1.2.0-05ff01
+ - Initial work on avoiding duplicate RPM keys [SME: 1174]

* Mon May  1 2006 Charlie Brady <charlieb@e-smith.com> 1.2.0-05
- Remove stray yum.pm.orig file. [SME: 1350]
```

The first line of the changelog entry shows when the change was made, by whom and notes the new version number. The other lines describe the change and refer to the Bugzilla entry or entries, using the format `[SME: 1174]` as a shorthand.

Add a patch reference

You now need to tell RPM to retrieve and apply the new patch. This needs to be done in two places, which is annoying, but allows for patches to be carried around, but not applied (e.g. because they are not quite finished). The first change is near the top of the file:

```
Patch0: smeserver-yum-1.2.0-DisplayStatus.patch
Patch1: smeserver-yum-1.2.0-ModifyUpdateDBs.patch
```

```
Patch2: smeserver-yum-1.2.0-DisplayStatus.patch2
+ Patch3: smeserver-yum-1.2.0-ImportKeysComment.patch
```

The second change is in the `%prep` section:

```
%prep
%setup
%patch0 -p1
%patch1 -p1
%patch2 -p1
rm root/usr/lib/perl5/site_perl/esmith/FormMagick/Panel/yum.pm.orig
+ %patch3 -p1
```

mzbuild: Build a new package

With all that done, you should be able to build a new package using **mzbuild**:

```
[gordonr@smebuild smeserver-yum]$ mzbuild

[...]
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.35037
+ umask 022
+ cd /home/e-smith/files/users/gordonr/smeserver/smeserver-yum/build.mezz/BUILD
+ cd smeserver-yum-1.2.0
+ /bin/rm -rf /var/tmp/mezzanine-buildroot.18508
+ exit 0
Package build succeeded.  Output files are:

smeserver-yum-1.2.0-05ff01.src.rpm
smeserver-yum-1.2.0-05ff01.noarch.rpm
```

Check the package

You should now perform some simple checks on the package before installing it for testing.

- Does the built package have your initials in the release tag? We never want to have two packages with the same name and version number.

  ```
  [gordonr@smebuild smeserver-yum]$ ls *.rpm
  smeserver-yum-1.2.0-05ff01.noarch.rpm  smeserver-yum-1.2.0-05ff01.src.rpm
  ```

- Did you update the changelog? It's easy to forget, but vital to do.

  ```
  [gordonr@smebuild smeserver-yum]$ rpm -qp --changelog \
    smeserver-yum-1.2.0-05ff01.noarch.rpm | head -8

  * Mon May 22 2006 Fred Frog <fred@example.com> 1.2.0-05ff01

  - Initial work on avoiding duplicate RPM keys [SME: 1174]
  ```

```
* Mon May 01 2006 Charlie Brady <charlieb@e-smith.com> 1.2.0-05

- Remove stray yum.pm.orig file. [SME: 1350]
```

- Did you remember to apply the patch? It happens, and leads to head scratching "I knew I made that change".

  ```
  [gordonr@smebuild smeserver-yum]$ mzbuild | grep patch
  + echo 'Patch #0 (smeserver-yum-1.2.0-DisplayStatus.patch):'
  Patch #0 (smeserver-yum-1.2.0-DisplayStatus.patch):
  + patch -p1 -s
  + echo 'Patch #1 (smeserver-yum-1.2.0-ModifyUpdateDBs.patch):'
  Patch #1 (smeserver-yum-1.2.0-ModifyUpdateDBs.patch):
  + patch -p1 -s
  + echo 'Patch #2 (smeserver-yum-1.2.0-DisplayStatus.patch2):'
  Patch #2 (smeserver-yum-1.2.0-DisplayStatus.patch2):
  + patch -p1 -s
  + echo 'Patch #3 (smeserver-yum-1.2.0-ImportKeysComment.patch):'
  Patch #3 (smeserver-yum-1.2.0-ImportKeysComment.patch):
  + patch -p1 -s
  ```

Test your new package

You should now upgrade your test system:

