

Wigwam 4

Release Notes - Version 4.0

This document provides the Release Notes to accompany the release of Wigwam 4.

These *Release Notes* list:

- Key differences between Wigwam 4 and earlier versions.
- Known limitations
- Future enhancements to Wigwam

For more details on bug fixes and feature requests, refer to the Wigwam 4 *Interim Release Notes* and `ChangeLog` files.

Key Differences between Wigwam 3 and Wigwam 4

This section summarizes the differences between Wigwam 3 and Wigwam 4.

Wigwam base

Streamlined wigwam-base

wigwam-base now contains only the package and environment machinery.

`servicectl` and `pubtool` are now separate packages.

The now smaller wigwam-base tarball is uuencoded and placed *inside* wigwam-bootstrap to simplify the bootstrap process and to increase its portability.

Cleaning uninstall

The `BASE pre-install` and `post-install` methods snapshot `$PLAYPEN_ROOT/ext/` to determine what files were created by the install. Wigwam then uses this in the `BASE uninstall` method to remove files that:

- were created by the install, and
- haven't been modified since install time

It also helps to identify packaging bugs that were previously unnoticed due to files left over from a previous installation.

As of 4.1.0, Wigwam reference counts files and directories in `ext/` and removes them if uninstall causes the reference count to hit zero.

setup-env

autogen now creates setup-env and unsetup-env, so you don't need to manage them under revision control. If you migrate a Wigwam 3 project to Wigwam 4 (see *Migration Guide*), you should remove any setup-env that was checked out for the Wigwam 3 project.

Packaging

The packaging system - **packaget1** - is more extensible and configurable.

Package meta file naming conventions

Wigwam 4 uses a different naming convention for package meta files.

Wigwam 3 naming

`$PACKAGE/$VERSION/$PACKAGE-$VERSION.file`

Wigwam 4 naming

`$PACKAGE/$VERSION/file`

Package methods

Wigwam 4 now organizes package installation into a series of install phases. Wigwam provides default methods for these packaging phases, but package developers may provide their own per package methods to handle these phases.

As with Wigwam 3, for raw-type packages, the package developer needs to provide custom scripts to handle the build, install, and optional uninstall phases.

Besides the default or custom build and install methods, the package developer can provide pre- and post-method hooks assembled as scripts with the package. For instance, the package developer can provide a pre-build, post-build, pre-install, and post-install scripts.

The package methods default dispatch rules are:

- Pre methods
 - BASE package class method, if any
 - type specific class method, if any
 - package any-version method, if any
 - package specific-version method, if any
- Methods
 - package specific-version method, if any
 - package any-version method, if any
 - type specific class method, if any
 - BASE package class method, if any
- Post methods
 - package specific-version method, if any
 - package any-version method, if any
 - type specific class method, if any

- BASE package class method, if any

A method can halt the dispatch chain by returning a non-zero exit status. The method can use the special exit status of 99 to indicate a successful operation that halts further packaging system dispatch.

A package of type X has an implicit dependency on "build-package-X".

- wigwam-base provides build-package-make for make-type packages
- wigwam-base provides build-package-raw for raw-type packages
- servicectl provides build-package-service for service-type packages

Dependencies and pre dependencies

The Wigwam 3 predep package file is deprecated in Wigwam 4, and is merged with dep files. As such, you can't selectively override predep; instead you must override dep.

Unification of in-project source code and packages

Wigwam 4 doesn't include the projectctl API that was in Wigwam 3. This is because Wigwam 4 expects all project source to be organized into packages. Wigwam 4 lets you place the source code for a project in `$PLAYPEN_ROOT/src/$PACKAGE-local`, then provide the meta files for that packaged source code in `$PLAYPEN_ROOT/packages/$PACKAGE-local`.

In-project packages have two distinctive characteristics:

- They have the special version `local`
- They are assumed to depend on all packages in `etc/project-packages`
- When listed in `etc/in-project-packages`, they aren't listed with any version number.

Wigwam interprets the version as `local`.

`packagectl update-packages` has these characteristics:

1. Installs/upgrades all packages in `etc/project-packages`
2. Uninstalls all existing local packages
3. Installs all local packages listed in `etc/in-project-packages`. `packagectl update-packages` always uninstalls then installs packages with the version `local`. Wigwam always "assumes" local versions have changed.

