Scmbug manual

RELEASE_0-26-15

The Scmbug Team

Scmbug manual: RELEASE_0-26-15

by The Scmbug Team

Published 2009-05-12

This is the documentation of Scmbug, a system that integrates software configuration management with bug-tracking.

Table of Contents

List of Figures

Chapter 1. About

1.1. Copyright Information

1.2. Disclaimer

No liability for the contents of this document can be accepted. Follow the instructions herein at your own risk.

1.3. Acknowledgments

The excellent *Bugzilla Guide* (http://www.bugzilla.org/docs/tip/html) served as an example in preparation of this document. Content and structural document elements were reused from this guide. The *LDP Author Guide* (http://www.tldp.org/LDP/LDP-Author-Guide/html/index.html) also served as a good example.

Development of this system benefited from invaluable insight on SCM and bug-tracking issues, along with feedback, from John C. Quillan, Mark S. Reibert, Dave Swegen, the Bugzilla developers, and the Subversion developers.

1.4. Document Conventions

This document uses the following conventions:

Descriptions	Appearance
Warning	**Caution** Don’t run with scissors!
Hint	**Tip:** Would you like a breath mint?
Note	**Note:** Dear John...
Information requiring special attention	**Warning** Read this or the cat gets it.
File or directory name	`filename`
Command to be typed	**command**
Application name	application
Normal user’s prompt under bash shell	bash$
Root user’s prompt under bash shell	bash#
Environment variables	VARIABLE
Term found in the glossary	*cvs2cl*

Code example	`<para>`
	`Beginning and end of paragraph`
	`</para>`

This documentation is maintained in DocBook 4.2 SGML format.

Chapter 2. Introduction

2.1. What Is It?

Scmbug is a system that integrates software configuration management with bug-tracking. It is implemented in Perl and has been successfully deployed on UNIX-like and Windows systems. It is pronounced *Skamm-bag*.

2.2. Why Use An SCM System?

Those who do not use a software configuration management (SCM) system do not maintain a history of modifications performed to their software. When bugs creep in their software, they do not have adequate information on how changes to source code came about.

SCM systems, or even simple source code version control systems, make sure that a record of all changes and enhancements to the software is maintained. They provide a method of creating, storing, and labeling software changes.

2.3. Why Use A Bug-tracking System?

Those who do not use a bug-tracking system tend to rely on shared lists, email, spreadsheets and/or Post-It notes to monitor the status of defects. This procedure is usually error-prone and tends to cause those bugs judged least significant by developers to be dropped or ignored.

Integrated defect-tracking systems make sure that nothing gets swept under the carpet; they provide a method of creating, storing, arranging and processing defect reports and enhancement requests.

2.4. Why Integrate SCM With Bug-tracking?

SCM systems maintain software changes. Bug-tracking systems maintain lists of software enhancements and defects. By examining a log of software changes, it is uncertain *why* the changes occurred. By examining a log of defect reports, it is uncertain *what* changed in software in response to the defects.

Integration of SCM with bug-tracking ties the reason *why* a feature/defect was developed/fixed with *what* software changes occurred in the SCM system to accomplish this.

Chapter 3. Design

3.1. Goals

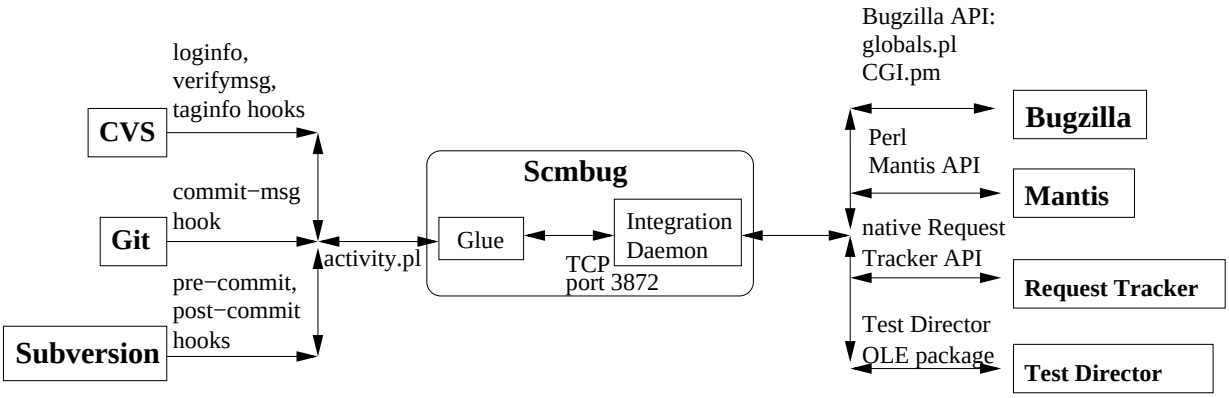
The goals of Scmbug are:

- **To solve the integration problem once and for all**.
- Provide synchronous verification checks of SCM actions against the bug-tracking system.
- Provide a flexible architecture that can be deployed across the public Internet with reasonable security. Permit integrating repositories hosted by multiple SCM systems in separate networks, against a single, publicly accessible bug-tracking system, for use with multiple mobile developers.
- Define an SCM to bug-tracking integration interface and mechanism that will permit integration of any SCM system with any bug-tracking system.
- Integrate most popular free SCM systems, such as CVS, Subversion, and Arch.
- Integrate most popular free bug-tracking systems, such as Bugzilla and Mantis.

3.2. System Architecture

Scmbug is developed as a client/server solution. As shown in Figure 3-1, it consists of a set of SCM system hooks that capture standard SCM events, a generic mechanism of handling these events on the machine hosting an SCM repository, a server daemon onto which integration requests corresponding to these events are transported, a generic mechanism of handling these requests, and functionality that can process these requests per bug-tracking system.

Figure 3-1. System architecture.



When various SCM events occur, such as committing software changes or labeling a repository, they are captured using hooks installed by Scmbug. Before the event's activity is allowed to proceed, various verification checks are performed as described in Section 4.4. These checks are synchronous; when an error is detected, the event's activity is stopped.

3.3. Related Systems

Scmbug is not original work. Most of the integration features separating it from related systems were first conceived and implemented by John C. Quillan for CVS and Bugzilla. The distinguishing features of his work are the synchronous nature of the verification checks, and a *VDD Generator* tool similar to the one described in Section 7.2. His work was never publicly released. Scmbug attempts to continue a redesigned, clean-room implementation of that work, supporting a wider variety of SCM and bug-tracking systems.

Other systems that integrate SCM with bug-tracking are described below. Along with Quillan's work, these solutions are unsuitable for certain development environments.

3.3.1. Bugzilla Email Gateway

SCM systems can integrate their actions with Bugzilla using the Bugzilla email gateway. Hook scripts installed in an SCM system can email the results of the system's actions to an email account configured to parse the email and process it accordingly. An example of such a configuration is available from *Steve McIntyre's* (http://www.einval.com/~steve/software/cvs-bugzilla/) web page.

This approach is not synchronous. For example, if a user accidentally commits against the wrong bug number, or a bug against which he is not the owner, the SCM system will proceed with the commit action regardless, without giving the user the option to correct his actions. Additionally, if the email gateway is not active, the developer will not be immediately aware that integration failed.

3.3.2. CVSZilla

Tony Garnock-Jones' *CVSZilla* (http://homepages.kcbbs.gen.nz/~tonyg/) integrates SCM events produced by CVS with Bugzilla. It also extends CVS to give rudimentary support to "change-sets", or "transactions".

CVSZilla does not support integration of events produced by any SCM system in a generic way. It modifies the Bugzilla schema and does not work with future versions of Bugzilla. Finally, it assumes that the TCP port used by MySQL is accessible from the machine hosting the CVS repository.

The last assumption does not always hold true, especially when a bug-tracking system is used to track development of mobile developers (laptop users), needing the flexibility to work both locally and remotely, on multiple projects, across different organizational units. Experience has shown that this is a common scenario in organizations with limited hardware, software, or labor resources, such as an academic environment.

For example, imagine integrating CVS actions from a repository hosted on a personal laptop with a Bugzilla instance that needs to be publicly accessible from the Internet. This same bug-tracking system is used for collaboration with other mobile developers of a different organization on a joint project hosted on a third machine. Integration over the public Internet is not possible without opening the MySQL database port Bugzilla uses. Opening this port is unnecessarily insecure, exposing access to other applications hosted on the same database system. The Scmbug daemon exposes the minimum required integration interface between an SCM system and a bug-tracking system.

3.3.3. Other Integration Systems

Commercial products concentrate only on integrating a particular SCM system with a particular bug-tracking system, in a proprietary way. They do not attempt to define a public SCM to bug-tracking integration interface, and solve the integration problem once and for all.

Scmbug is an effort for defining an SCM to bug-tracking integration interface and mechanism that will permit integration of any SCM system with any bug-tracking system.

Chapter 4. Features

4.1. Integration Actions

For every SCM event, multiple corresponding integration actions may be issued from the glue to the daemon. The list of possible integration actions is:

- `activity_verify`: An `activity_commit` integration action will soon follow, and data collected from the corresponding SCM event must be verified first.
- `activity_commit`: A software change has already been committed in the SCM system, and this event must be integrated with bug-tracking. This action usually follows an `activity_verify` integration action.
- `activity_tag`: A software labeling operation is issued in the SCM system, and this event must be integrated with bug-tracking.
- `activity_get_vdd`: A VDD will be produced by the *VDD Generator*.
- `activity_get_bugs`: The information of a list of bugs is returned for processing by the *Merger*.

4.2. Integration Of SCM Log Messages

When a software change occurs in the SCM system, a log message may be manually supplied by a user to describe the change. Using an `activity_commit` integration request, this log message is automatically inserted in the bug-tracking system against the specified bug id. Additional changeset information are automatically detected, such as the branch name that hosted the change and the list of files affected by the changeset. This information complements the users original log message when added in the bug-tracker, as shown in Figure 4-1.

Figure 4-1. Example of Integrated Log Message.

```
Added a glossary, and a document conventions section, just
like the Bugzilla Guide uses. Began linking names of systems/tools to
the glossary, and using a url to their webpage there. This will
unclutter the pdf version of the document from url links inlined with
content.


Branch:         HEAD
Affected files:
---------------
1.7 --> 1.8 Scmbug:system/doc/manual/about.sgml
1.7 --> 1.8 Scmbug:system/doc/manual/bugtracking-backends.sgml
NONE --> 1.1 Scmbug:system/doc/manual/conventions.sgml
NONE --> 1.1 Scmbug:system/doc/manual/glossary.sgml
1.12 --> 1.13 Scmbug:system/doc/manual/manual.sgml.in
1.8 --> 1.9 Scmbug:system/doc/manual/scm-frontends.sgml
1.9 --> 1.10 Scmbug:system/doc/manual/tools.sgml
```

4.3. Integration Of SCM Labeling Operations

When a tag or branch operation is issued in the SCM system, a name is used to describe this software labeling operation. Using an `activity_tag` integration request, this label is inserted in the list of possible versions of the product in the bug-tracking system.

