v0.2.16 **March 2015**

Enversion Administration Guide

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and limitations under the License.

© Copyright 2015 Snakebite, Inc.

Issue Record

The amendments made between approved issues of this document are captured below:

Tag	Date	Amendments	Author
v0.2.15	16-Feb-2015	Initial revision.	Trent Nelson
v0.2.16	10-Mar-2015	Updated to v0.2.16.	Trent Nelson

Table 1 Issue Record

Table of Contents

List of Figures

List of Tables

Executive Summary

Enversion is a server-side hook framework for Subversion that blocks over 80 different types of unwanted or problematic commits. It is ideal in enterprise environments where tight controls must be kept on source code control.

The framework is written in Python, and is compatible with Python 2.6 and 2.7. It runs on Linux and OS X.

It is licensed under the same terms as the Apache Subversion project (Apache 2.0).

The project's source code and issue tracker is hosted on github.com:

http://github.com/enversion/enversion

Releases are made available via git tags, signed by "Trent Nelson <trent@trent.me>" with GPG public key DDCBBC36, which is available from the MIT public key server:

https://pgp.mit.edu/pks/lookup?op=get&search=0x25D93491DDCBBC36

1 What's New

This section contains new features and enhancements in reverse chronological order (i.e. newest features are listed first).

1.1 v0.2.16

1.1.1 Configurable Exclusion of Blocking Large Files

Support for blocking commits for files that were over a configurable maximum file size (default 25MB) was added in v0.2.7. This logic was enhanced in two ways in v0.2.16:

1. The ability to exclude files matching a configurable regex from the max size checks was added (see max-file-size-exclusion-regex configuration property). A command was also added to easily test that the regex was matching expected paths after being altered without having to commit files: evnadmin verify-path-matches-max-file-size-exclusion-regex.
2. The original logic was extended such that file sizes are checked in all applicable circumstances – not just when the file is first committed. This prevents the file size limits from being circumvented by committing a file under the limit, then altering it such that it is over the limit.

1.1.2 Update File Extension Blocking Logic to Cover Renames and Replaces

The Blocked File Extensions introduced in v0.2.15 has been updated to cover renames and replaces. Prior to this change, a path was only verified that it didn't match the list of blocked file extensions when it was first added to the repository. The file could then be subsequently renamed to a path that had a blocked file extension without being blocked by Enversion. This change fixes that so files can't be renamed to a blocked file extension.

Additionally, a new command was added to allow an administrator to verify that the regular expression is matching expected paths after being changed without having to commit the files: evnadmin verify-path-matches-blocked-file-extensions-regex.

Since	0.2.16
Config	`blocked-file-extensions-regex`
Issue	https://github.com/enversion/enversion/issues/30
Initial Commit	https://github.com/enversion/enversion/commit/b827e484c691d996aae06ba9d6564539464672c4
Tests	https://github.com/enversion/enversion/blob/b827e484c691d996aae06ba9d6564539464672c4/lib/evn/test/test_blocked_file_extensions.py#L86

Table 2 What's New: Update Blocked File Extension Logic to Cover Renames and Replaces

1.2 v0.2.15

1.2.1 Blocked File Extensions

This change adds support for blocking a commit if it contains a path matching a given regular expression.

Since	0.2.15
Config	`blocked-file-extensions-regex`
Issue	https://github.com/enversion/enversion/issues/28
Initial Commit	https://github.com/enversion/enversion/commit/d36734a3af05cf77a07ccfaff1d87c7a436ca189
Tests	https://github.com/enversion/enversion/blob/master/lib/evn/test/test_blocked_file_extensions.py

Table 3 What's New: Blocked File Extensions

1.2.2 Add Unit Tests for Blocking Large Files

Support was added for blocking large files from being committed to the repository in v0.2.7. This change was made prior to the unit test enhancements, so it didn't feature any unit tests. Unit tests were added in v0.2.15.

Since	0.2.15
Config	`max-file-size-in-bytes`
Issue	https://github.com/enversion/enversion/issues/27
Initial Commit	https://github.com/enversion/enversion/commit/d70dc7d4b6cdaff118215ac899e39570677f4d47
Tests	https://github.com/enversion/enversion/blob/d70dc7d4b6cdaff118215ac899e39570677f4d47/lib/evn/test/test_file_size_limits.py

Table 4 What's New: Unit Tests Verifying Blocking Large Files Logic

1.3 v0.2.14

1.3.1 Custom Hooks

Support for custom hooks was introduced in v0.2.14. A custom hook in this context is a Python class that derives from `evn.custom_hook.CustomHook` and implements either pre_commit or post_commit (or both), e.g.:

```
class OurCustomHook(evn.custom_hook.CustomHook):
      def pre_commit(self, commit, *args, **kwds):
            ...

      def post_commit(self, commit, *args, **kwds):
            ...
```

Figure 1 Example Custom Hook

Enversion will automatically invoke the custom hook class after it has done its processing of the changeset. This allows custom hook logic to leverage things like how the commit was classified (i.e. was a tag created, was a branch modified).

Since	0.2.14
Config	`custom-hook-classname`
Issue	https://github.com/enversion/enversion/issues/7
Initial Commit	https://github.com/enversion/enversion/commit/fa53601b94205ec1c0df4fbabf61970490c4a591
Tests	https://github.com/enversion/enversion/blob/v0.2.14/lib/evn/test/test_custom_hooks.py

Table 5 What's New: Custom Hooks

2 Command Reference

Enversion is interacted with via the CLI program `evnadmin`. This program has been modelled after the Subversion CLI program `svnadmin`, and thus, shares a similar interface and command line parameters where possible.

When run with no arguments, it will print out the available subcommands, e.g.:

```
% evnadmin
Type 'evnadmin help <subcommand>' for help on a specific subcommand.

Available subcommands:
    analyze
    create
    debug
    disable
    disable-remote-debug (drd)
    doctest
    dump-config (dc)
    dump-default-config (ddc)
    dump-hook-code (dhc)
    dump-modified-repo-config (dmrc)
    dump-repo-config (drc)
    enable
    enable-remote-debug (erd)
    find-merges (fm)
    fix-hooks (fh)
    get-repo-component-depth (grcd)
    get-repo-custom-hook-class (grchc)
    list-unit-test-classnames (lutc)
    purge-evn-props (pep)
    root-info (ri)
    run-hook (rh)
    selftest
    set-repo-component-depth (srcd)
    set-repo-custom-hook-class (srchc)
    show-actual-config-file-load-order (sacflo)
    show-actual-repo-config-file-load-order (sarcflo)
    show-debug-sessions (sds)
    show-possible-config-file-load-order (spcflo)
    show-possible-repo-config-file-load-order (sprcflo)
    status
    show-repo-hook-status (srhs)
    show-roots (sr)
    show-writable-repo-override-config-filename (swrocf)
    toggle-remote-debug (trd)
    unittest
    version
```

Figure 2 Sample Output: evnadmin

The commands can be invoked via the shorter abbreviations listed in parenthesis.

2.1 Administration Commands

The most commonly used commands for day-to-day administration of Enversion are listed in this section.

2.1.1 evnadmin create

Creates a new Subversion repository with Enversion enabled, and (optionally) prime the initial repository layout (using svnmucc).

2.1.2 evnadmin analyze

Before Enversion can be enabled against an existing repository, the repository must be analyzed using this command. This will process every revision of the repository and construct the necessary roots in order for automatic root detection to be done once the hooks are enabled.

2.1.3 evnadmin enable

Enables Subversion against an existing repository. This will analyse the repository first if necessary. Enabling Enversion means that the pre and post-commit hooks become active, such that invalid commits will be blocked at the pre-commit stage, and root-mutating commits will alter the evn:roots revision properties at the post-commit stage.

2.1.4 evnadmin disable

Disable Enversion for the given repository. This alters the hooks such that Enversion is no longer invoked during pre and post-commit. This would typically be called only in extra-ordinary circumstances where Enversion is failing or raising unexpected exceptions and normal, valid commits aren't being let through.

Once Enversion is enabled against a repository, it should not be disabled as long as commits can be made. Disabling Enversion will mean that root validation will not take place, nor will evn:roots revision properties be maintained until Enversion is re-enabled (and analysis is run against the revisions that were committed whilst it was disabled).

2.1.5 evnadmin status

This is an alias for `evnadmin show-repo-hook-status`, and provides a means to quickly display whether or not Enversion has been enabled for a given repository.

2.2 Customization Commands

This section describes the commands that you would run against an existing Enversion-enabled Subversion repository in order to customize the behaviour of major features.

2.2.1 evnadmin set-repo-component-depth

This changes the component depth of a repository between single and multi (or disables it completely). See section *3 Repository Layout: Single vs Multiple Component* for more information on repository depth/layout.

2.2.2 evnadmin get-repo-component-depth

Display the current component depth of a repository (single, multi or disabled).

2.2.3 evnadmin set-repo-custom-hook-class

This command changes a repository's custom hook class. It validates that the class can be loaded and is a subclass of evn.custom_hook.CustomHook, then writes the custom-hook-class configuration file property to the writable configuration file for the repository. This is the recommended way to alter a repository's custom hook class.

2.2.4 evnadmin get-repo-custom-hook-class

Displays the current custom hook class for a given repository.

2.2.5 evnadmin verify-path-matches-blocked-file-extensions-regex

This command is used to verify that a given path matches the blocked-file-extensions-regex configuration value for a given repository. It should be used to test that a given path is properly matched by the regex whenever making changes to it.

(This is a helper command that has been added for convenience. The only way to test the regex is correctly matching what you expect, without this command, is to manually try commit files to the repository.)

2.2.6 evnadmin verify-path-matches-max-file-size-exclusion-regex

This is a helper command similar to the one above that allows an administrator to verify that a given path is matched by the max-file-size-exclusion-regex configuration property. It should be used to verify that a given path is properly matched by a regex whenever making changes to said configuration property.

2.3 Configuration Commands

These commands interact with the configuration files used by Enversion.

3 Repository Layout: Single vs Multiple Component

Enversion has built-in support for two types of repository layouts: single-component layout and multiple-component layout. This is also referred to as component depth.

A single-component repository has a depth of 0 and would have a layout where the standard tags, branches and trunk directories are at the root of the repository:

```
/tags/
      /v1.0/
      /v1.1/
/branches/
      /v1.x/
/trunk/
```

Figure 3 Example Single-Component Layout

A multi-component repository has a depth of 1 and would have a layout where the tags, branches and trunk directories live one level under the root directory – with the actual root directory being the name of the component:

```
/widget/
      /tags/
            /v1.0/
            /v1.1/
      /branches/
            /v1.x/
      /trunk/
/gadget/
      /tags/
            /v2.0/
      /branches/
            /v2.x/
      /trunk/
```

Figure 4 Example Multiple-Component Layout

4 Configuration Reference

4.1 [main]

4.1.1 blocked-file-extensions-regex

Data Type	String
Default Value	`\.(com|ocx|mdb|dll|war|jar|so|exe|o|bin|iso|zip|tar|tgz|dat|tar\.gz|msi|msp|7z|pkg|rpm|nupkg|deb|dmg)$`
Since	v0.2.15
Description	If set, a case-insensitive regular expression search is performed against each new path in a commit, and if a match is found, that path is flagged with the *BlockedFileExtension* error. This functionality can be disabled with an empty string, e.g.: `blocked-file-extension-regex =` Use the evnadmin verify-path-matches-blocked-file-extensions-regex command once you have changed this value to verify that the regular expression is behaving as you expect.

Table 6 Configuration: blocked-file-extension-regex

4.1.2 custom-hook-classname

Data Type	String
Default Value	`evn.custom_hook.DummyCustomHook`
Since	v0.2.14
Description	This is the name of a Python class that Enversion will create an instance of during pre and post-commit hooks. It is a literal string value that should represent a fully-qualified class name (i.e. include module prefix). The Python environment should be set up such that Enversion can import the containing module, as this is how the class name is resolved. (See `evn.util.load_class` for the exact code used to load the class.) The value of this property must always resolve to a valid class name that derives from `evn.custom_hook.CustomHook`, which is why the default value refers to a dummy hook class that doesn't do anything in its pre_commit and post_commit callbacks. As an invalid class name will prevent hooks from running, it is recommended that this property be set via the `evnadmin set-repo-custom-hook-class` command.

Table 7 Configuration: custom-hook-classname

4.1.3 max-file-size-in-bytes

Data Type	Integer
Default Value	`26214400 (25MB)`
Since	0.2.7
Description	When set, blocks any file from being added to the repository that is over the max size, and blocks any existing file from growing past the maximum size. Exclusions can be added via the max-file-size-exclusion-regex configuration property. This functionality can be disabled with an empty string, e.g.: `max-file-size-exclusion-regex =`

Table 8 Configuration: max-file-size-in-bytes

4.1.4 max-file-size-exclusion-regex

Data Type	String
Default Value	<no value>
Since	v0.2.16
Description	If set, files matching this pattern are excluded from the max file size restrictions. An administrator would set this configuration property to match a file (or set of files via a wild card or extended regex using |) that is permitted to grow past the maximum file size (currently defaults to 25MB). Use the evnadmin verify-path-matches-max-file-size-exclusion-regex command once you have changed this value to verify that the regular expression is behaving as you expect.

Table 9 Configuration: max-file-size-exclusion-regex

5 Errors, Warnings and Notes

5.1 Errors

5.1.1 BlockedFileExtension

Format	“blocked file extension”
Controlled By	`blocked-file-extensions-regex`
What It Means	The affected file matched the regex specified above.

Table 10 Error: BlockedFileExtension