```
[gordonr@sevendev1 tmp]$ sudo rpm -Uvh smeserver-yum-1.2.0-05ff01.noarch.rpm
Preparing...                ########################################### [100%]
   1:smeserver-yum          ########################################### [100%]
```

and test that you have fixed the bug.

Lather, rinse, repeat

If you haven't fixed the bug, repeat the cycle: make changes, build a new patch, update the release number (e.g. to 05ff02, 05ff03, etc.), build a new package, test. Don't worry about using up release numbers.

> **Note:** Actually fixing bug 1174 is left as an exercise for the reader. Fame and fortune (or at least thanks) await the person who does.

Attach your patch(es) to the bug

You should now attach your patch(es) and SPEC file changes to the appropriate Bugzilla entry. If the change is the simple application of a patch, you do not need to attach the whole SPEC file - just the changelog entry.

Please attach the patch(es) and ensure that they are in Unix text format. Patches make it obvious which sections of code have been changed and attaching them ensures that their formatting is preserved.

SourceForge developer CVS access

If you are going to work on many bugs, or wish to help with the many tasks involved in a release, you should create a SourceForge developer account.

You do not need a SourceForge account in order to do development. You can contribute very effectively by producing patches and attaching them to the Bugzilla entries.

Create a SourceForge account

- Create a SourceForge user id
- Contact the Development Manager (or one of the other project admins) listed on the SourceForge project page, detailing the packages you would like to work on
- The project admin will add you as a developer
- Upload an SSH2 key to SourceForge, following the SourceForge documentation: https://sourceforge.net/docs/F02/en/#key_posting
- You will also need to generate another SSH2 key for access to the smebuild build server

mzgetsf: SourceForge developer CVS checkout

Once you have SourceForge CVS access, you can use **mzgetsf** (instead of **mzgetsf_anon**) to retrieve the sources. Your copy will be writable and you will be able to put changes back to CVS. When you run **mzpatch** you should not see an error during the **cvs add** step.

```
[gordonr@smebuild smeserver-yum]$ mzpatch -n smeserver-yum-1.2.0-ImportKeysComment.patch
Creating working directory /home/e-smith/files/users/gordonr/smeserver/smeserver-yum/work
Created P/smeserver-yum-1.2.0-ImportKeysComment.patch (11 lines).
cvs add: use 'cvs commit' to add this file permanently
Patch added to SCM.  Use 'mzput' to upload to repository.
```

mzput: Put the changes back to CVS

Instead of attaching the patch to the Bugzilla entry, you will be able to put it back to CVS. Mezzanine normally edits the `ChangeLog`, but we don't need to do that as we keep the changes in the `%changelog` section of the `SPEC` file. The `-m` option to **mzput** allows a simple comment to be appended, which is all that is required:

```
[gordonr@smebuild smeserver-yum]$ mzput -m 'See changelog'
```

Build the official package

Official packages have unadorned release tags (e.g. smeserver-yum-1.2.0-05) and must be built on the official build servers. There are three steps involved - build the RPMs (**mzbuild**), sign the RPMs (**rpm --addsign *.rpm**) and release the RPMs (**release_rpms *.rpm**).

mzimportsf: SourceForge package import

If a package has not already been checked into SourceForge CVS, you can use **mzimportsf** to import it. You should be careful to run this command from a directory which does not contain an imported package or mezzanine may import the package underneath the existing package.

mzmerge: Merge changed source RPM

If a new source RPM has been built without corresponding changes in SourceForge CVS, you can use **mzmerge** to merge in the changes. This can also be used to merge in changes from a modified upstream package. Care should be exercised with this command, especially if you want to maintain previous patches.

IV. Advanced customization of the SME Server

Chapter 15. Advanced customization principles

Leveraging the provisioning system for users, groups, and i-bays

One of the themes in the SME Server is that concepts such as users, groups, and shared information (information bays) are *simplified* and *reused* in the user interface. SME Server users are email users, filesharing users, web users and users for any other sofware installed on the system.

For example, in the user interface you can create an information bay called *salesdata* representing information of interest to the sales team. Creating the information bay automatically reconfigures Samba and Netatalk to share *salesdata* as a new shared folder, reconfigures Apache to present `http://www.example.com/salesdata/` as a new part of the web site, and reconfigures the FTP server - so that the information can be accessed by logging in as user *salesdata*.

Another example of this type of *concept-reuse* is that you can create a group called *marketing* that will, among other things, create an email alias called *marketing* to automatically forward email to all the group members. This group can also be used as a unit of information-sharing.

In order to enable this concept-reuse, there are certain namespace restrictions. You cannot have a user account and an information bay with the same name - since there would be ambiguity when logging into the FTP server. You cannot have a user account and a group with the same name either - since there would be ambiguity when sending email to the server.

To enforce these restrictions, the SME Server defines a concept of *account*. Users, groups, and information bays are all different types of account. No two accounts can have the same name. The account list is maintained in the *accounts* database.

Whenever a user, group, or information bay is created, the following steps are performed automatically by the SME Server:

1. Check if there is an existing account (of any type) with the same name. If so, display an error and terminate.
2. If there was no error, then create a new *accounts* database entry. The entry contains the name of the account, its type (e.g. user, group, ibay), and all associated properties.
3. Signal the *create* event for that account type - **user-create**, **group-create**, **ibay-create**, and so on.
4. The actions for that event will then do all the work to set up the account - creating underlying user accounts if necessary, creating groups and directories, reconfiguring services, and so on.

The SME Server supports the following account types:

Table 15-1. SME Server software

Account type	Purpose
User	Individual users of the system with local email accounts, home directories, etc.

Account type	Purpose
Group	A list of users. All applications which require a list of users should use the standard SME Server group mechanism. They should extend the properties of the group, if required, but should not create additional group types - group lists, work groups, etc.
Information bay	A shared storage area - shared folder, intranet, extranet, etc.
System	Any account name that is reserved by the SME Server for internal use.
URL	Any subdirectory of the primary web site (e.g. "webmail")
Pseudonym	Any email alias for a user or group
Printer	Any shared printer

When creating applications, you should always try to make use of the built-in SME Server account types. If your application has any concept of users, groups, or shared data - try to make your application use the built-in SME Server mechanisms for all of these.

Programmatically creating users, groups, and i-bays

You can create users, groups, and i-bays by creating database defaults, or through code. Refer to the useraccounts, groups and ibays panels for examples of how to create these items. You can also create accounts with simple shell scripts.

For example, here is a shell script to create a user (account "abc", name "Albert Collins"):

```
#!/bin/sh

PATH=/sbin/e-smith:$PATH

if db accounts get abc >/dev/null
then
    echo "abc already exists in the accounts database"
    db accounts show abc
    exit 1
fi

db accounts set abc user PasswordSet no
db accounts setprop abc FirstName Albert
db accounts setprop abc LastName Collins
db accounts setprop abc EmailForward local

signal-event user-create abc
```

Note that we could have provided all of the properties to the **set** command and created the record in one step. Here's the same example using the Perl libraries

```
#!/usr/bin/perl -w

use strict;
use warnings;
use esmith::AccountsDB;
```

```
my $db = esmith::AccountsDB->open or die "Couldn't open AccountsDB\n";

my $abc = $db->get("abc");

if ($abc)
{
    die "abc already exists in the accounts database\n" .
        $abc->show . "\n";
}

$db->new_record("abc",
        {
            type         => 'user',
            PasswordSet  => 'no',
            FirstName    => 'Albert',
            LastName     => 'Collins',
            EmailForward => 'local',
        });

unless ( system("/sbin/e-smith/signal-event", "user-create", "abc") == 0 )
{
    die "user-create abc failed\n";
}

exit 0;
```

Reserving accounts to avoid conflicts with user, group, or i-bay names

If your application creates a new directory within your web site e.g. `http://www.example.com/magicstuff/`, you should make sure the name isn't *also* used for an information bay, since that would create a conflict. Simply reserve the name by creating a *url*account. This can be done by creating a defaults file:

```
cd /etc/e-smith/db/accounts/defaults/

mkdir magicstuff
cd magicstuff

echo url >type
```

If you package the file in your RPM, the account will be created automatically. To test your change before packaging, you'll need to tell the SME Server to reconfigure the databases:

```
/etc/e-smith/events/actions/initialize-default-databases

db accounts show magicstuff
```

Adding new account properties

Just as you can spontaneously introduce new configuration settings you can spontaneously introduce new properties as well.

> **Note:** You should not create new options for existing properties. For example, if the server-manager can only set three possible values, you should not invent a fourth one. Use another property and raise a bug to suggest the required changes.

For example, let's say that your application needs a concept of *cell phone number* stored for each user account. This is not a standard property in the SME Server. Your application can simply choose a name for the new property, e.g. `CellNumber`, and immediately start reading and writing that property for the various users - as though the property had always existed.

If you *read* from a non-existant property, an empty string is returned for shell scripts and the *undef* value is returned when using the Perl interfaces. If you *write* to a non-existent property, it is spontaneously created in the `accounts` database.

Here is an example of a user interface screen which allows you to edit cell phone numbers for each user account. As before, the form descriptions goes in `/etc/e-smith/web/functions/cellnumbers`:

```
#!/usr/bin/perl -wT
# vim: ft=xml ts=4 sw=4 et:
#----------------------------------------------------------------------
# heading     : Collaboration
# description : Cell numbers (fm)
# navigation  : 3000 3150
#----------------------------------------------------------------------
use strict;
use esmith::TestUtils;
use esmith::FormMagick::Panel::cellnumbers;

my $fm = esmith::FormMagick::Panel::cellnumbers->new();

$fm->display();

__DATA__
<form title="FORM_TITLE"
    header="/etc/e-smith/web/common/head.tmpl"
    footer="/etc/e-smith/web/common/foot.tmpl">

    <page name="First" pre-event="print_status_message()">

        <description>FORM_DESCRIPTION</description>

        <subroutine src="print_cellnumbers_table()" />
    </page>

    <page name="CELLNUMBERS_PAGE_MODIFY"
            pre-event="turn_off_buttons()"
            post-event="modify_cellnumber()" >
        <description>MODIFY_TITLE</description>

        <field type="literal" id="User" >
            <label>LABEL_USER</label>
        </field>

        <field type="literal" id="FullName">
            <label>LABEL_FULLNAME</label>
        </field>

        <field type="text" id="CellNumber">
            <label>LABEL_CELLNUMBER</label>
        </field>

        <subroutine src="print_button('SAVE')" />
    </page>
</form>
```

And the form implementation goes in
`/usr/lib/perl5/site_perl/esmith/FormMagick/Panels/cellnumbers.pm`:

```
#!/usr/bin/perl -w
package    esmith::FormMagick::Panel::cellnumbers;

use strict;

use esmith::FormMagick;
use esmith::AccountsDB;
use esmith::ConfigDB;

use Exporter;
use Carp qw(verbose);

use HTML::Tabulate;

our @ISA = qw(esmith::FormMagick Exporter);

our @EXPORT = qw();

our $db = esmith::ConfigDB->open();
our $adb = esmith::AccountsDB->open();

sub new
{
    shift;
    my $self = esmith::FormMagick->new();
    $self->{calling_package} = (caller)[0];
    bless $self;
    return $self;
}

sub print_cellnumbers_table
```

```
{
    my $self = shift;
    my $q = $self->{cgi};

    my $cellnumbers_table =
    {
       title => $self->localise('CURRENT_LIST_OF_CELLNUMBERS'),

       stripe => '#D4D0C8',

       fields => [ qw(User FullName CellNumber Modify) ],

       labels => 1,

       field_attr => {
                       User => { label => $self->localise('USER_LABEL') },

                       FullName => { label => $self->localise('FULLNAME_LABEL') },

                       CellNumber => { label => $self->localise('CELLNUMBER_LABEL') },

                       Modify => {
                                   label => $self->localise('MODIFY'),
                                   link => \&modify_link },
                                 }
           };

    my @data = ();

    my $modify = $self->localise('MODIFY');

    for my $user ($adb->users)
    {
        push @data,
            {
              User => $user->key,

              FullName => $user->prop('FirstName') . " " .
                          $user->prop('LastName'),

              CellNumber => $user->prop('CellNumber') || '',

              Modify => $modify,
            }
    }

    my $t = HTML::Tabulate->new($cellnumbers_table);

    $t->render(\@data, $cellnumbers_table);
}

sub modify_link
{
    my ($data_item, $row, $field) = @_;

    return "cellnumbers?" .
            join("&",
                "page=0",
                "page_stack=",
                "Next=Next",
                "User="     . $row->{User},
                "FullName=" . $row->{FullName},
                "CellNumber=" . $row->{CellNumber},
                "wherenext=CELLNUMBERS_PAGE_MODIFY");
}

sub modify_cellnumber
{
    my $self = shift;
    my $q = $self->{cgi};

    my $user = $adb->get( $q->param('User') );