Figure 4-2 shows an example applying a label in Subversion. The label name is autodetected and does not need to be specified in the log message.

Figure 4-2. Example applying a label in Subversion.

```
$ svn copy trunk tags/MYPROGRAM_RELEASE_4-6-1
$ svn commit -m "bug 30541: Tagged the version that fixes the \
  backwards compatibility problems in the communication \
  protocol." tags/MYPROGRAM_RELEASE_4-6-1
```

4.4. Verification Checks

When various SCM events occur, verification checks are performed before the SCM event is allowed to proceed. Some checks are performed by the glue on the client side, the machine the SCM repository resides. Other checks are performed on the server side, the machine hosting the Scmbug daemon.

4.4.1. Enabled Integration

SCM events will be integrated with bug-tracking only if the integration glue is enabled. This can be controlled in the glue configuration file using the `enabled` variable, as shown in Figure 4-3.

Figure 4-3. Glue enabling variable.

```
#
# Flags whether the glue is active
#
enabled => 1,
```

4.4.2. Supported SCM version

SCM tools may work considerably different between separate versions. For example, CVS changed during the 1.12.x series the format of command-line arguments supplied to integration hooks. Appropriate support is needed to handle their idiosynchracies. The SCM tool's version is detected at runtime, and interaction with the tool is handled accordingly.

One of the problems with some SCM systems is that they may unset the `PATH` variable (e.g. Subversion in 1.2.x). The list of paths to the tool's binaries must be supplied in the glue configuration file using the `binary_paths` variable, as shown in Figure 4-4. It is verified at runtime that each binary needed by an SCM tool is present in only one path from the list supplied, to avoid accidentally invoking the wrong version of the tool due to an incomplete installation of the SCM tool.

Figure 4-4. Paths to the SCM tool's binaries.

```
# Comma(,)-separated list of paths to any binaries the SCM
# tool may need to use
binary_paths => '/usr/local/bin,/usr/bin,/bin',
```

4.4.3. Use Of A Log Message Template

The log message supplied to the SCM system when committing a software change is verified to match a log template expected by Scmbug. Two regular expressions describe how the bug id and log comment will be identified. These are defined as part of the `log_template` policy variable as shown in Figure 4-5, through the variables `log_bugid_regex` and `log_body_regex`.

A way to split a list of multiple bug ids into separate ids is also described with a regular expression through the variable `log_bugid_split_regex`. This is needed in order to permit special characters to preceed a bug number. For example, instead of separating bug ids using a whitespace or comma, one may want to also prefix a bug id with a '#'. Bug-trackers may then autolinkify in their comments a bug id that is prefixed by a '#' (e.g. Bugzilla). An example log message accepted by these expressions is shown in Figure 4-6.

Figure 4-5. Regular expressions describing the bug id, the split of bug ids and the log message body.

```
log_template =>  {
    # The log_bugid_regex is a regular expression that must
    # set the unnamed variable $1 to the bug number, or list
    # of bug numbers. It is checked for a match as: m/$regex/s
    log_bugid_regex => '^\s*bug\s*([\d|\s|,|#]*?):',
    # The log_bugid_split_regex is a regular expression
    # describing how a list of bug ids will be split in
    # individual bug numbers. It is split as: /$regex/
    log_bugid_split_regex => ',\s?#|\s?#|,|\s+',
```

```
    # The log_body_regex is a regular expression that must set
    # the unnamed variable $1 to the log comment. It is
    # checked for a match as: m/$regex/s
    log_body_regex => '^\s*bug.*?:\s*(.*)'
    },
```

This template can be customized when the Scmbug codebase is configured, prior to installation. The arguments `--with-log-template-bugid-regex=<regular_expression>`, `--with-log-template-bugid-split-regex=<regular_expression>`, and `--with-log-template-body-regex=<regular_expression>` can be passed to `configure`, shown in Figure 8-4.

Figure 4-6. Example log message accepted.

```
bug 441:Improved the documentation of policy log_template by adding
an example log message.
```

4.4.4. Presence Of Distinct Bug Ids

The log message supplied to the SCM system when committing a software change is verified to include only distinct bug ids.

4.4.5. Valid Log Message Size

It is verified that the log message supplied to the SCM system system meets a configurable minimum log message size limit. This behavior is defined in the glue configuration file using the `minimum_log_message_size` policy variable, as shown in Figure 4-7.

Figure 4-7. Minimum log message size policy.

```
# Minimum number of characters log message.
minimum_log_message_size => {
    enabled => 1,
    size => 50
    },
```

4.4.6. Convention-based Labeling

It is verified that the names used in labeling operations, such as creation of tags or branches, match a configurable label naming convention. This behavior is defined in the glue configuration file using the `label_name` policy variable, as shown in Figure 4-8.

Figure 4-8. Label naming convention policy.

```
# Format of label names (tag or branch names) defined as
# regular expressions.
label_name => {
    enabled => 1,
    names => [
      # Convention for official releases.
      # For example:
      # SCMBUG_RELEASE_0-2-7
      '^.+?_RELEASE_[0-9]+-[0-9]+-[0-9]+$',

      # Convention for development builds.
      # For example:
      # SCMBUG_BUILD_28_added_a_policies_mechanism
      '^.+?_BUILD_[0-9]+_.+$',

      # Convention for branches.
      # For example:
      # b_experimenting_with_policies_on_glue_side
      '^b_.+$',

      # Convention for private developer tags. Uses
      # the developer's initials (either 2 or 3).
      # For example:
      # p_kpm_prior_to_bug353_stabilization_fixes
      '^p_[a-zA-Z][a-zA-Z]?[a-zA-Z]_.+$'
      ]
  }
```

Labels for official releases can correspond to versions of the software that will be made public to users, and are applied by release managers.

Developments builds can be helpful in development environments that produce weekly builds, as a checkpoint of a stable codebase. They are applied by release managers. They can also correspond to development milestones that will incorporate specific features, even though the codebase is not ready for an official release.

Branches can be created when maintaining a previous, already released, version of a software. For example, to correct critical bugs or apply security fixes. They are applied by release managers. They can also be created when a feature may require a considerable amount of time to implement while disrupting

the main codebase drastically (e.g. a core API change). A branch is created, the feature is implemented, and when it is deemed stable it is merged in the main codebase.

Private developer tags are very similar to development builds. They can be applied directly by developers, instead of strictly release managers, and should be thought of as personal, developer milestones applied at stable points. The developers may be working on features of increased complexity, or features that will require a significant amount of time to complete. Incrementally applying private developer tags that match distinct progress steps can help a developer debug a regression in his implementation. As another example, assume a developer implements a feature using a specific algorithm. Later, software requirements change, and the developer is tasked to reimplement this feature using a different algorithm. The developer can first apply a private tag, and then reimplement the feature using the new algorithm. If it is later determined that the feature should revert back to using the original algorithm, the developer can retrieve the original implementation using the private developer tag he had applied. In a sense, private developer tags can be applied at a finer granularity than group development builds.

4.4.7. Presence Of Bug Ids

The log message supplied to the SCM system when committing a software change may be required to include at least one bug id. This is determined using the `presence_of_bug_ids` policy variable, as shown in Figure 4-9.

Figure 4-9. Presence of bug ids policy.

```
#
# Presence of bug ids. There are 3 options:
#
# - 'required'. A bug id must be specified during each
#   activity. Activities without a bug id will not be permitted.
#
# - 'optional'. If a bug id is supplied, the activity will be
#   integrated. If not the activity will be permitted to go
#   through in the SCM system, but without bug-tracking
#   integration.
#
# - 'none'. Never integrate activities regardless. This is
#   different than flagging the glue inactive. The remaining
#   policies are still enforced were applicable.
#   (e.g. policy minimum_log_message_size).
#
# This policy is ALWAYS enabled
presence_of_bug_ids =>  {
    value => 'required'
    },
```

4.4.8. Valid SCM To Bug-tracking Username Mapping

All integration requests must include the SCM username of the user issuing an integration request. This username must be mapped to the username of the user in the bug-tracking system. Bug-tracking systems that do not support SCM usernames are accomodated through a username mapping list defined in the daemon configuration file using the `userlist` variable. Three mapping mechanisms are available:

- **LDAP**. An LDAP directory is accessed to map the SCM username, as one LDAP attribute, into the bug-tracking username defined by another LDAP attribute. This can be configured in the daemon configuration file using the `mapping_ldap` variable, as shown in Figure 4-10.

Figure 4-10. SCM to bug-tracking username mapping based on `mapping_ldap`.

```
# This is a mapping based on LDAP. ldap_scm_username_attribute
# defines the LDAP attribute that will be used to match the
# SCM username. The SCM username will be mapped into the
# bug-tracking username defined by
# ldap_bugtracking_username_attribute.
mapping_ldap => {
    enabled => 0,
    ldap_server => '127.0.0.1',
    ldap_port => '389',
    # A binddn (e.g. cn=default) that has access to read all
    # attributes
    ldap_binddn => 'replace_with_binddn',
    # The password of the binddn that has access to read all
    # attributes
    ldap_binddn_password => 'replace_with_binddn_password',
    # The BaseDN in which to search for the
    # ldap_scm_username_attribute (e.g. "ou=People,o=Company")
    ldap_basedn => 'replace_with_basedn',
    # The name of the attribute containing the user's SCM
    # username
    ldap_scm_username_attribute => 'uid',
    # The name of the attribute containing the user's
    # bug-tracking username
    ldap_bugtracking_username_attribute => 'mail',
    # LDAP filter to AND with the ldap_scm_username_attribute
    # for filtering the list of valid SCM users.
    ldap_filter => ''
},
```

- **List of regular expressions**. A list of regular expressions describe how the SCM username will be matched and how it will be transformed into a bug-tracking username using the `mapping_regexes` variable. This can be configured in the daemon configuration file as shown in Figure 4-11.

Figure 4-11. SCM to bug-tracking username mapping based on `mapping_regexes`.

```
# This is a mapping based on regular expressions. The first
# expression defines how the SCM username will be matched. The
# second defines how it will be transformed, and uses the
# unnamed variable $1 that was described by the first
# expression. The mapping is checked for a match as:
# m/$first_regex/
# and is applied as: s/$first_regex/$second_regex/
mapping_regexes => {
    enabled => 0,
    values => {
    # This is an example of mapping a Windows Domain user
    # from 'DOMAIN\user' to 'user@EMAIL_DOMAIN.com'
    '^DOMAIN\\\\(\w+)$' => '$1\@EMAIL_DOMAIN.com',
    # This is an example of mapping a UNIX user from
    # 'example_user' to 'example_user@exampledomain.com'
    '^(\w+)$' => '$1\@exampledomain.com'
    }
},
```

- **Direct**. A direct one-to-one mapping of an SCM username to a bug-tracking username using the `mapping_values` variable. This can be configured in the daemon configuration file as shown in Figure 4-12.