Package installed to different playpen location

Wigwam now downloads package meta files to `ext/packages/$PACKAGE/$VERSION`; package source code gets unpacked to `ext/src/$PACKAGE/$VERSION`.

Both Wigwam 3 and Wigwam 4 packages should use the `EXT_PACKAGEDIR` environment variable, which is set in both major versions, to refer to the download directory. Custom build/install scripts that reference the `EXT_PACKAGEDIR` explicitly are incorrect, and will break.

Package management planning

The package management system and package archive provide mutual support:—the archives now contain sufficient information for the packaging system to construct an install plan before it downloads any packages.

- The `packagectl` command provides a `--dry-run` capability, wherein the user can determine what will happen if `$PACKAGE` and/or `$VERSION` were installed. The command format is as follows:

```
packagectl ACTION --dry-run ARGS
```

- Archives now contain the files `dependencies`, `conflicts`, `provides`, and `types` that get constructed automatically by the `wigwam-packaging-utils` module. Wigwam downloads these metadata files from the archive at the same time that it downloads the `package-list` files into the playpen. Wigwam compiles all these files into local playpen databases for fast manipulation.

Major advantages of this integration of package management and package-archives include the following capabilities:

- `backtrack` in the plan construction to overcome conflicts
- `search provides` to satisfy a dependency.

Update-local eliminated

Wigwam no longer requires or supports the `packagectl update-local` API. Instead, `wigwam-base` signs each file that it places outside of `ext/`. If these files haven't been modified upon upgrade, `wigwam-base` replaces these files; otherwise, it places a `.dist` version of the file beside the modified file. The playpen maintainer is required to merge the differences manually.

servicectl

`wigwam-base` no longer provides `servicectl`. `servicectl` is now a separate module that you can install as a package with `packagectl` commands. Its `servicectl` command API is consistent with Wigwam 3.

The `servicectl` API no longer provides the `servicectl check-config` command.

pubtool

`wigwam-base` no longer provides `pubtool`. `pubtool` is now a separate module that you can install as a package with `packagectl` commands.

Its `pubtool` API is consistent with Wigwam 3. Because the `pubtool` is no longer in `wigwam-base`, `wigwam-base` has no concept of role or cluster. These entities become significant only after you install `pubtool` in the project. At that point, you can specify role- and cluster-specific configuration information in the respective scoped configuration files, which Wigwam inserts into the construction chain.

Environment construction

A package developer can change the way the environment is constructed by manipulating an `sysv-init` style directory, `ext/etc/config-order/`. Normal override rules apply, as described in *Overrides*, below; hence, a project developer may override this via `etc/config-order/`

When you install `wigwam-base`, `servicectl`, and `pubtool`, the resulting default set of files and the source order exactly matches the order specified for Wigwam 3. See <http://www.wigwam-framework.org/doc/wigwam.html#SCRIPTS-TO-LOAD-CONFIGURATIONS>

Overrides

Wigwam 4 extends the package overrides feature to allow overrides for all project components. This overrides capability lets you customize behavior and lets Wigwam 4 aggressively set defaults.

File overrides

Wigwam looks for files in the following order, and uses the first one that it finds.

```
myfile.local (local override)
myfile (revision controlled file)
/ext/./myfile (non-revision-controlled default)
```

Note that any file with `.local` appended to it doesn't get checked into revision control; hence, it doesn't get communicated to other users of a project.

Package overrides

To override packages, place the desired override in `packages/package-version/X`

Packagectl planner

To override the `packagectl` planner, place a new binary in `libexec/packagectl/plan`

Unsanitized variables

To override the set of unsanitized variables, place a file in `etc/unsanitized`

Known limitations

The `packagectl` API should have a command `packagectl list`. That command should have an argument that lets the user view either the currently installed packages (i.e., the default `ext/installed-packages` or the `etc/project-packages`).

Workaround: To view the files that are currently installed in the playpen, look at the `$PLAYPEN_ROOT/ext/etc/installed-packages`. Note that this file may change any time you run `autogen.sh`, `packagectl install`, `packagectl uninstall`, `packagectl upgrade`, or `update-packages`.