    $user->set_prop('CellNumber', $q->param('CellNumber'));

    return $self->success('SUCCESSFULLY_MODIFIED');
}

1;
```

Save the two files in the correct locations and then set the correct permissions and ownership:

```
cd /etc/e-smith/web/functions
chown root:admin cellnumbers
chmod 4750 cellnumbers
```

Then create a symbolic link to the script from the web manager `cgi-bin/` directory:

```
cd /etc/e-smith/web/panels/manager/cgi-bin
ln -s ../../../functions/cellnumbers cellnumbers

/etc/e-smith/events/actions/navigation-conf
```

If you refresh the navigation bar, you will see a *Cell numbers* screen, which can be used to edit cell phone numbers for each user.

You could easily package this into an RPM and would just need the `cellnumbers` description, the `cellnumbers.pm` implementation and the symbolic link in the RPM. If you installed this application on any SME Server you could immediately start entering cell phone numbers for each user.

Using the LDAP server

The SME Server automatically creates and maintains an LDAP *address book*. The LDAP server listens for requests on port 389, which is the standard TCP/IP port for LDAP. At this time, the LDAP server should be considered read-only as it is generated from the system configuration and accounts data. Changes to the LDAP schema will be backed up and restored, but major system reconfiguration may reset the LDAP database to the default schema.

Data backup

The SME Server supports two methods for data backup. For light-usage sites, end users can use their web browser to select a *backup to desktop* option; this creates a compressed file of the configuration databases and all user data on the server, and uploads it to the user's desktop via their web browser.

Note: The desktop backup is limited to 2GBytes of data on most operating systems.

For heavier-usage sites, automatic nightly tape backup can be configured.

Third party application writers do not need to make special backup arrangements. All that is required is to ensure that all data files are placed within the standard directories that are backed up. All files and directories within the `/home/e-smith/files/` tree are always backed up by all of the SME Server backup mechanisms.

There is a **pre-backup** event which is signalled before a backup is performed. This can be used to shutdown applications or databases to ensure that a consistent state is backed up. The SME Server automatically performs an ASCII export of all **MySQL** databases in **pre-backup** event.

There is a corresponding **post-backup** event which is signalled after the backup has been performed. This can be used to restart services after the backup.

Using the MySQL database

The SME Server provides a standard method for performing MySQL database initialization and migration. This is done by creating files in the `/etc/e-smith/sql/init/` directory. These files are run automatically when MySQL is started, and deleted if they run successfully.

A separate MySQL database and one or more database users should be created for each application. The database password should be stored in the configuration database and either retrieved from the configuration database by the application or passed to the application via an `httpd.conf` fragment. The password should be automatically generated, unique to this server and this application, and stored as a property in the configuration database for later use in database scripts.

Note: Database passwords required for application configuration files should be retrieved from the configuration database.

The MySQL root is automatically generated and configured for command-line MySQL use by the root system user. The MySQL root password should only be used for database maintenance such as creating and deleting databases and performing database backups.

> **Note:** *Applications should never use the MySQL root password for access to the database and it should never be entered into application configuration files.*

First choose a name for your database, as well as a username to access the data. For example, let’s say your database is called **loggerdemo**, the username is **loguser** and the password is **$loggerdemo{DbPassword}**. A migrate fragment like this might be used to create the password:

```
{
    my $rec = $DB->get('loggerdemo')
        || $DB->new_record('loggerdemo', {type => 'service'});

    my $password = $rec->prop('DbPassword');
    return "" if $password;

    use MIME::Base64;

    my $rand     = sprintf("%08d", int(1_000_000_000 * rand()));
    my $password = MIME::Base64::encode($rand, "");

    $rec->set_prop('DbPassword', $password);
}
```

Then create a template which generates a file in the `/etc/e-smith/sql/` directory, and put the relevant SQL commands in that file. The SQL commands should set up the application’s username and retrieve the database password from the configuration database. It creates the new MySQL database and any tables required by your application. Write these SQL commands using the IF NOT EXISTS clause so that they do nothing if the tables have already been created. For example, you might create the template `/etc/e-smith/templates/etc/e-smith/sql/loggerdemo-create-schema.sql` with the following contents:

```
# Create the user account and password. (This is harmless if the
# user account and password already exist.) Note the reference
# to the 'loggerdemo' database which will be created in the next
# few statements.