Figure 4-12. SCM to bug-tracking username mapping based on `mapping_values`.

```
# This is a one-to-one mapping of SCM usernames to bugtracking
# usernames. Mappings in this list override mappings from
# mapping_regexes.
mapping_values => {
    enabled => 0,
    values => {
    'DOMAIN\\example_user' => 'example_user@DOMAIN.com',
    'example_user2' => 'example_user2@exampledomain.com'
    }
},
```

If the SCM username already matches the bug-tracking username, these mappings can be disabled in the daemon configuration file as shown in Figure 4-13. If any of these mappings are enabled, they are executed in the order presented here. Mappings based on `mapping_ldap` are appiled first. Mappings based on `mapping_regexes` are applied second and can override a mapping based on `mapping_ldap`. Mappings based on `mapping_values` are applied last and can override all other mappings.

Figure 4-13. Disabling SCM to bug-tracking username mappings.

```
mappings => {
    # Enable SCM username translation. This flag must be
    # turned on for any of the mappings that follow to apply.
```

```
    enabled => 0,
```

The username verification is applied case sensitive. This can be configured in the daemon configuration file as shown in Figure 4-14. For example, Microsoft Active Directory tends to capitalize the first letter of each word in the email address. The email address returned does not match the email address reported by the bug-tracker in lowercase, and needs `case_sensitive_username_verification` to be disabled.

Figure 4-14. Disabling case sensitive SCM to bug-tracking username verification.

```
# Apply a case sensitive username verification.
case_sensitive_username_verification => 0,
```

4.4.9. Valid Product Name

An `activity_verify` integration request must refer to bug ids filed against the SCM system's associated product name in the bug-tracking system. Similarly, an `activity_tag` needs to know the product name in which the tag will be inserted as a version. This behavior is optional and can be configured in the glue configuration file using the `valid_product_name` policy variable, as shown in Figure 4-15.

Figure 4-15. Valid product name policy.

```
# The bug against which an activity is issued must be filed
# against a valid product name.
valid_product_name => {
    enabled => 1
    },
```

Regardless of the configuration of the `valid_product_name` policy, the product name is required by other parts of the integration. For example, it is needed to detect labeling operations in Subversion, as shown in Figure 7-7, for *Mail notification* and for *Autolinkification*. It can be specified in the glue configuration file using the `product_name_definition` policy variable in two ways:

- **Manually defined**. All SCM activity in the repository will be integrated against a single product name in the bug-tracker. This requires setting in the `product_name_definition` policy's `values` the entry of the special regular expression `(.*)` to the desired product name, as shown in Figure 4-16.

  Figure 4-16. Manually defined product name.

  ```
  # Product name definition.
  #
  # NOTE: The regular expression '(.*)' is special and means
  ```

```
#       replace with exactly this value. It is meant to be used
#       with SCM systems that do not provide the list of
#       affected files during verification (e.g. CVS 1.11.x)
#
# This policy is ALWAYS enabled
product_name_definition => {
    type => 'manual',
    values => { '(.*)' => 'TestProduct' }
    },
```

- **Automatically defined**. Some organizations may follow a development model that permits multiple products to be hosted under the same SCM repository. For example, multiple product names in the bug-tracking system may correspond to multiple branches in the SCM system. Scmbug can autodetect the appropriate product name by consulting a list of regular expressions that describe how to identify and map the product name from a path in a repository.

Figure 4-17 shows a configuration example that can autodetect from the repository structure of Figure 4-18 the list of product names in Figure 4-19. When a file from one of the directories shown in Figure 4-18 is commited, the appropriate product name shown in Figure 4-19 will be defined.

Figure 4-17. Automatically defined product name.

```
# Product name definition.
#
# The product name is autodefined based on regular
# expressions. The first expression defines how each commited
# filename will be matched. The second defines how it will be
# transformed, and uses the unnamed variables (e.g. $1 $2)
# that were described by the first expression. The mapping is
# checked for a match as:
# m/$match_regex/
# and is applied as: s/$match_regex/$replace_regex/
#
# This policy is ALWAYS enabled
product_name_definition => {
    type => 'auto',
    values => { 'dir/prefix1/productMain_(.+?)/' => '$1',
                'dir2/(.+?)/' => '$1',
                'dir3/(?:trunk|tags|branches)/(.+?)/' => '$1',
                'dir4/productLine_(.*)/subproducts/(.*)/' => '$2_$1'
              }
    },
```

Figure 4-18. Repository structure with product names that can be automatically defined.

```
dir/prefix1/productMain_subproductA/trunk
dir/prefix1/productMain_subproductA/tags
dir/prefix1/productMain_subproductA/branches
```

```
dir/prefix1/productMain_subproductB/trunk
dir/prefix1/productMain_subproductB/tags
dir/prefix1/productMain_subproductB/branches
dir2/productA_3-1/trunk
dir2/productA_3-1/tags
dir2/productA_3-1/branches
dir2/productB
dir2/productB_multiuser
dir3/trunk/productC
dir3/tags/productC
dir3/branches/productC
dir4/productLine_2009/subproducts/subproductC
```

Figure 4-19. Automatically mapped product names from Figure 4-18.

```
subproductA
subproductB
productA_3-1
productB
productB_multiuser
productC
subproductC_2009
```

Another example where multiple products may be required would be a contracting company maintaining all their contracts in the same SCM repository but using separate product names in the bug-tracking tool.

> **Note:** We must note that, from an SCM perspective, hosting multiple products that share a common codebase in the same SCM system may not be the ideal way to go. Organizations that follow this development model may want to consider developing their common codebase in it's own SCM repository, as a separate product. They can then import the common code as a vendor branch in multiple SCM repositories, each corresponding to a single product name they wish to publicly release. More information on vendor branches can be found in the CVS (https://www.cvshome.org/docs/manual/cvs-1.12.9/cvs_13.html#SEC104) and Subversion (http://svnbook.red-bean.com/svnbook/ch07s04.html) manuals.

> **Note:** Automatically defined product names are not supported for CVS 1.11.x (http://bugzilla.mkgnu.net/show_bug.cgi?id=746) or for Git, yet (http://bugzilla.mkgnu.net/show_bug.cgi?id=994).

4.4.10. Valid Bug Owner

It is verified that the SCM user issuing an `activity_verify` integration request is the owner or one of the owners of the bug against which subsequent integration requests will be issued. This behavior is optional and can be configured in the glue configuration file using the `valid_bug_owner` policy variable, as shown in Figure 4-20.

Figure 4-20. Valid bug owner policy.

```
# The SCM user issuing an activity must be the user to which
# the bug is assigned
valid_bug_owner => {
    enabled => 1,
},
```

4.4.11. Anonymous SCM Username

It is always verified that the SCM system supplied a username when generating activity. However, some SCM systems may not always supply an SCM username. One example is Subversion running an `svnserve` daemon granting anonymous access. The username of the SCM user under which activity should be generated can be optionally configured in the glue configuration file using the `anonymous_scm_username` policy variable, as shown in Figure 4-21.

Figure 4-21. Anonymous SCM username policy.

```
# All integration activity must originate from a specific SCM
# user. If the SCM system does not provide the SCM user
# information (e.g Subversion running an svnserve daemon with
# anonymous access), assume the activity originated from a
# specific SCM user
anonymous_scm_username => {
    enabled => 0,
    value => 'anonymous_scm_user'
    },
```

4.4.12. Open Bug State

It is verified that the bug against which an `activity_verify` integration request is issued must is in an open, active state in the bug-tracking system. This behavior is optional and can be configured in the glue configuration file using the `open_bug_state` policy variable, as shown in Figure 4-22.

Figure 4-22. Open bug state policy.

```
# The bug against which an activity is issued must be in an
# open state
open_bug_state => {
    enabled => 1
    },
```

4.5. Additional Features

4.5.1. Mail Notification

An email can be sent when an integration activity, either commiting or labeling, is accepted. This is defined in the glue configuration file using the `mail_on_success` policy variable, as shown in Figure 4-23. Emails can also be sent after a failed commit activity if the SCM system overshadows and does not report the error message using the `mail_on_failure` policy variable.

> **Warning**
>
> For example, Subversion does not report error messages of its `post-commit` hook.

Figure 4-23. Mail notification policy.

```
#
# Send email notifications after integration activity
#
mail_notification => {
# Send an email after a successful activity (both
        # verifying and labeling)
mail_on_success => 0,
# Send an email after a failed commit activity that the
# SCM system may overshadow and not report
# (e.g. Subversion does not report error messages of its
# post-commit hook.) .
mail_on_failure => 1,
mail_settings => {
    # Must be a valid email address. Can remain empty if
    # other users should be notified.
    To => 'replace_with_commit_mailing_list_email@exampledomain.com',
    # Must be a valid email address. Can remain empty if
    # mail_also_appears_from_scm_user is enabled.
    From => 'Scmbug <replace_with_mailing_list_owner_email@exampledomain.com>',
    # Defaults to localhost if left empty
    Smtp => 'replace_with_mail_server.exampledomain.com'
```

```
},
# Sending email when a tag is moved or deleted in CVS can
# be annoying, since multiple emails are sent per
# directory (but not when a tag is added). mail_on_label
# can disable that behavior.
mail_on_label => 1,
mail_recipients => {
# Make the email also appear to have been sent by the
# SCM user.
mail_also_appears_from_scm_user => 1,
# List of users that will be notified
mail_scm_user => 1,
mail_bug_owner => 1,
mail_bug_reporter => 1,
mail_bug_monitors => 1,
mail_product_owners => 1
}
}
```

4.5.2. Bug Resolution

It is possible to automatically change the resolution status of a bug at the time a software change is committed. It is verified that the requested resolution status is a valid resolution state in the bug-tracker and that the requested change does not violate the workflow of the bug-tracker. For example, changing a bug resolution in Bugzilla from `REOPENED` to `UNCONFIRMED` is an invalid status change.

Changing resolution status is accomplished if a resolution template expected by Scmbug is identified in the log message. The resolution does not need to apply to the same bug the software change is applied against. It could be applied to a different bug, multiple bugs, or multiple different resolutions could be applied to multiple different bugs. The resolution template must be specified on a separate line on its own and is completely removed before the log message is identified.

Four regular expressions describe how the bug id, the new status, the status resolution, and the status resolution data will be identified. These are defined as part of the `resolution_template` policy variable as shown in Figure 4-24, through the variables `resolution_bugid_regex`, `resolution_status_regex`, `resolution_status_resolution_regex`, and `resolution_status_resolution_data_regex`.

A way to split a list of multiple bug ids into separate ids is also described with a regular expression through the variable `resolution_bugid_split_regex`. An example log message accepted by these expressions is shown in Figure 4-26.

