Enversion Administration Guide

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
v0.2.19+	06-May-2015	Added notes on v0.2.17, v0.2.18 and v0.2.19. Begin work on documentation for v0.2.20 (root hints).	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 v0.2.20 [IN DEVELOPMENT]

1.1.1 Root Hints Support

This release adds initial support for root hints.

1.2 v0.2.19

1.2.1 Fix Python 2.6 Compatibility

This release fixes some general compatibility issues with Python 2.6.

1.3 v0.2.18

1.3.1 Fix Root Handling Bug

This release fixes a critical bug with regards to root handling. Related github issues:

https://github.com/enversion/enversion/issues/24

https://github.com/enversion/enversion/issues/35

The underlying issue presented itself in the following situation: a known root was being copied to another known root's subtree. For example, presuming trunk and /branches/1.0.x are known roots (i.e. they have evn:roots entries):

```
A /branches/1.0.x/trunk (from /trunk:926)
```

The erroneous code path was not treating copies different from renames. If this commit were a rename (e.g. /trunk was also deleted in the same commit), then the root entry for /trunk would be removed from that commit's revision property. That logic was also being applied for commits that were simply copying roots, too.

1.3.2 Implement Show Rev Prop Command

This is a helper command that conveniently prints all revision properties for a given revision in JSON format. See section *2.6.1 evnadmin show-rev-prop* for more information.

1.4 v0.2.17

1.4.1 Initial PyPI Support

This release only contained minor (non-functional) changes to the source code (setup.py) required in order to upload Enversion to PyPI.

1.5 v0.2.16

1.5.1 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.

1.5.2 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

1.6 v0.2.15

1.6.1 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

1.6.2 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

1.7 v0.2.14

1.7.1 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 Read-only Commands

Repositories can be set read-only by administrators via these commands. When a user attempts to commit to a repository flagged as read-only, they are presented with an error message.

2.2.1 evnadmin set-repo-readonly

This command allows a repository to be set read-only with an optional, customizable error message (e.g. "This repository is currently undergoing maintenance and is in a read-only state until 10:00AM EST.").

2.2.2 evnadmin unset-repo-readonly

Clears a previously 'set-repo-readonly' command and allows commits against the repository to begin again.

2.2.3 evnadmin is-repo-readonly

Prints "yes" if a repository is currently marked as read-only, "no" if not. This information is also stored in the revision 0 revision property evn:readonly.

2.3 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.

2.3.1 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.

2.3.2 evnadmin get-repo-component-depth

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

2.3.3 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.

2.3.4 evnadmin get-repo-custom-hook-class

Displays the current custom hook class for a given repository.

2.3.5 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.)

2.3.6 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.

2.4 Configuration Commands

These commands interact with the configuration files used by Enversion.

2.4.1 evnadmin dump-config

2.4.2 evnadmin dump-default-config

2.4.3 evnadmin dump-modified-repo-config

2.4.4 evnadmin dump-repo-config

2.4.5 evnadmin show-possible-config-file-load-order

2.4.6 evnadmin show-possible-repo-config-file-load-order

2.4.7 evnadmin show-actual-config-file-load-order

2.4.8 evnadmin show-actual-repo-config-file-load-order

2.4.9 evnadmin show-writable-repo-override-config-filename

2.5 Root Management Commands

2.5.1 evnadmin show-roots

2.6 Miscellaneous Commands

2.6.1 evnadmin show-rev-prop

This command provides a convenient way to display all the revision properties for a given revision. The output will be in JSON format. For example:

```
bash-4.1$ evnadmin show-rev-props -r1415 ACME
Showing revision properties for repository 'ACME' at r1415:
```

```
{'evn': {'errors': {'/branches/ACME-1.0.X/': ['known root subtree path
copied to valid root path']},
         'roots': {'/trunk/': {'created': 1}}},
'svn': {'author': 'trent',
         'date': '2015-02-12T08:17:25.678052Z',
         'log': 'Create branch ACME-1.0.X'}}
```