USE mysql;

REPLACE INTO user (host, user, password)
    VALUES ('localhost', 'loguser', PASSWORD ('{ $loggerdemo{DbPassword} }'));

REPLACE INTO db (host, db, user,
                   select_priv, insert_priv, update_priv,
                   delete_priv, create_priv, drop_priv )
     VALUES ('localhost', 'loggerdemo', 'loguser',
             'Y', 'Y', 'Y', 'Y', 'Y', 'Y');

FLUSH PRIVILEGES;

# Create 'loggerdemo' database. (Do nothing if the database
# already exists.)
```

```
CREATE DATABASE IF NOT EXISTS loggerdemo;

# Create log_entry table within the 'loggerdemo' database.
# (Do nothing if the table already exists.)

USE loggerdemo;

CREATE TABLE IF NOT EXISTS log_entry
(
    entry_message  varchar(60),
    entry_time     datetime
);
```

Include the migrate fragment and your template in your RPM. Note that the password generated in this way is unique to this server and this application and automatically stored in the configuration database for later use. This means that it is backed up and restored through the normal server operations.

> **Note:** For more documentation on MySQL schema creation commands, see: http://www.mysql.com/documentation/mysql/bychapter/

In the post-installation section of your RPM, expand the template, and run the `/etc/rc.d/init.d/mysql.init` script. For example the post-installation section of your RPM SPEC file might look like this:

```
%post
expand-template /etc/e-smith/sql/init/loggerdemo-create-schema.sql
/etc/rc.d/init.d/mysql.init
true
```

Installing this RPM will create the `/etc/e-smith/sql/loggerdemo-create-schema.sql` templates (because it is part of the RPM), and the post-installation actions will expand the template and run the `mysql.init` script, which will execute the schema creation commands and delete the generated file. When the RPM installation is finished, the database schema will have been created, and the MySQL database will be ready to process SQL commands from your application.

It is also possible to perform MySQL initialization in languages other than SQL, for example if the logic is better suited to another language, simply by creating a file in the `/etc/e-smith/sql/init/` directory. The file must be executable and not have a `.sql` extension. For example, the template expansion might generate this file:

```
#! /bin/sh

exec mysql < /home/httpd/html/horde/scripts/db/mysql_create_tables.sql
```

You can use the `templates.metadata` mechanism to ensure the correct permissions on the generated file. Remember, the files are removed from the `/etc/e-smith/sql/init/` directory if they run successfully.

It is important to think through what will happen when your application is installed, uninstalled, reinstalled, or upgraded. The instructions described above do not specify any uninstallation procedure - therefore the database tables will be left unchanged if your application is removed, reinstalled, or upgraded. If you want the data to be deleted when the application is removed, create a *post-uninstallation* script using the same technique as the post-installation script.

The instructions above apply to an application with a schema that does not evolve. If you create a new version of your application that requires schema changes, your post-installation script will have to migrate the database from the old to the new schema. In that case you have two options. Say the original version of the application is 1.0, and the new version is 1.1.

1. The first option is to release two versions of the 1.1 application - one for new installations (containing SQL commands to create a new schema), and a second version for upgrading 1.0 installations (containing SQL commands to upgrade the 1.0 schema). The RPM mechanism allows you to specify dependencies to ensure that only the correct version of each RPM can be installed on a given SME Server.
2. The better option is to change the template so that it includes the appropriate MySQL code to query the database and automatically determine whether to migrate an existing schema or create a new one. The `e-smith-horde` package contains a number of MySQL database initialisation and migration scripts which may be useful to study.

Sending email messages

If your application needs to send an email message, it should use the SMTP protocol and send the message through the local SMTP server (connect to localhost, port 25).

There are many toolkits available to make this simpler, for example the `Mail::Send` library (see **perldoc Mail::Send**) if you are sending the message from a Perl program.

Managing the firewall

The SME Server approach provides better security than a typical firewall, because the SME Server is managed automatically. Conventional firewalls have complex user interfaces, and require a system administrator to choose policies (e.g. which services should be permitted, which ports should be forwarded, etc.) The SME Server firewall has no user interface. It automatically generates the best ruleset that is consistent with the server settings, and is automatically regenerated whenever the server settings are changed.

Creating firewall pinholes for your application

Let us say that your service needs to provide a public service on TCP/IP port 4321, which is normally blocked by the firewall rules. All that you need to do is define this to the SME Server