Figure 4-24. Regular expressions describing the bug id, the split of bug ids and the resolution status.

```
# Resolution template.
#
# Regular expressions that describe how a resolution status
# for a list of bug ids can be identified
resolution_template => {
    enabled => 1,
    # The resolution_bugid_regex is a regular expression that
    # must set the unnamed variable $1 to the bug number, or
    # list of bug numbers. It is checked for a match as:
    # m/$regex/s
    resolution_bugid_regex => '^\s*status\s*([\d|\s|,|#]*?):',
    # The resolution_bugid_split_regex is a regular expression
    # describing how a list of bug ids will be split in
    # individual bug numbers. It is split as: /$regex/
    resolution_bugid_split_regex => ',\s?#|\s?#|,|\s+',
    # The resolution_status_regex is a regular expression that
    # must set the unnamed variable $1 to the requested
    # status. It is checked for a match as: m/$regex/s
    #
    # For example, if one issued in the log message the
    # resolution command:
    #
    # status 547: reopened
    #
    # Then the resolution_status_regex is expected to match
    # "reopened"
    resolution_status_regex => '^\s*status.*?:\s*(\S+)\s*.*',
    # The resolution_status_resolution_regex is a regular
    # expression that must set the unnamed variable $1 to the
    # requested resolution. It is checked for a match as:
    # m/$regex/s
    #
    # For example, if one issued in the log message the
    # resolution command:
    #
    # status 547: resolved fixed
    #
    # Then the resolution_status_resolution_regex is expected
    # to match "fixed"
    resolution_status_resolution_regex => '^\s*status.*?:\s*\S+\s+(\S+)'
    # The resolution_status_resolution_data_regex is a regular
    # expression that must set the unnamed variable $1 to the
    # additional data supplied by the resolution status. It is
    # checked for a match as:
    # m/$regex/s
    #
    # For example, if one issued in the log message the
    # resolution command:
    #
    # status 548: resolved duplicate 547
```

```
    #
    # Then the resolution_status_resolution_data_regex is
    # expected to match "547"
    resolution_status_resolution_data_regex => '^\s*status.*?:\s*\S+\s+\S+\s+(\S+)',
    },
```

Some bug-trackers may report resolution-related information with a token that contains spaces. For example, Mantis 1.0.0 offers the resolution "unable to reproduce". This would make it difficult to develop regular expressions that will correctly identify the new status and resolution. This resolution could instead be written in a log message by the user as "unable_to_reproduce" and have Scmbug configured to replace all underscores ("_") with spaces (" ") using the `resolution_status_convert` policy, as shown in Figure 4-25.

Figure 4-25. Regular expressions defining a resolution status character conversion.

```
    # The resolution_status_* information can have all of the
    # following characters converted according to a regular
    # expression. This is useful in addressing the limitation
    # of some bug-trackers that report a resolution-related
    # information with a token that contains spaces. For
    # example:
    #
    # "unable to reproduce" in Mantis.
resolution_template => {
    resolution_status_convert => {
enabled => 0,
# Regular expressions that will be applied to convert
# the characters of all resolution_status_*
# information. It is applied for substitution as:
#
# s/$convert_from/$convert_to/g
resolution_status_convert_from => '_',
resolution_status_convert_to => ' '
},
}
```

This template can be customized when the Scmbug codebase is configured, prior to installation. The arguments `--with-resolution-template-bugid-regex=<regular_expression>`, `--with-resolution-template-bugid-split-regex=<regular_expression>`, `--with-resolution-template-status-regex=<regular_expression>`, `--with-resolution-template-status-resolution-regex=<regular_expression>`, `--with-resolution-template-status-resolution-data-regex=<regular_expression>`, `--with-resolution-template-status-convert-from-regex=<regular_expression>`, and `--with-resolution-template-status-convert-to-regex=<regular_expression>` can be passed to `configure`, shown in Figure 8-4.

Figure 4-26. Example log message that changes the resolution status of multiple bugs.

```
status 548,622: reopened
status 755: resolved worksforme
bug 547:Implemented automatic status resolution as a new policy. This seems
to work but will need improvements in the testsuite.
status 547: REsolved fIXED
status 548: RESOLVED duplicate 547
status 647: assigned unassigned@mkgnu.net
```

Note: Figure 4-26 does not show multiple examples. It shows **one** example of **one** log message.

The resolution status and resolution descriptions are verified case insensitive by default, as shown in Figure 4-26. This can be controlled using the `resolution_status_case_sensitive_verification` variable, as shown in Figure 4-27.

Figure 4-27. Case sensitive resolution verification variable.

```
    # Apply a case sensitive resolution and resolution status verification
    resolution_status_case_sensitive_verification => 0,
```

A resolution description must refer to bug ids filed against the SCM system's associated product name in the bug-tracking system. This behavior is optional and can be configured in the glue configuration file using the `resolution_valid_product_name` policy variable, as shown in Figure 4-28.

Figure 4-28. Valid product name policy in reference to bug resolution.

```
resolution_template => {
    # The bugs whose resolution status will be changed must be
    # filed against a valid product name.
    resolution_valid_product_name => 1,
},
```

The SCM user issuing a bug resolution must be the owner of the bug against which subsequent integration requests will be issued. This behavior is optional and can be configured in the glue configuration file using the `resolution_valid_bug_owner` policy variable, as shown in Figure 4-29.

Figure 4-29. Valid bug owner policy in reference to bug resolution.

```
resolution_template => {
    # The SCM user must be the user to which the bugs whose
    # resolution status will be changed are assigned
    resolution_valid_bug_owner => 1
},
```

4.5.3. Autolinkification

There are plans (http://bugzilla.mkgnu.net/show_bug.cgi?id=266) to support autolinkification.

Chapter 5. SCM Frontends

5.1. CVS

CVS is the Concurrent Versions System, the dominant open-source network-transparent version control system.

Scmbug supports verification checks, integration of log messages with the bug-tracking system, and integration of labeling operations for CVS. Various deficiencies of CVS introduce complications in integration. It is recommended that users of CVS upgrade to *Subversion*.

CVS does not require users to enter a log comment when directories are added in a repository. Scmbug does not overcome this limitation, even though it is possible (http://bugzilla.mkgnu.net/show_bug.cgi?id=285) to do so.

Another limitation of CVS is that it does not provide an integration hook on '`cvs admin -o` (http://bugzilla.mkgnu.net/show_bug.cgi?id=176)'.

The verification hook of CVS 1.11.x does not provide the list of files that will be modified. As a result, the *Valid product name* policy cannot be set to `auto` for CVS 1.11.x. This policy will be implemented (http://bugzilla.mkgnu.net/show_bug.cgi?id=746) in the future for CVS 1.12.x and, if possible, for CVSNT.

The temporary log message file of CVS 1.11.x reports the directory in which a change is applied. However it does not report separately the repository path. As a result, the *Mail notification* policy reports the changeset directory only. This will be implemented (http://bugzilla.mkgnu.net/show_bug.cgi?id=826) in the future for CVS 1.12.x and CVSNT.

A common limitation of other systems integrating CVS with bug-tracking resulted from the inadequate mechanism CVS 1.11.x uses to provide the list of affected files in a commit trigger. For each file, the old version, new version, and the filename, all separated by commas, can be passed as command-line arguments to an integration trigger script. When the filenames, or the directory in which the files reside, contain either commas or whitespaces, a processing script using a single regular expression to parse these arguments will get confused. Some systems addressed this by requiring source modifications (http://www.einval.com/~steve/software/cvs-bugzilla/#loginfo) to the CVS binary, and distributed patches for it. Scmbug addresses this issue by employing a stateful parser (http://bugzilla.mkgnu.net/show_bug.cgi?id=286#c2). Using this parser, the probability of files and directories with whitespaces or commas contained in their names to confuse the integration glue is marginal. For example, the filename shown in Figure 5-1 is accepted.

Figure 5-1. A complex filename accepted by the CVS glue.

```
a file with spaces,NONEs, commas, digits,1.1,2.numbers close to dots.1.2,NONe.txt
```

The verification hook for CVS does not provide the list of filenames that will be commited, hence it is impossible to dynamically detect the product name. Automatically detected product names are not supported (http://bugzilla.mkgnu.net/show_bug.cgi?id=746) for CVS.

Scmbug has been verified to work against the following releases of CVS:

- *1.11.21*
- *1.12.13*
- *CVSNT 2.5.03 Build 2260*

Somewhere during the 1.12.x series of CVS (e.g. 1.12.9) the command line template format changed (https://www.cvshome.org/docs/manual/cvs-1.12.10/cvs_18.html#SEC186). This newer format is also (http://bugzilla.mkgnu.net/show_bug.cgi?id=464) supported by Scmbug.

5.2. Git

Git is a distributed source code management tool designed to handle massive projects such as the Linux kernel with speed and efficiency.

Scmbug supports verification checks and integration of log messages with the bug-tracking system for Git. Integration of labeling operations (http://bugzilla.mkgnu.net/show_bug.cgi?id=991) is not supported yet.

The integration may conservatively and incorrectly (http://bugzilla.mkgnu.net/show_bug.cgi?id=1253) detect multiple product names being defined if *Valid product name* is configured to be automatically detected as shown inFigure 4-17.

Scmbug has been verified to work against the following releases of Git:

- *1.5.6*

5.3. Subversion

Subversion is a compelling replacement for CVS.

Scmbug supports verification checks, integration of log messages with the bug-tracking system, and integration of labeling operations for Subversion.

Subversion is not yet (http://subversion.tigris.org/issues/show_bug.cgi?id=1973) capable of using a predefined log template when the user's editor is opened to enter a log comment. As a result, the log template expected by Scmbug must be entered by the user.

Scmbug has been verified to work against the following releases of Subversion:

- *1.0.6*
- *1.1.3*
- *1.2.3*
- *1.3.0*
- *1.4.0,1.4.5*
- *1.5.1*
- *1.6.1*

5.4. Other SCM Systems

Additional SCM frontends can be supported by Scmbug. Developers and system integrators of the following SCM systems are **strongly encouraged** to contribute an SCM integration frontend:

- *Aegis*
- *Arch*
- *Bazaar-NG*
- *Bitkeeper*
- *Clearcase*
- *Katie*
- *Mercurial*
- *Monotone*
- *OpenCM*
- *Perforce*

Developing a frontend requires:

- Committing to support this frontend in future releases of Scmbug.

- Creating a new frontend module named `src/lib/product/Glue/FrontendName.pm.in`. The CVS frontend `src/lib/product/Glue/CVS.pm.in` serves as a good example.
- Updating `src/lib/product/Glue/Glue.pm.in:check_configuration` accordingly.
- Updating the configuration management files `configure.in` and `Makefile.in` to autogenerate, and autocleanup the new frontend.
- Updating the `install-tools` rule of `Makefile.in` to install the new frontend from source.
- Creating a new directory named `src/glue/templates/frontendname` that includes template hook/trigger scripts. The CVS trigger scripts in `src/glue/templates/cvs` serve as a good example.
- Updating `src/glue/templates/cvs/*/checkoutlist.in` to always extract `lib/scmbug/Scmbug/Glue/FrontendName.pm` so the CVS SCM frontend does not break.
- Updating the *Glue Installer* `src/scripts/install_glue.pl.in` and its manpage `doc/manpages/install_glue.sgml.1.in` to support the new frontend.
- Updating the *VDD Generator* `src/scripts/vdd_generator.pl.in` and its manpage `doc/manpages/vdd_generator.sgml.1.in` to support the new frontend.
- Updating the *Merger* `src/scripts/merge.pl.in` and its manpage `doc/manpages/merge.sgml.1.in` to support the new frontend.
- Updating the documentation in `doc/manual/content` to reflect support for the new frontend.

Chapter 6. Bug-tracking Backends

6.1. Bugzilla

Bugzilla is an enterprise-class piece of software that tracks millions of bugs and issues for hundreds of organizations around the world.

Bugzilla does not provide a public interface for SCM integration. Nevertheless, the Scmbug daemon attempts to reuse functionality already available in the Perl-based Bugzilla libraries. As a result, the source code used to host a Bugzilla instance must be locally accessible by the daemon. The `installation_directory` variable is used to define the location of the Bugzilla source installation as shown in Figure 6-1.

Figure 6-1. Bug-tracker installation directory for Bugzilla.

```
# Used for Bugzilla and RequestTracker.
#
# Path to the directory of the bugtracker sources providing an
# API that the daemon can use
installation_directory => '/usr/share/bugzilla/lib'
```

Additionally, the `installed_locally` variable is used to flag whether Bugzilla is installed locally on the same machine the daemon is running as shown in Figure 6-2. This is required to flush Bugzilla's version cache when SCM tags are added or deleted, and immediately reflect tag manipulation to Bugzilla's user interface.

Figure 6-2. Bug-tracker installed locally variable.

```
# Used only for Bugzilla.
#
# Flags whether the live bugtracker instance is installed
# locally on the same machine the daemon is running
installed_locally => 1,
```

> **Note:** For example, it is possible to install Bugzilla in machine A, and install a duplicate Bugzilla source in machine B where Scmbug is installed. In this scenario `installation_directory` would be set to Bugzilla's source code path on machine B, but the variable `installed_locally` would be set to 0.

The Bugzilla developers are planning (http://bugzilla.mozilla.org/show_bug.cgi?id=254400) to provide a formal SCM integration interface in future releases.

As of version 2.22.0, Bugzilla does not yet support a mapping of the SCM username to a Bugzilla username. An SCM to bug-tracking username mapping is accomodated by the integration daemon using the `userlist` variable, as described in Section 4.4.8.

Bugzilla currently lacks a per-bug comment id (http://bugzilla.mkgnu.net/show_bug.cgi?id=763). This makes it very difficult to accurately report the comment ids in a VDD for Bugzilla. Currently, the VDD comment ids are prefixed with the string "`inaccurate_`".

Scmbug has been verified to work against the following releases of Bugzilla:

- *2.14.2*
- *2.16.5*
- *2.18rc2*
- *2.19.2*
- *2.20.1*
- *2.22.0*
- *3.0.0*

6.2. Mantis

Mantis is a web based PHP/MySQL-based bugtracking system.

Mantis does not provide a public interface for SCM integration. However, the Scmbug daemon includes a native Perl-based bug-tracking backend for integration with Mantis. Configuration of the variables `installation_directory`(Figure 6-1) and `installed_locally`(Figure 6-2) is not required.

Mantis supports installation with multiple database backends. The daemon configuration variable `database_vendor` defines the selected database backend, as shown in Figure 6-3.

Figure 6-3. Database vendor variable.

```
# Valid values are the ones accepted by the Perl DBI.
#
# For Bugzilla, this value is ignored
#
# For Mantis, *some* valid values (there are others) are:
# - 'mysql'
# - 'Pg'
```

```
database_vendor => 'mysql',
```

The Mantis developers are planning (http://www.futureware.biz/mantis/view.php?id=151) to provide a formal SCM integration interface (http://www.futureware.biz/mantisconnect/) in future releases.

As of version 0.19.0, Mantis does not yet support a mapping of the SCM username to a Mantis username. An SCM to bug-tracking username mapping is accomodated by the integration daemon using the `userlist` variable, as described in Section 4.4.8.

Scmbug has been verified to work against the following releases of Mantis:

- *0.19.0*
- *1.0.0rc3*
- *1.1.0rc1*

6.3. Request Tracker

Request Tracker is a web based request tracking system written in object-oriented Perl.

Request Tracker provides a public interface for SCM integration. The Scmbug daemon reuses functionality already available in the Perl-based Request Tracker libraries. As a result, the library code used to host a Request Tracker instance must be locally accessible by the daemon. The `installation_directory` variable is used to define the location of the Request Tracker installation as shown in Figure 6-4.

Figure 6-4. Bug-tracker installation directory for Request Tracker.

```
# Used for Bugzilla and RequestTracker.
#
# Path to the directory of the bugtracker sources providing an
# API that the daemon can use
installation_directory => '/usr/share/request-tracker3.4/lib'
```

This backend integration with Request Tracker assumes the product names in `glue.conf` match the queue names set in the Request Tracker.

The `activity_tag` integration action is not yet supported.

The Request Tracker backend does not currently support the *VDD Generator* and *Merger*.

As of version 3.4.5, Request Tracker does not require a mapping of the SCM username to a Request Tracker username. The username mappings can be disabled as shown in Figure 4-13.

Scmbug has been verified to work against the following releases of Request Tracker:

- *3.4.5*
- *3.6.1*

6.4. Test Director

Test Director is a test and fault tracking system.

Test Director is based on MS Windows and hence the interfaces into the system are available on MS Windows only. Users have to run the Scmbug daemon on a Windows machine.

In order to use Test Director as your fault tracking system you will need to ensure that you install:

- The Perl OLE package.
- Test Director *Mercury Quality Center System Test Remote Agent Add-in* (Available from the Test Director `Help-> Add-ins` menu).

For Test Director, some example configuration settings for the daemon configuration file `/etc/scmbug/daemon.conf` are shown in Figure 6-5.

Figure 6-5. Example daemon configuration settings for Test Director.

```
# The URL connection string to Test Director.
database_location => 'http://emea-testdir:8080/qcbin',

# The database vendor is the same as the Domain within Test Director,
# This must match the value that you would use on manual login.
database_vendor => 'Domain',

# The database name is the same as the Project within Test Director.
database_name => 'Project',

# The user name is the Test Director user login.
```

```
database_username => 'username',

# The password for the Test Director login.
database_password => 'password',
```

Test Director and its fields can be further customized in the file `/etc/scmbug/TestDirector.conf`. This file must be updated to match your system.

The Test Director backend does not currently support the *VDD Generator*.

As of version 9.0.0, Test Director does not yet support a mapping of the SCM username to a Test Director username. An SCM to bug-tracking username mapping is accomodated by the integration daemon using the `userlist` variable, as described in Section 4.4.8.

> **Note:** Due to a bug (http://bugs.activestate.com/show_bug.cgi?id=38968) in Windows ActiveState Perl, the backend would crash if a Win32::OLE object was called from a forked process (the forked daemon connection handler). For this reason the Test Director backend calls are all executed from a seperate script and the results are read back in. This does not limit the functionality, but it does mean that sub-processes are created.

Scmbug has been verified to work against the following releases of Test Director:

- *9.0.0 (Quality Center), 9.2*

6.5. Other Bug-tracking Systems

Additional bug-tracking backends can be supported by Scmbug. Developers and system integrators of the following bug-tracking systems are **strongly encouraged** to contribute a bug-tracking integration backend:

- *AntHill*
- *Bosco*
- *debbugs*
- *Double Choco Latte*
- *Eventum*
- *GForge*

- *GNATS*
- *Helis*
- *ITracker*
- *phpBugTracker*
- *Roundup*
- *Scarab*
- *Trac*
- *TUTOS*
- *Workbench*

Developing a backend requires:

- Committing to support this backend in future releases of Scmbug.
- Creating a new backend module named `src/lib/product/Daemon/BackendName.pm.in`. The Bugzilla backend `src/lib/product/Daemon/Bugzilla.pm.in` serves as a good example.
- Updating `src/lib/product/Daemon/Daemon.pm.in:read_configuration` accordingly.
- Updating the configuration management files `configure.in` and `Makefile.in` to autogenerate, and autocleanup the new backend.
- Updating the `install-server` rule of `Makefile.in` to install the new backend from source.
- Updating the documentation in `doc/manual/content` to reflect support for the new backend.

Chapter 7. Integration Tools

7.1. Glue Installer

The command `scmbug_install_glue.pl` is used to install the integration glue in an SCM repository. Existing hooks used in the SCM system are not overwritten.

The `--binary-paths` configuration option should be set to a list of paths that include all binaries needed by the integration tools and the SCM system. This includes the tools `diff`, `xsltproc`, `docbook2pdf`, `docbook2html` and the SCM system's binaries. The `--bug` configuration option should be set to the id of a bug in the bug-tracker against which the integration glue installation will be documented.

> **Tip:** It is preferable to set in `--binary-paths` the path to the directory that contains a binary, instead of the path to the binary itself. Using Subversion in Windows for example, the binaries require additional libraries (e.g. .dll files) that are contained in the directory that holds the binary.

An example installing the glue in a Subversion repository under UNIX is shown in Figure 7-1.

Figure 7-1. Glue Installation in a Subversion repository under UNIX.

```
$ scmbug_install_glue.pl --scm=Subversion --product=TestProduct \
  --repository=file:///tmp/testrepository --bug=770 --binary-paths=/bin,/usr/bin \
  --daemon=127.0.0.1
 This is the installation script of the Scmbug glue.
 The glue will be installed in repository: file:///tmp/testrepository.
 This is a repository for the Subversion SCM tool.
 The product name in the bug tracking system is TestProduct.
 The integration glue will be committed against bug 770.
 The IP address of the Scmbug integration daemon to contact is 127.0.0.1.
 The binary paths used are: /bin,/usr/bin
 -------------
 Press Enter to continue or Ctrl-C to cancel

Glue processing has been prepared in /tmp/Scmbug.30670
Installing part1
Check everything there before I commit or hit Ctrl-C to exit

Glue processing has been prepared in /tmp/Scmbug.30670
Installing part2
Check everything there before I commit or hit Ctrl-C to exit
```

> **Note:** Under Windows, the paths supplied to the installer should have directories separated with a **forward (/) slash**. Figure 7-2 shows an example running the installer under Windows.

Figure 7-2. Glue Installation in a CVSNT repository under Windows.

```
C:\Program Files\Scmbug\bin> scmbug_install_glue.pl --scm=CVS --product=sdvel \
--repository=c:/cvsroot --bug=22 --daemon=192.168.136.140 \
--binary-paths="C:/Program Files/CVSNT"
```

7.1.1. CVS

Integration glue can be installed in both local and remote CVS repositories.

After installation, the file <CVS_REPOSITORY_PATH>/CVSROOT/etc/scmbug/glue.conf holds the configuration of the glue.

Warning

CVSNT integration has only been verified to work with a `:pserver:` configuration. A `CVSROOT` starting with `:local:` does not work. This is because of the presumptious way CVSNT handles the `CVSROOT` variable. Apparently for some configurations it expands it to include the configuration type. The integration hooks use this expanded variable and as a result are pointed to incorrect paths.

Warning

CVSNT uses the two variables `Name` and `Root` when configuring a repository, instead of only the `CVSROOT` variable. When hooks execute, CVSNT sets `CVSROOT` to be equal to `Name`. By default, CVSNT sets `Name` omitting the drive letter. The hooks will not execute if `Name` is not set to the full path of the repository since `$CVSROOT` will be referring to a path that does not exist. An example invalid `Name/Root` configuration is shown in Figure 7-3. An example valid `Name/Root` configuration is shown in Figure 7-4.

Figure 7-3. Example invalid `Name/Root` for CVSNT.

```
Name: /Projects
Root: C:/cvsrepos/Projects
```

Figure 7-4. Example valid `Name/Root` for CVSNT.

```
Name: C:/cvsrepos/Projects
Root: C:/cvsrepos/Projects
```

Setting `Name` to be equal to `Root` produces the warning shown in Figure 7-5. The *compatibility problems* seem to refer to some clients simply not parsing a `CVSROOT` with a drive letter in it. If your development environment includes UNIX clients, and problems do occur, one alternative is to let `Name` lack a drive letter and manually edit the hook files to include the full repository path.

Figure 7-5. CVSNT warning when `Name` is set to `Root`.

```
Using drive letters in repository names can create compatibility problems w
Unix clients and is not recommended. Are you sure you want to continue ?
```

A major drawback of CVS is it's lack of atomic transactions. As a side-effect, when the same log message is used to commit files in two separate directories, two integration activities are issued using the same log message. Duplicate log messages are then entered in the bug-tracking system. Scmbug solves this problem by using the `commitinfo` and `loginfo` hooks to detect commits in separate directories and consolidate the log messages entered in the bug-tracking system as one log message (one integration activity). This behavior is optional and can be configured in the glue configuration file using the `consolidate_cvs_messages` variable, as shown in Figure 7-6. However, ActiveState ActivePerl does not yet implement the `getppid()` (http://bugzilla.mkgnu.net/show_bug.cgi?id=1074) function. Windows systems running Scmbug need to disable `consolidate_cvs_messages`.

Figure 7-6. Configuration option that consolidates CVS messages.

```
# This applies only to CVS. When a commit affects more than
# one directory, multiple duplicate log comments are inserted,
# one-per-directory. Enabling this option would consolidate
# the commits to all use the first log message.
consolidate_cvs_messages => 1
```

7.1.2. Subversion

Subversion repositories do not support installation of the integration glue remotely. Local repository access is required.

After installation, the file <SVN_REPOSITORY_PATH>/hooks/etc/scmbug/glue.conf holds the configuration of the glue.

Subversion does not distinguish between commit activities and creation of tags or branches. It *recommends* (http://svnbook.red-bean.com/svnbook/ch04s07.html) that the user manually creates top-level directories named /trunk, /tags and /branches. When it's time to create a tag or branch, Subversion proposes following the convention of creating a copy of the main trunk using 'svn copy' in the /tags or /branches directories. As a result, the glue must manually detect if an activity_verify issued by Subversion also implies an activity_tag. To do so, it checks for addition of new subdirectories in the directories /tags or /branches. This behavior is defined in the glue configuration file using the label_directories variable, as shown in Figure 7-7.

Figure 7-7. Defining the Subversion labeling directories.

```
# This applies only to Subversion. It is recommended that tags
# are stored in the 'tags' directory, and branches in the
# 'branches' directory.
label_directories => [
      'tags',
      'branches'
      ]
```

Similarly, Subversion does not distinguish between the main development line and other branches. The glue must manually detect if a changeset is committed under /trunk. This behavior is defined in the glue configuration file using the main_trunk_directories variable, as shown in Figure 7-8.

Figure 7-8. Defining the Subversion main trunk directories.

```
# This applies only to Subversion. It is recommended that the
# main trunk work is stored in the 'trunk' directory.
main_trunk_directories => [
```

```
'trunk'
]
```

Note: It is not mandatory that `trunk`, `tags`, and `branches` are created in the root of the repository. A more flexible directory structure can be defined using the `product_name_definition` variable as shown in Figure 4-17.

7.1.3. Git

Git repositories do not support installation of the integration glue remotely. Local repository access is required.

After installation, the file <`GIT_REPOSITORY_PATH`>`/.git/hooks/etc/scmbug/glue.conf` holds the configuration of the glue.

7.2. Version Description Document Generator

Since SCM changes are integrated with bug-tracking, it is possible to produce a list of changes that occured for a particular version of a software at a level that's higher than source changes.

`ChangeLog` information derived strictly from the SCM system, such as a report produced using the *cvs2cl* tool for CVS or using `'svn log'` in Subversion, is overly detailed. It describes software changesets at a lower level, which interests mostly developers. It is of little value to a user simply interested in a summary of added features. Moreover, when multiple changesets are committed in response to a defect, such a document becomes lengthy. It takes considerable time to follow the history of changes and decipher if, or how, a defect was corrected.

Instead, a VDD reports at a higher level a summary of the features/defects worked on and *why*, using information recorded in the bug-tracking system. It provides additional useful information such as resolution status, bug owner, severity, and priority. It also reports *what* changes occured at a lower level in the SCM system per bug, effectively superseding `ChangeLog` documents produced strictly from the SCM system. Without integration of SCM with bug-tracking, this level of detail in a release document would not be possible.

Given two SCM label names (tag or branch names), this tool queries the SCM system for the dates the labels were applied. It then queries the bug-tracking system to produce a report of the bugs worked between that date range. A VDD can additionally reflect decisions of the development team which are not documented in the SCM logs, such as choosing to not add a feature, resolving it as WONTFIX. It

may also display bugs that were added in the period between releases but not worked yet, alerting users of newly discovered defects.

A VDD can be generated using the command `scmbug_vdd_generator.pl`. An example producing this document is shown in Figure 7-9.

Figure 7-9. Generating a Version Description Document.

```
$ scmbug_vdd_generator.pl --scm=Subversion --product=TestProduct \
  --repository=file:///tmp/testrepository --from=tags/SCMBUG_RELEASE_0-8-1 \
  --to=tags/SCMBUG_RELEASE_0-8-2
```

The output of this tool is a collection of files. An XML file is produced that contains the result of the VDD query. This file is also transformed using XSLT into a Docbook 4.2 SGML file. Finally, this SGML file is processed using Docbook tools to produce PDF and HTML output.

7.3. Merger

There are plans (http://bugzilla.mkgnu.net/show_bug.cgi?id=545) to provide a tool that will merge the work done on a list of bugs into a given label.

Work done on bugs can be merged in a label using the command `scmbug_merge.pl`. Two merging capabilities are available:

- **Merging bug changes in a codebase based on an existing tag, and applying a new tag**: The user supplies a `--base-label` option which specifies the label (usually a tag) against which the changes in the specified list of bug ids should be applied. A temporary branch is created based on this label, and the bug changes are applied on this branch. The resulting codebase is label with the name specified by `--target-label`, and the temporary branch is deleted.

  An example merging bug changes in a codebase based on an existing tag is shown in Figure 7-10.

  Figure 7-10. Merging bug changes in a codebase based on an existing tag.

  ```
  $ scmbug_merge.pl --scm=Subversion --product=TestProduct \
    --repository=file:///tmp/testrepository --base-label=tags/SOMEPRODUCT_RELEASE_1-3-0 \
    --new-label=tags/SOMEPRODUCT_RELEASE_1-3-1 --merge-bugs=545,591 --commit-bugs=771
  ```

- **Merging bug changes directly in an existing branch**:The user supplies a `--base-label` option which specifies the label (usually a branch or the main development line) against which the changes in the specified list of bug ids should be applied. The changes are applied directly in this label. No `--target-label` option is supplied.

An example merging bug changes directly in an existing branch is shown in Figure 7-11.

Figure 7-11. Merging bug changes directly in an existing branch.

```
$ scmbug_merge.pl --scm=Subversion --product=TestProduct \
  --repository=file:///tmp/testrepository \
  --base-label=branches/SOMEPRODUCT_RELEASE_1-x-0_series --merge-bugs=708 --commit-bugs
```

7.4. Web Reports

A collection of Web Reports provide easy to use graphical access to the integration tools.

Chapter 8. Resources

8.1. Availability

Scmbug is available for UNIX and Windows systems. It is expected to work with Perl version 5.6.1 or later and has been confirmed to work with Perl version 5.8.4. Dependencies on additional programs are described in Section 8.2.

The project's webpage (http://freshmeat.net/projects/scmbug) contains the most up to date information on the project, including the latest release and manual. A users mailing list is available for subscription (http://lists.mkgnu.net/mailman/listinfo/scmbug-users), or simply for sending email (mailto:scmbug-users@lists.mkgnu.net). The project's bug-tracking system (http://bugzilla.mkgnu.net) contains the latest TODO list (http://bugzilla.mkgnu.net/buglist.cgi?query_format=specific&bug_status=__open__&product=Scmbug&content=&orc Source code access to developers is available using anonymous CVS as shown in Figure 8-1 or through ViewCVS (http://www.mkgnu.net/cgi-bin/viewcvs.cgi/scmbug/).

Figure 8-1. Developer access to project's CVS repository.

```
bash$  cvs -d:pserver:anonymous@cvs.mkgnu.net:/projects/scmbug/cvsroot login
Password:
bash$  cvs -d:pserver:anonymous@cvs.mkgnu.net:/projects/scmbug/cvsroot co .
```

8.2. Installation

8.2.1. System

Under UNIX, Scmbug is available in the form of Debian and RPM packages. The provided packages are:

- `scmbug-common`: common libraries.
- `scmbug-doc`: documentation.
- `scmbug-tools`: tools that can install the integration glue in an SCM repository and enhance the experience of integrating SCM with bug-tracking.
- `scmbug-server`: the integration daemon.

Tip: If you believe your system meets the package dependencies, but installing packages fails due to missing dependencies, installation of the packages is still possible. Installation of RPM packages can be forced as shown in Figure 8-2, and installation of Debian packages can be forced as shown in Figure 8-3.

Figure 8-2. Forcing installation of RPM packages.

```
bash$ rpm -ivh --force --nodeps <RPM_PACKAGE_NAME>
```

Figure 8-3. Forcing installation of Debian packages.

```
bash$ dpkg -i --force-depends <DEB_PACKAGE_NAME>
```

Under Windows, the entire Scmbug system is available in the form of a single .zip file. It must be manually installed in `C:/Program Files/Scmbug`. It requires ActiveState ActivePerl (http://www.activestate.com/Products/ActivePerl/) installed in `C:/Perl/bin/perl`, and a temporary directory called `C:/Temp`.

Source code distributions are also available. Figure 8-4 shows how the system can be configured and installed from source. It is possible to choose a different destination of the libraries and binaries of Scmbug both in UNIX and Windows systems at configuration time. It is also possible to build the system without documentation. If you are configuring from source code, running **'./configure --help'** can provide more information.

Figure 8-4. Installation of the system from source.

```
bash$ ./configure
bash$ make
bash$ su
bash# make install-common
bash# make install-doc
bash# make install-tools
bash# make install-server
```

Tip: Installing all these packages will **NOT** automatically integrate an SCM repository with a bug-tracking system. It will only install the basic software needed to do so.

A user must configure the *Integration Daemon* and start it as shown in Figure 8-5. Then, a user must run the *Glue Installer* to install the Scmbug integration in an SCM repository.

Figure 8-5. Integration daemon start.

Starting the Integration Daemon in UNIX systems:

```
bash# /etc/init.d/scmbug-server start
```

Starting the Integration Daemon in Windows systems:

```
C:\> cd C:/Program Files/Scmbug/etc/init.d
C:/Program Files/Scmbug/etc/init.d> scmbug-server.bat
```

Tip: When specifying paths in Windows, either in the glue installer, the glue configuration file, or the integration daemon, paths should have directories separated with a **forward (/) slash**.

Tip: There are various dependencies on the packages provided. Both dependencies on other packages, and on Perl modules. If they are ignored, the *Integration Daemon* and the *Integration Tools* will detect the missing dependency and refuse to execute.

8.2.2. Documentation

Installation of documentation, including this manual, requires installing the package `scmbug-doc`.

8.2.3. Common libraries

The common libraries require instaling the Perl module `Log::Log4perl` for logging, as shown in Figure 8-6.

Figure 8-6. XML::Simple installation.

Installation in UNIX systems:

```
bash$ su
bash# perl -MCPAN -e "install Log::Log4perl"
```

Installation in Windows systems:

```
C:\> ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState Corp. All Rights Reserved.
ActiveState is a devision of Sophos.

Entering interactive shell. Using Term::ReadLine::Stub as readline library.
```

```
Type 'help' to get started.
ppm> install Log-Log4perl
```

8.2.4. Integration Tools

Installation of the *Integration Tools* requires installing the packages `scmbug-common` and `scmbug-tools`. It also requires a diffing tool. For Windows, one such binary (`diff.exe`) is available in GnuWin32 (http://gnuwin32.sourceforge.net//).

Running the *VDD Generator* additionally requires installing the `xsltproc`, and `docbook-utils` packages. It is uncertain where one could get these packages for Windows. An `xsltproc` distribution for Windows seems to be available from Igor Zlatkovic's website (http://www.zlatkovic.com/libxml.en.html), but we have yet to locate a `docbook-utils` package for Windows. eDE (http://www.e-novative.info/software/ede.php) seems to be usable under Windows, but he have yet to try it. However, both `xsltproc` and `docbook-utils` are provided by Cygwin (http://www.cygwin.com/).

Running the *Merger* requires instaling the Perl module `XML::Simple` (for an `activity_get_bugs`), as shown in Figure 8-7.

Figure 8-7. XML::Simple installation.

Installation in UNIX systems:

```
bash$ su
bash# perl -MCPAN -e "install XML::Simple"
```

Installation in Windows systems:

```
C:\> ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState Corp. All Rights Reserved.
ActiveState is a devision of Sophos.

Entering interactive shell. Using Term::ReadLine::Stub as readline library.

Type 'help' to get started.
ppm> install XML-Simple
```

Installation of the *Web Reports* requires configuring a webserver in a way similar to the Apache configuration shown in Figure 8-8.

Figure 8-8. Apache configuration for Web Reports.

```
Include /etc/scmbug/apache.conf
```

8.2.5. Integration Daemon

Installation of the daemon requires installing the packages `scmbug-common` and `scmbug-server`. Additionally, it requires installing the Perl modules `Mail::Sendmail` (for policy *Mail notification*) and `XML::Simple` (for an `activity_get_vdd` issued by the *VDD Generator*), as shown in Figure 8-9. If an SCM to bug-tracking username mapping is configured based on variable `mapping_ldap`, as described in Section 4.4.8, the Perl module `Net::LDAP` must also be installed.

Figure 8-9. Mail::Sendmail, XML::Simple installation.

Installation in UNIX systems:

```
bash$ su
bash# perl -MCPAN -e "install Mail::Sendmail"
bash# perl -MCPAN -e "install XML::Simple"
```

Installation in Windows systems:

```
C:\> ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState Corp. All Rights Reserved.
ActiveState is a devision of Sophos.

Entering interactive shell. Using Term::ReadLine::Stub as readline library.

Type 'help' to get started.
ppm> install Mail-Sendmail
ppm> install XML-Simple
```

The file `/etc/scmbug/daemon.conf` holds the configuration of the daemon.

The daemon can be started in different execution modes. These are defined in the daemon configuration file using the `daemon_mode` variable and are:

- **Threaded**. Multiple threads are created to handle incoming connections. This was observed to work well under Windows but cause a significant slowdown under UNIX. For more information see bug 264 (http://bugzilla.mkgnu.net/show_bug.cgi?id=264#c7).

- **Forked**. Multiple processes are created to handle incoming connections. This was observed to work well under UNIX and most Windows systems. The daemon unexpectedly dies during some connections in Windows. For a while, it was believed that this was due to an outdated ActiveState Perl or running a version of the DBI < 1.50, but there seems to be more to the problem. For more information see bug 597 (http://bugzilla.mkgnu.net/show_bug.cgi?id=597#c9), bug 264 (http://bugzilla.mkgnu.net/show_bug.cgi?id=264#c20), and bug 646 (http://bugzilla.mkgnu.net/show_bug.cgi?id=646#c7).
- **Automatically detected**. Automatically chooses between a threaded or forked mode.

Integration with multiple bug-tracking systems can be accomplished by starting multiple daemons listening at different ports.

8.2.5.1. Bugzilla

For reasons explained in Section 6.1, the integration daemon requires local presence of the source code used to run a Bugzilla instance. It also requires access to the database used to store Bugzilla's data. It is recommended that the daemon is installed on the same machine Bugzilla runs, but this is not required. If the daemon *is* installed on the same machine Bugzilla runs, Scmbug will force the Bugzilla version cache (which is a local file) to be regenerated when `activity_tag` integration requests are processed.

> **Warning**
>
> On Windows, DBI version 1.50 or greater is required, and can be installed as shown in Figure 8-10. Earlier versions have threading issues.

8.2.5.2. Mantis

Integration with Mantis requires access to the database used to store Mantis' data. At a minimum, it requires installing the `DBI` Perl module, as shown in Figure 8-10.

> **Warning**
>
> On Windows, DBI version 1.50 or greater is required. Earlier versions have threading issues.

Figure 8-10. DBI installation.

Installation in UNIX systems:

```
bash$ su
bash# perl -MCPAN -e "install DBI"
```

Installation in Windows systems:

```
C:\> ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState Corp. All Rights Reserved.
ActiveState is a devision of Sophos.

Entering interactive shell. Using Term::ReadLine::Stub as readline library.

Type 'help' to get started.
ppm> install DBI
```

As explained in Section 6.2, Mantis supports installation with multiple database backends, and requires the corresponding DBD Perl module to be installed. For example, if Mantis is installed with a MySQL database backend, the `DBD::mysql` Perl module must also be installed(Figure 8-11) and the `database_vendor` variable must be configured as shown in Figure 6-3.

Figure 8-11. DBD::mysql installation.

Installation in UNIX systems:

```
bash$ su
bash# perl -MCPAN -e "install DBD::mysql"
```

Installation in Windows systems:

```
C:\> ppm
PPM - Programmer's Package Manager version 3.1.
Copyright (c) 2001 ActiveState Corp. All Rights Reserved.
ActiveState is a devision of Sophos.

Entering interactive shell. Using Term::ReadLine::Stub as readline library.

Type 'help' to get started.
ppm> install DBD-mysql
```

8.2.5.3. Request Tracker

For reasons explained in Section 6.3, the integration daemon requires local presence of the source code used to run a RequestTracker instance. It has not yet been investigated if it also requires local access to the database used to store RequestTracker's data.

8.3. Upgrading

Upgrading to newer versions of Scmbug is possible. It first requires understanding the issues involved in running non-matching versions of the Glue and Daemon.

8.3.1. Issues

The communication protocol between the Glue and the Daemon is upgraded and is backwards-incompatible when the minor version number of Scmbug increases. For example, SCMBUG_RELEASE_0-1-x and SCMBUG_RELEASE_0-2-x are not compatible, but SCMBUG_RELEASE_0-2-1 and SCMBUG_RELEASE_0-2-8 are.

Tip: During SCM events activity, if the Glue detects an incompatible version of the Daemon, it will refuse to continue.

Warning

Running incompatible versions of the Glue and Daemon can result in developers being locked out of an SCM repository.

For example, when using CVS it could result in the integration failing to commit any changes, leading into a dead-end. This would prohibit disabling the Glue in its entirety, and require manually editing the repository locally using RCS commands. Certainly, an SCM repository can always be brought back to a working state by a competent administrator.

Since version SCMBUG_RELEASE_0-3-3, the Glue Installer is able to safely upgrade the integration to newer versions. It first completely disables all SCM hooks installed by Scmbug, upgrades the Glue code, and then enables the hooks. During the last step, the Daemon does not need to be contacted, since the hooks are not yet active.

Note: If you understand the implementation internals of Scmbug you will be able to determine whether a new feature is implemented in the Glue or the Daemon, and upgrade only the Glue or only the Daemon.

One example where you might not want to upgrade both the Glue and Daemon is if you have a lot of SCM repositories and you need a new feature that has been implemented only on the Daemon. Upgrading only the daemon is a lot easier than first upgrading all repositories. Another example would be upgrading only the Glue of a single SCM repository if you need a new Scmbug feature only in that repository.

Tip: If in doubt, upgrade everything.

8.3.2. Steps

Upgrading to a newer version of Scmbug requires first upgrading the Glue and then the Daemon, by carrying out in order the following steps.

8.3.2.1. Glue Upgrading

- *Upgrading Scmbug*: Install newer versions of the packages `scmbug-common`, `scmbug-doc`, and `scmbug-tools`, as shown in Section 8.2.1.

  Tip: If one of the SCM repositories that will be upgraded is hosted by Subversion, then these packages must also be installed on the machine hosting the Subversion repository. As explained in Section 7.1.2, local repository access will be required in the next step to upgrade the integration glue.

- *Upgrading all SCM repositories*: Run the *Glue Installer* once per SCM repository with respective arguments to upgrade to a newer version of the Glue, as described in Section 7.1.

  Warning

  The Glue Installer will display changes between the existing and updated glue configuration file in diff format. Manually merge-in any updates to the glue configuration file as needed. Failure to do so could result in the Glue failing to work.

8.3.2.2. Daemon Upgrading

- *Upgrading the Integration Daemon*: After **all** SCM repositories have been upgraded, upgrade the Integration Daemon by installing a newer version of the packages `scmbug-common` and

`scmbug-server`, as shown in Section 8.2.5.

> **Warning**
>
> Ommiting to upgrade an SCM repository before upgrading the Daemon could result in developers being locked out of the repository.
>
> Should that happen, it's possible to downgrade to the previous version of the Daemon, upgrade the repository as described in Section 8.3.2.1, and upgrade the Daemon again. During this process, all other upgraded SCM repositories will be unable to accept activity, since the Glue will detect an incompatible Daemon and refuse to continue.

Appendix A. FAQ

This FAQ includes questions not covered elsewhere in this manual.

Appendix B. GNU Free Documentation License

Version 1.1, March 2000

0. Preamble

The purpose of this License is to make a manual, textbook, or other written 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 non-commercially. 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.

1. Applicability and Definition

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

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. (For example, 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.

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 "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or non-commercially, 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.

3. Copying in Quantity

If you publish printed copies 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 publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. 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.

4. 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:

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 less than five).

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", and 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. In any section entitled "Acknowledgments" or "Dedications", preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgments 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 as "Endorsements" or to conflict in title with any Invariant Section.

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.

5. 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 Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on

covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. 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.

10. 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.

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:

> 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 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossary

A

Aegis

Aegis (http://freshmeat.net/projects/aegis/) is a transaction-based software configuration management system. It provides a framework within which a team of developers may work on many changes to a program independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible. Aegis supports geographically distributed development.

AntHill

Anthill (http://freshmeat.net/projects/anthill/) is a bug tracking database system written in PHP. It provides the standard bug tracking features such as: user logins, summary reports, submitting bugs, querying bugs, various severity and status levels. It also provides some unique features, such as a template system, and multi-lingual support.

Arch

Arch (http://regexps.srparish.net/www/#Gnu-arch) is a modern replacement for CVS, specifically designed for the distributed development needs of open source projects. It has uniquely good support for development on branches (especially good merging tools), distributed repositories (every developer can have branches in their own repository), changeset-oriented project management (arch commits changes to multiple files at once), and, of course, file and directory renaming.

B

Bazaar-NG

Bazaar-NG (http://www.bazaar-ng.org/) is a powerful, friendly, and scalable open source distributed version control system. It attempts to combine the best features from all free version control systems into a single coherent and simple system.

Bitkeeper

Bitkeeper (http://freshmeat.net/projects/bitkeeper/) is a fully distributed source management system, supporting globally distributed development, disconnected operation, change sets, and many active branches. It is used by some major projects such as the Linux kernel, MySQL, Xaraya, and Xen.

Bosco

Bosco (http://freshmeat.net/projects/bosco/) is a rewrite of the popular Bugzilla defect tracking software in PHP. It is database-independent, and aims to be easy to maintain and modify. It also has an API to allow external applications to work with its data.

Bugzilla

Bugzilla (http://www.bugzilla.org) is an enterprise-class piece of software that tracks millions of bugs and issues for hundreds of organizations around the world.

C

ClearCase

ClearCase (http://www-306.ibm.com/software/awdtools/clearcase/) is a version control system.

CVS

CVS (http://www.cvshome.org/) is the Concurrent Versions System, the dominant, open-source, network-transparent version control system. It allows you to keep old versions of files (usually source code), keep a log of who, when, and why changes occurred, etc., like RCS or SCCS. Unlike the simpler systems, CVS does not just operate on one file at a time or one directory at a time, but operates on hierarchical collections of directories consisting of version controlled files. CVS helps to manage releases and to control the concurrent editing of source files among multiple authors. CVS allows triggers to enable/log/control various operations and works well over a wide area network.

cvs2cl

A *tool* (http://www.red-bean.com/cvs2cl/) that autogenerates a `ChangeLog` document from CVS.

D

debbugs

debbugs (http://www.benham.net/debbugs/) is the Debian bug-tracking system.

Double Choco Latte

Double Choco Latte (http://freshmeat.net/projects/doublechocolatte/) is a system for tracking bugs, changes, enhancements, and requests for software. The system is suited for multiple products and multiple accounts (clients). It is also known to handle call center activity, although this will evolve into a separate module.

E

Eventum

Eventum (http://dev.mysql.com/downloads/other/eventum/) is a user-friendly and flexible issue tracking system that can be used by a support department to track incoming technical support requests, or by a software development team to quickly organize tasks and bugs. Eventum is used by the MySQL AB Technical Support team.

G

GForge

GForge (http://www.gforge.org) is a Web-based collaborative development environment. It's based on a fork of the 2.61 SourceForge code, which used to be available via anonymous CVS from VA Software, but has been extensively rewritten and enhanced.

GIT

GIT (http://freshmeat.net/projects/git) is a "directory content manager" that was designed to handle massive projects such as the Linux kernel with speed and efficiency. It falls in the category of distributed source code management tools and is similar to GNU Arch, Monotone, and BitKeeper. Every GIT working directory is a fully-fledged repository with full revision tracking capabilities and is not dependent on network access to a central server.

GNATS

GNATS (http://freshmeat.net/projects/gnats/) is the GNU bug-tracking system, a portable incident/bug report/help request-tracking system which runs on UNIX-like operating systems. It easily handles thousands of problem reports, has been in wide use since the early 90s, and can do most of its operations over e-mail. Several front end interfaces exist, including command line, emacs, and Tcl/Tk interfaces. There are also a number of Web (CGI) interfaces written in scripting languages like Perl and Python.

H

Helis

Helis (http://freshmeat.net/projects/helis/) includes the main features of most bug tracking systems. It is helpful for managing required evolutions, lacks, proposals, and bugs. Authenticated users can reach the database through a Web browser (Mozilla, Netscape, or IE). Distinct features include the ability to manage releases (e.g. return resolved bugs between release 1.04 and 1.00), precise access rights, and managing validation reports.

I

ITracker

ITracker (http://freshmeat.net/projects/itracker/) is a Java J2EE issue/bug tracking system designed to support multiple projects with independent user bases. It supports features such as full i18n support, multiple versions and project components, detailed histories, issue searching, file attachments, dynamic reports with charts, configurable field values, customizable project level fields, pluggable authentication, a built-in scheduler, and email notifications.

K

Katie

Katie (http://freshmeat.net/projects/katie/) is a revision control system, somewhat like a cross between CVS and NFS, that was inspired by Rational ClearCase. The three most interesting features are that the repository is mounted as a filesystem (rather than being copied to a local workspace), that all versions of all files (even deleted ones) are accessible through this filesystem (so the "katie diff" command is a convenience rather than a necessity like "cvs diff"), and that directories are versioned (just like files are). It is functional enough to be self-hosting, but there is much work still to go before it will be a generally useful tool. Features that are implemented already include VOBs, elements, branches, dynamic views, view-extended pathnames, config specs (including auto-make-branch rules), labels, hard links, and symbolic links.

M

Mantis

Mantis (http://freshmeat.net/projects/mantis/) is a PHP/MySQL-based bug-tracking system. It is extremely easy to deploy and customize, and features one of the simplest and cleanest interfaces of any tracking tool available. It supports multiple projects and email notification, and is localized for over 18 languages.

Mercurial

Mercurial (http://freshmeat.net/projects/mercurial/) is a fast, lightweight Source Control Management system designed for the efficient handling of very large distributed projects.

Monotone

Monotone (http://freshmeat.net/projects/monotone/) is a distributed version control system with a flat peer model, cryptographic version naming, meta-data certificates, decentralized authority, and overlapping branches. It works out of a transactional version database stored in a regular file, and uses a custom network protocol for efficient database synchronization.

O

OpenCM

OpenCM (http://freshmeat.net/projects/opencm/) is designed as a secure, high-integrity replacement for CVS. It includes features such as file renaming, branch and file level access control, cryptographic authentication, and end-to-end integrity controls.

P

Perforce

Perforce (http://freshmeat.net/projects/perforce/) is a software configuration management system that is fast, robust, runs on over 50 platforms, and scales to over 1000 users on a single repository. It supports atomic submits and works well over wide area networks, including the Internet.

phpBugTracker

phpBugTracker (http://freshmeat.net/projects/phpbt/) is an attempt to copy the functionality of Bugzilla while providing a codebase that is independent of the database and presentation layers.

R

Request Tracker

Request Tracker (http://www.bestpractical.com/rt/) is an enterprise-grade ticketing system which enables a group of people to intelligently and efficiently manage tasks, issues, and requests submitted by a community of users. Written in object-oriented Perl, RT is a high-level, portable, platform independent system that eases collaboration within organizations and makes it easy for them to take care of their customers.

Roundup

Roundup (http://roundup.sourceforge.net/) is a simple-to-use and -install issue-tracking system with command-line, web and e-mail interfaces. It is based on the winning design from Ka-Ping Yee in the Software Carpentry "Track" design competition.

S

Scarab

Scarab (http://freshmeat.net/projects/scarab/) is an issue tracking system that features data entry, queries, reports, notifications to interested parties, collaborative accumulation of comments, dependency tracking, and collaborative prioritization (voting). It uses Java Servlet technology to enhance speed, scalability, maintainability, and ease of installation. It contains XML import/export support, allowing easy migration from other systems (like Bugzilla). The modular code design eases the modification of features. It is fully customizable via a set of administrative Web pages. The look and feel of the UI can easily be modified, and Scarab can easily be integrated into larger systems. .

Subversion

Subversion (http://subversion.tigris.org/) is a compelling replacement for CVS.

T

Test Director

Test Director (http://www.mercury.com/us/products/quality-center/) is a test and fault tracking system sold by Mercury.

Trac

Trac (http://freshmeat.net/projects/trac/) is a minimalistic but highly useful issue tracker and software project environment based around an integrated Wiki engine. Features include an interface to Subversion (source revision control), a bug/issue tracking database, and convenient report facilities.

TUTOS

TUTOS (http://freshmeat.net/projects/tutos/) (The Ultimate Team Organization Software) is a groupware, ERP (Enterprise Resource Planing), CRM (Customer Relationship Management), and

PLM (Project Lifecycle Management) suite that helps small to medium teams manage various things in one place. Its features include personal and group calendars, an address book, product and project management, bug tracking, installation management, a task list, notes, files, mailboxes, and useful links between all of the above.

W

Workbench

Workbench (http://freshmeat.net/projects/workbench/) is a powerful PHP/MySQL bug-tracking application. It supports multiple projects, full change history, custom reporting, and many other useful tools.