Table 6 Example output from evnadmin show-rev-props

The equivalent command using svn would be:

```
-bash-4.1$ svn proplist -v --revprop -r1415 file://`pwd`/ACME
Unversioned properties on revision 1415:
  evn:errors
     {'/branches/ACME-1.0.X/': ['known root subtree path copied to
valid root path']}
  evn:roots
    {'/trunk/': {'created': 1}}
  svn:author
    trent
  svn:date
    2015-02-12T08:17:25.678052Z
  svn:log
    Create branch ACME-1.0.X
```

Table 7 Example output from svn proplist

Primary design motivation for adding this functionality was that the unit tests for https://github.com/enversion/enversion/issues/35 needed to be able to programmatically parse revprop output. Having JSON output, where the propname prefix (the part that comes before ':', e.g. 'evn' for 'evn:roots') is a key to another dict of key/values, is useful, as it simplifies the parsing logic.

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 Roots

Enversion works by tracking roots. A root is the "root path" of a trunk, branch or tag. Tags are always read-only, whereas trunks and branches can be modified.

Roots can be created in one of two ways:

- Creating a directory named trunk.
- Copying a known root path to a valid root path.
- Renaming a known root to a valid root path.
- Adding a root hint.

A known root path is a path for which there is an evn:roots entry at a given revision. A valid root path is a path that represents a valid root name, but for which there is no evn:roots entry at a given revision.

Enversion uses the standard Subversion conventions for classifying trunk, branches and tags:

- A trunk is any directory named 'trunk'.
- A branch is any directory named 'branches/<branch-name>/'.
- A tag is any directory named 'tags/<tag-name>/'.

The names 'trunk', 'branches' and 'tags' are considered reserved names, and thus, cannot be used as names for branches or tags.

There can only ever be one known root for a given path – nesting of roots is not permitted. For example, given /branches/1.0.x/foo as a known root, you cannot also have /branches/1.0.x/foo/bar as a root.

In the real world, branches are often not created correctly, for example:

- svn cp ^/trunk/widget ^/branches/widget-1.x
    - This will not create a root for /branches/widget-1.x because the known root path /trunk was not used, the subtree /trunk/widget was used instead.
    - Not having /branches/widget-1.x as a known root means that no further branches can be created from it, nor can it be tagged.

- svn mkdir /branches/widget-1.x
    - In this example a user manually creates the branch directory. This is usually followed by manually copying over file contents from the source branch (e.g. copying trunk files via Windows Explorer).
    - Again, as /branches/widget-1.x does not get registered as a known root, no further root actions (creating subsequent branches or tags) will be permitted against that path.

For repositories created with Enversion enabled from the start, this is less of a problem, as Enversion will automatically block these sorts of commits. However, if Enversion is ever disabled and commits like this are let through, or Enversion is being enabled against an existing repository that never had any protection in place, then this situation can arise.

A third situation that can arise is where a repository used a different naming convention to refer to their "trunk" paths. The FreeBSD Subversion repository is an excellent example of this – they use a path called '/head' to represent their trunk, and all branches are created from it as normal.

In order to deal with these situations, Enversion supports a concept known as "root hints".

Root hints are set via a revision property on revision 0 named evn:root_hints. This property is in Python dictionary format. The keys represent revision numbers, the values are lists of two-item tuples, the first value is the path name created in that revision that should be treated as a root, the second is one of the following strings indicating the root type: 'trunk', 'tag' or 'branch'. E.g.

```
{
      1: [
            ('/head/', 'trunk'),
      ],

      383: [
            ('/branches/1.0.x/', 'branch'),
      ],
}
```

Figure 5 Example evn:root_hints

The root hints are only consulted when Enversion is analysing a repository. They are hints because they only alter Enversion's classification of a new path being created or copied – they do not override the normal root protection and invariant logic that takes place before a modification is made to evn:roots for a given revision.

Additionally, altering a repository's root hints will require re-analysis of the repository from the lowest revision onward. This will require setting the repository to read-only for the duration of the analysis, which can be done via the `set-read-only` and `unset-read-only` commands.

A helper command called `evnadmin suggest-root-hint` is provided that aims to take some of the guess work out of setting root hints correctly. This command can be called against a path that the administrator wants to have classed as a root, and Enversion will attempt to infer the root hint required to affect such a change.

Dev checklist:

1. evnadmin set-read-only -m <message> <REPO>
2. evnadmin unset-read-only <REPO>
3. evnadmin suggest-root-hint -p /head <REPO>
4. evnadmin add-root-hint -r <rev> -p <path> <REPO>
5. evnadmin remove-root-hint -r <rev> -p <path> <REPO>
6. evnadmin add-branches-basedir –p <path> <REPO>
7. evnadmin add-tags-basedir –p <path> <REPO>
8. evnadmin show-root-hints <REPO>
9. evnadmin validate-root-hints <REPO>

4.1 Example: FreeBSD Repository

The FreeBSD Subversion repository uses a non-conventional layout for tags, trunks and branches. The main 'trunk' is a path called /head/, which was added in revision 1.

Subsequent branches of /head/ were created in the /stable/ directory, e.g. /stable/1/, /stable/2/. Tags were created by copying /stable/<rev>/ branches into /releng/<relno>/, e.g. /stable/8/ was copied to /releng/8/ which represented the FreeBSD 8.0-RELEASE.

4.1.1 Add Root Hint for /head/

We need to tell Enversion that when /head/ was created in revision 1, it should be considered as though a trunk was being created. We do this as follows:

```
evnadmin add-root-hint --path /head/ --root-type trunk freebsd
```

We can verify the root hint as follows:

```
evnadmin show-root-hints freebsd
```

Which will indicate a structure along the following lines has been set:

```
{
      1: [
            ('/head/', 'trunk'),
      ],
}
```

When we analyze the repository now, Enversion will detect the creation of the /head/ directory as a "root creation", and create a corresponding evn:roots entry for it:

```
evnadmin analyze freebsd
...
...
^C
```

We enter a control-C to stop the analysis. Now, if we look at the roots, we can see /head/ has been added:

```
evnadmin show-roots freebsd
evnadmin root-info –p /head/ freebsd
```

Now, when /head/ is copied into /stable/<N>/, new roots will be created as if they were branches.

4.1.2 Adding /stable/ as a branches directory and /releng/ as a tags directory

We would like to leverage Enversion's ability to detect when a branch or tag has been created incorrectly – that is, not copied from a known root, or, in the case of a tag, not copied cleanly from a known root.

Enversion normally provides this logic by simply testing the base directory of a commit against the pattern 'branches' or 'tags', e.g. the base directory of /branches/1.0.x/ is 'branches', thus, we infer /branches/1.0.x/ as a valid root path.

Because FreeBSD uses /stable/ instead of /branches/, and /releng/ instead of /tags/, we need to add a branches base directory and tags base directory for each of them, respectively:

evnadmin add-branches-basedir –p /stable/ freebsd

evnadmin add-tags-basedir –p /releng/ freebsd

These are simply set as lists against the revision 0 revprop named evn:branches_basedirs and evn:tags_basedirs.

Now, if we try and create a directory manually under /stable/ or /releng/, Enversion will block this as expected. Additionally, because /releng/ is classed as a tags base directory, roots under it will be treated as tags, which means they will be immutable and also prevented from copying directly.

Configuration Reference

4.2 [main]

4.2.1 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 8 Configuration: blocked-file-extension-regex

4.2.2 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 9 Configuration: custom-hook-classname

4.2.3 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 10 Configuration: max-file-size-in-bytes

4.2.4 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 11 Configuration: max-file-size-exclusion-regex

4.2.5 readonly-error-message

Data Type	String
Default Value	This repository cannot be committed to at the present time because an administrator has marked it as read-only.
Since	*v0.2.20 [IN DEVELOPMENT]*
Description	This is the default error message seen by the user when they attempt to commit to a repository that has been marked read-only via the

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 12 Error: BlockedFileExtension