```
config set myservice service TCPPort 4321 access public status enabled

signal-event remoteaccess-update
```

Note that a firewall hole is only opened if three things are true - the service has a TCPPort (or UDPPort) definition, the service is set to public access, and the service is enabled. Run the commands above, and then these ones:

```
cp /etc/rc.d/init.d/masq /tmp

config setprop myservice access private

signal-event remoteaccess-update

diff -u /etc/rc.d/init.d/masq /tmp/masq
```

This will produce output something like this:

```
[root@gsxdev1 esmith]# diff -u /tmp/masq /etc/rc.d/init.d/masq
--- /tmp/masq   2006-02-02 13:14:09.000000000 +1100
+++ /etc/rc.d/init.d/masq       2006-02-02 13:14:13.000000000 +1100
@@ -340,9 +340,7 @@
     /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 389 \
        --destination $OUTERNET --jump denylog

-    # myservice: TCPPort 4321, AllowHosts: 0.0.0.0/0, DenyHosts:
-    /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 4321 \
-       --destination $OUTERNET --src 0.0.0.0/0 --jump ACCEPT
+    # myservice: TCPPort 4321, AllowHosts: , DenyHosts:
     /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 4321 \
        --destination $OUTERNET --jump denylog
```

The output above is the differences between two copies of the firewall rules - the first (marked with minus signs) is when `myservice` was set to `public`. The second (marked with plus signs) is when the service was set to `private`. The important change is from `--jump ACCEPT` to `--jump denylog`.

Restricting services to specific external hosts: AllowHosts and DenyHosts

As well as being set to public and private, it is possible to allow or deny remote machines access to a particular service. Let's make the service `public` once more, but limit access to one host and one subnet:

```
config setprop myservice access public

config setprop myservice AllowHosts 1.2.3.4,100.100.100.0/24

signal-event remotaccess-update

diff -u /etc/rc.d/init.d/masq /tmp/masq
```

which should produce output something like this:

```
[root@gsxdev1 esmith]# diff -u /tmp/masq /etc/rc.d/init.d/masq
--- /tmp/masq   2006-02-02 13:14:09.000000000 +1100
+++ /etc/rc.d/init.d/masq       2006-02-02 13:22:32.000000000 +1100
```

```
@@ -340,9 +340,11 @@
     /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 389 \
        --destination $OUTERNET --jump denylog

-    # myservice: TCPPort 4321, AllowHosts: 0.0.0.0/0, DenyHosts:
+    # myservice: TCPPort 4321, AllowHosts: 1.2.3.4,100.100.100.0/24, DenyHosts:
     /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 4321 \
-       --destination $OUTERNET --src 0.0.0.0/0 --jump ACCEPT
+       --destination $OUTERNET --src 1.2.3.4 --jump ACCEPT
+    /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 4321 \
+       --destination $OUTERNET --src 100.100.100.0/24 --jump ACCEPT
     /sbin/iptables -A $NEW_InboundTCP --proto tcp --dport 4321 \
        --destination $OUTERNET --jump denylog
```

The firewall rulesets are automatically changed so that instead of allowing access from all hosts `0.0.0.0/0`, they now two specific rules to allow the host and network specified, and a new `--jump denylog` rule which blocks and logs any others.

> **Note:** Hosts which are not listed in `AllowHosts` are denied, if this property is configured.

There is also a `DenyHosts` property which generates rules to block specific hosts, if this is required. If there is a specific reason to limit access to a service, you should normally use `AllowHosts` to list the ones which do require access. The DenyHosts property is provided for completeness and can be useful in specific situations, for example to block mail from a misbehaving mail server while allowing it from all other sites.

Starting up programs automatically upon system boot

If your package implements a server or daemon, you will probably want it to be started automatically when the system boots.

The SME Server boots in runlevel 7, so you can get an idea of the startup processes by listing the contents of `/etc/rc.d/rc7.d`.

These are similar to the init scripts you may be familiar with from other Linux systems, with one important differnce. Instead of pointing to scripts within `/etc/rc.d/init.d`, all of those init entries are links to `/etc/rc.d/init.d/e-smith-service`. This is a wrapper which checks the configuration database to see if the service is supposed to be running and if so, starts the service from `/etc/rc.d/init.d/whatever`.

So for example, you might have:

```
S90squid -> /etc/rc.d/init.d/e-smith-service
```

The e-smith-service script looks up the name it was invoked with (`S90squid`), drops the prefix (leaving `squid`), checks the configuration database for the "squid" service, then if it's supposed to run, does:

```
/etc/rc.d/init.d/squid start
```

Here is the step-by-step procedure for making your package start up a program called `myserver` at boot time.

1. First, create the traditional init script `/etc/rc.d/init.d/myserver` which can be run with the command-line arguments "start" or "stop" to perform the appropriate action on the server. Look at other init scripts in the same directory for examples. This script should be included in your RPM.

   Note: If your service is managed by runit, all you need is a link to the **daemontools** script.

2. Second, create a symbolic link as shown below, choosing the two-digit number after the letter S to control the startup order of the server programs. Include this symbolic link in your RPM.

   ```
   /etc/rc.d/rc7.d/S55myserver -> /etc/rc.d/init.d/e-smith-service
   ```

   These two steps are typical for any Linux/Unix server application, except that the symbolic link traditionally points directly to the init script, rather than to `e-smith-service`. Remember, we want to link to `e-smith-service` to ensure that a `disabled` service does *not* start at boot time.

3. Third, let's assume for now that `myserver` should be enabled by default, and so start at boot time. You just need to create some properties in the configuration database to make that happen:

   ```
   cd /etc/e-smith/db/configuration/defaults
   mkdir myserver
   cd myserver

   echo service >type
   echo enabled >status
   ```

   For testing, you will also need to run **initialize-default-databases** to load these new defaults.

4. Your RPM can also start the service in the `%post` section, but you need to be very careful to only do this in run-level 7. The same `%post` section is run during installation from CDROM, and we do not want services started during that installation. They will most likely fail and may cause the CD install to fail.

All of these steps result in the server starting automatically upon installation of the RPM, and whenever the server is rebooted.

Care should also be taken for the RPM to uninstall cleanly. The service should be stopped and marked not to be restarted and so your RPM should contain the following lines in the *pre-uninstallation* script:

```
%preun
/sbin/e-smith/db configuration setprop myserver status disabled
/etc/rc7.d/S55myserver stop
true
```

The `/service/myserver` symbolic link is owned by the RPM, and when it is removed, **runsvdir** will soon notice and kill the **runsv** supervision process. The final **true** command ensures that a failure from the other commands won't cause the removal of the RPM to fail. Note that these steps cannot be in the post-uninstallation script, since some of the required files may have been removed by that time.

V. Documentation and resources

Chapter 16. Perl modules

If you are not already familiar with the Perl programming language, you will need to read up on at least the basics. One online course is available from http://sourceforge.net/projects/spork.

The SME Server has a wealth of Perl libraries to perform common functions including manipulating the configuration database, performing common CGI tasks, etc. The modules available include:

- `esmith::ConfigDB`
- `esmith::AccountsDB`
- `esmith::NetworksDB`
- `esmith::HostsDB`
- `esmith::NetworkServicesDB`
- `esmith::DB`
- `esmith::FormMagick`
- `CGI::FormMagick`
- `Text::Template`

The following libraries are also installed for compatibility with older code, but they should no longer be used. *These libraries may be removed in future releases.*

- `esmith::util` (deprecated)
- `esmith::db` (deprecated)
- `esmith::cgi` (deprecated)

The documentation can be accessed from the Linux command line on your SME Server by typing **perldoc esmith::ConfigDB** (or whatever module name you're interested in).

More information about building RPMs can be found at http://www.rpm.org/RPM-HOWTO/build.html. This is especially recommended if you wish to use anything more than the extremely simple outline given above. For instance, you may wish to to build RPMs using original source and patches or include more detail and functionality in your spec file.

VI. License texts

Appendix A. GNU Free Documentation License

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

GNU FDL Modification Conditions

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the

also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Sample Invariant Sections list

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

Sample Invariant Sections list

Appendix B. GNU General Public License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING,

DISTRIBUTION AND MODIFICATION

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a “work based on the Program ” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification ”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

Exception:: If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE