zopetoolkit Documentation

Release 1.0.8

Apr 15, 2019

Contents

The Zope Toolkit 1.0 release is the first release of the Zope Toolkit. The Zope Toolkit really is just a collection of libraries managed together by the Zope developers. We typically treat each library independently, so you would like to look at the CHANGES.txt in each library for updates. Here we note larger changes, especially ones that affect multiple libraries.

CHAPTER 1

Installation

The Zope Toolkit cannot be installed directly except as individual libraries (such as `zope.component`). To install it you typically would install a framework or application that makes use of these libraries. Examples of such projects are BlueBream, Grok and Zope 2.

If you want to use the Zope Toolkit KGS, you can use the buildout extends mechanism (replace 1.0 by the desired version):

```
[buildout]
extends = http://download.zope.org/zopetoolkit/index/1.0/ztk-versions.cfg
```

You can also copy the file locally or additionally extend the zopeapp-versions.cfg file from the same location.

Frameworks and applications have their own set of install instructions. You should follow these in most cases.

CHAPTER 2

Python versions

The ZTK 1.0 release series supports Python 2.4 up to Python 2.6. Neither Python 2.7 nor any Python 3.x series is supported.

CHAPTER 3

News

The 1.0 release of the Zope Toolkit contains a number of refactorings that are aimed to clean up dependencies between pieces of code. Many packages in `zope.app` have had their code moved to existing or newly created packages in the `zope` namespace. These new packages do generally not contain user interface code (typically what's in `.browser`), and have much clearer dependency relationships as a result.

Backwards compatibility imports have been left in place so that your existing code should still work. In some cases you will have to explicitly add dependencies to a `zope.app.` to your code, as due to the cleanup they might not come in automatically anymore due to indirect dependencies; if you see an import error this is probably the case.

We recommend you update your existing code to import from the new packages if possible. The transition support and `zope.app.*` support is limited: the legacy support will be officially retired from the ZTK in subsequent releases.

CHAPTER 4

Migration issues

4.1 zope.app.keyreference -> zope.keyreference

This package was renamed to `zope.keyreference` and all its functionality was moved to the new one. The new package contains a little workaround for making old persistent keyreferences loadable without `zope.app.keyreference` installed, so the latter one is not needed at all anymore. Still review your code for any imports coming from `zope.app.keyreference` and modify it to use `zope.keyreference` instead.

4.2 zope.app.intid -> zope.intid

The non-UI functionality of these packages was moved to `zope.intid` with backwards compatibility imports left in place. Review your imports from `zope.app.intid` to see whether they cannot come directly from `zope.intid` instead.

4.3 zope.app.catalog -> zope.catalog

The non-UI functionality of these packages was moved to `zope.catalog`. Review your imports from `zope.app.catalog` to see whether they cannot come directly from `zope.catalog` instead.

4.4 zope.app.container -> zope.container

The non-UI functionality of these packages was moved to `zope.container`. Review your imports from `zope.app.container` to see whether they cannot come directly from `zope.container` instead.

In addition, the exceptions used by `zope.container` were changed, so if your code catches them, you need to review it:

- The `DuplicationError` in `setitem` was changed to `KeyError`.

- The `UserError` in `NameChooser` was changed to `ValueError`.

4.5 zope.app.component -> zope.security, zope.site

The implementation of the `<class>` ZCML directive moved from this package to `zope.security`. Packages that relied on `zope.app.component` to obtain this directive should declare a direct dependency on `zope.security`, and it may be possible to lose the dependency on `zope.app.component` altogether.

Non-UI site related functionality has been moved to the `zope.site` package. with backwards compatibility imports left in place. Review your imports from `zope.app.component` to see whether they cannot come directly from `zope.site` instead.

4.6 zope.app.folder -> zope.site, zope.container

The implementation of the `zope.app.folder.Folder` class has moved to `zope.site.folder` instead, with backwards compatibility imports left in place. Review your imports from `zope.app.folder` to see whether they cannot come directly from `zope.site` instead. In addition, `Folder` is an `IContainer` implementation that also mixes in site management functionality. If such site management support is not necessary, in some cases your code does not need `Folder` but may be able to rely on a `Container` implementation from `zope.container` instead.

A base class with the implementation of the container-like behavior of `Folder` has moved to `zope.container` (and `zope.site` uses this for its implementation of `Folder`). This is not normally something you should need to retain backwards compatibility.

4.7 zc.copy -> zope.copy, zope.copypastemove, zope.location

The pluggable object copying mechanism once developed in the `zc.copy` package was merged back into `zope.location`, `zope.copypastemove` and the new `zope.copy` package. The `zope.copy` package now provides a pluggable mechanism for copying objects from `zc.copy` and doesn't depend on anything but `zope.interface`. The `zope.copypastemove` uses the `copy` function from `zope.copy` in its `ObjectCopier`.

The `zope.location` now provides an `ICopyHook` adapter that implements conditional copy functionality based on object locations, that old `zope.location.pickling.CopyPersistent` used to provide. Note, that if you don't use ZCML configuration of `zope.location`, you may need to register `zope.location.pickling.LocationCopyHook` yourself.

The `zope.location.pickling.locationCopy` and `zope.location.pickling.CopyPersistent` are now deprecated in favor of `zope.copy` and were replaced by backward-compatibility imports. See `zope.copy` package documentation for information on how to use the new mechanism.

The new version of the `zc.copy` package now only contains backward-compatibility imports and is deprecated. `zope.copy` should be preferred for new developments.

4.8 zope.app.security refactoring

The `zope.app.security` package was finally refactored into a few small parts with less dependencies and more clear purpose.

The implementation of the <module> ZCML directive moved from this package to zope.security. Packages that relied on zope.app.security to obtain this directive should declare a direct dependency on zope.security, and it may be possible to lose the dependency on zope.app.security altogether.

The protectclass module in this package has moved to zope.security, with backwards compatibility imports left in place. Review your imports from zope.app.security to see whether they cannot come directly from zope.security instead.

All interfaces (*IAuthentication*, *IUnauthenticatedPrincipal*, *ILoginPassword* and so on.) were moved into a new zope.authentication package, as well as several utility things, like *PrincipalSource* and *checkPrincipal* function. The new package has much less dependencies and defines an abstract contract for implementing authentication policies. While backward compatibility imports are left in place, it's strongly recommended to update your imports to the zope.authentication.

The *global principal registry* and its ZCML directives are moved into a new zope.principalregistry package with backward-compatibility imports left in place. If your application uses global principals, review your code and ZCML configuration to update it to the new place.

The *local permission* functionality was moved into a new zope.app.localpermission package. This functionality is a part of Through-The-Web development pattern that seems not to be used and supported much by Zope Toolkit and Application anymore, so it can be considered deprecated. However, it can serve as a great example of TTW-related component.

The *Permission vocabularies* and standard protections for Message objects and *__name__*, *__parent__* attributes as well as some common permissions, like *zope.View* and *zope.ManageContent* were merged into *zope.security*.

The adapters from zope.publisher's *IHTTPCredentials* and *IFTPCredentials* to the *ILoginPassword* were moved into zope.publisher, thus making zope.authentication a dependency for zope.publisher.

The original zope.app.security package now only contains several deprecated or application-specific permission definitions, python module protections, that are only likely to be needed with deprecated Through-The-Web development pattern, and ZMI-related browser views (login.html, zope.app.form view for PrincipalSource and so on), as well as backward-compatibility imports. So, if you're not using TTW and/or standard ZMI browser views, you probably should review update your imports to a new places and drop dependency on zope.app.security to reduce package dependencies count.

Other packages, that used zope.app.security, like zope.securitypolicy are either already adapted to the changes or will be adapted soon.

4.9 zope.app.publisher refactoring

The zope.app.publisher package was also refactored into smaller parts with less dependencies and clearer purpose.

The browser resources mechanism (mostly used for serving static files and directories) was factored out to the new zope.browserresource package. It was also made more pluggable, so you can register specific resource classes for some file extensions, if you need special processing. One of the example is the new zope.ptresource package, where the PageTemplateResource was moved, another example is z3c.zrtresource package that was adapted to automatically use ZRT resource class for files with .zrt extensions.

Browser menu mechanism was moved into a new zope.browsermenu package with no further changes.

ZCML directives for easy creation of browser views (the browser:page directive and friends) was moved into a new small package, zope.browserpage. Also, the directives don't depend the menu mechanism now and will simply ignore "menu" and "title" arguments if zope.browsermenu package is not installed.

The IModifiableBrowserLanguages adapter was moved into zope.publisher along with several ZCML security declarations for zope.publisher classes that used to be in zope.app.publisher.

ZCML registrations for `IXMLRPCPublisher` adapter for containers was moved into the `zope.container`, because the actual adapters code were already in `zope.container` and registered there as `IBrowserPublisher` adapters. However, both adapters and their ZCML registrations will probably move elsewhere when we'll be refactoring `zope.container`.

Several parts are left in `zope.app.publisher` untouched:

- `Browser Skins` vocabulary.
- `date` field converter for `zope.publisher`'s form values conversion mechanism.
- `ManagementViewSelector` browser view (ZMI-related part).
- `xmlrpc:view` directive for publishing XML-RPC methods.

The latter, `xmlrpc:view` directive is generally useful, so it may be moved into a separate package in future, however there are no clear decision about how to move XML-RPC and FTP-related things currently.

4.10 Password managers extracted from zope.app.authentication

The *IPasswordManager* interface and its implementations were extracted from `zope.app.authentication` into a new `zope.password` package to make them usable with other authentication systems, like `z3c.authenticator` or `zope.principalregistry` or any custom one.

It basically depends only on `zope.interface`, so it can be really useful even in non-Zope environments, like `Pylons`, for example.

The *Password Manager Names* vocabulary is also moved into `zope.password`, however, it's only useful with `zope.schema` and `zope.component`, so you need them installed to work with them. They're listed in the "vocabulary" extra requirement specification.

4.11 ZODB 3.9 FileStorage native blob support

The FileStorage component of ZODB 3.9 used in Zope Toolkit 1.0 now supports blobs natively, so you don't need to use BlobStorage proxy for it anymore.

Thus, you can specify blob directory directly to FileStorage. If you use ZConfig, that means something like this:

```
<filestorage>
  path var/Data.fs
  blob-dir var/blobs
</filestorage>
```

instead of:

```
<blobstorage>
  blob-dir var/blobs
  <filestorage>
    path var/Data.fs
  </filestorage>
</blobstorage>
```

If you creating a storage from python, that means something like this:

```
storage = FileStorage('var/Data.fs', blob_dir='var/blobs')
```

instead of:

```
storage = BlobStorage('var/blobs', FileStorage('var/Data.fs'))
```

4.12 zope.dublincore permission renaming

The `zope.app.dublincore.*` permissions have been renamed to `zope.dublincore.*`. Applications using these permissions have to fix up grants based on the old permissions.