To view the list of files that are currently required for the project, look at `$PLAYPEN_ROOT/etc/project-packages`. Note that this file may change any time you run `packagectl install`, `packagectl uninstall`, or `packagectl upgrade`.

The `servicectl check-config` functionality doesn't currently exist as an API in Wigwam 4.

Workaround: The user may attempt to start services and, based on the error message, determine that one or more required service-specific parameters aren't set. The user can look at `ext/$PACKAGE/$VERSION/config` to determine required environment variables for that service, then set those variables appropriately in the relevant project configuration files.

The `make-package-md5sums` program that ships with `wigwam-packaging-utils` 3.1 is currently incompatible with Wigwam 4, as it looks for the Wigwam 3 *package* files naming format as one of its arguments.

Workaround: Use one of the generic md5sums generating tools to produce md5sums for each file listed in `$PACKAGE/$VERSION/files`. An example is the Linux `md5sum` utility.

Future enhancements

Packaging utils integration

`wigwam-packaging-utils` will be integrated with `wigwam-base` so that a local archive can be done entirely from `wigwam-bootstrap`.

Extensible archive types

An abstracted archive API that lets the developer define new archive types.

Wigwam 4

Interim Release Notes

This document provides the Release Notes that list the changes since the last major release of Wigwam, which is Wigwam 4. For more details, refer to the Wigwam 4 ChangeLog file.

4.3.1 - 2/24/2004

- Ensure that `cpio -a` and `-m` options are mutually exclusive.
- Allow `load_config_order` to be relocatable.

4.3.0 - 2/11/2004

- Have `wigdo` evaluate argument list.
- Always use `Wigwam::DB::Dumb`
- Expand patch files with the `patch` filename extension.
- Adjust `cpio` options in `BASE/unpack` to facilitate incremental builds.

4.2.0 - 2/4/2004

- Ignore package supplied `installed_files` file.
- Don't cleanup `ext/src/` for local packages.
- Don't delete `installed_files` file before reinstall.
- Match Wigwam 3 `perl_makefile_options` case in `make/build/` directory.

4.1.1 - 2/2/2004

- Provide environment variable to override `packagectl auto-sync` behavior
- Mark `WIGWAM_PACKAGECTL_NO_AUTO_SYNC` as unsanitized by default.
- Allow `apply_patch` to signal failure.
- Provide environment variable for switchable auto-cleanup of `ext/src/`
- Mark `MANPATH` unsanitized by default.

4.1.0 - 1/19/2004

- Modify packaging planner overhaul so that provides generates conflicts.
- Sort providers by version when inverting provides database.
- Atomically write backing file in `Wigwam::DB::Dumb`
- Match Wigwam 3 behavior with respect to `move_files_up` in `BASE/unpack/` directory.
- Atomically update `load-config-order` in `load-environment`.
- Mark `PERL5LIB` interactive due to `Wigwam::DB`
- Reference counting files in `ext/` for `uninstall`. Require a `--force-rebuild` on `publish`.
- Implement `reinstall` as `install/uninstall` pair. To facilitate this, allow for temporary double installation of packages.
- Provide more restrictive `regex` for finding in-project defined package versions. Require that package version either be `local`, or begin with a digit. This prevents confusion of packages wherein names are prefixes of each other.
 - Valid formats: `local`, `2.1-1`
 - Invalid formats: `A`, `A.1`
- Delete removed packages as part of `packagectl update-packages`.

4.0.2 - 12/23/2003

- Track Wigwam 3 changes in `packagectl/make/` methods with respect to evaluation
- Honor `dont_expand_tarballs` in `packagectl/BASE/unpack`
- Honor `pre_build_script` and `post_build_script` in `packagectl/make/build`
- Use `LDFLAGS` for `-L`, `LIBS` for `-l`
- Make `env-api` available to package methods via `BASE/functions`
- Use `gnumake` to build `wigwam-base` during bootstrap

4.0.1 - 12/15/2003

- Don't compute `installed_files` if provided by package or project override
- Don't check `setup-env` or `unsetup-env` into `CVS`