
zope.dublincore Documentation

Release 4.1.2

Zope Foundation and Contributors

Oct 05, 2017

Contents

1	Using <code>zope.dublincore</code>	3
1.1	Dublin Core Properties	3
1.2	Dublin Core metadata as content data	4
1.3	Time annotators	6
2	<code>zope.dublincore</code> API	9
3	Hacking on <code>zope.dublincore</code>	15
3.1	Getting the Code	15
3.2	Working in a <code>virtualenv</code>	15
3.3	Using <code>zc.buildout</code>	17
3.4	Using <code>tox</code>	18
3.5	Contributing to <code>zope.dublincore</code>	19
4	Indices and tables	21
	Python Module Index	23

Contents:

Dublin Core Properties

A dublin core property allows us to use properties from dublin core by simply defining a property as `DCProperty`.

```
>>> from zope.dublincore import property
>>> from zope.interface import implementer
>>> from zope.annotation.interfaces import IAttributeAnnotatable
>>> @implementer(IAttributeAnnotatable)
... class DC(object):
...     title = property.DCProperty('title')
...     author = property.DCProperty('creators')
...     authors = property.DCListProperty('creators')
>>> obj = DC()
>>> obj.title = u'My title'
>>> print(obj.title)
My title
```

Let's see if the title is really stored in dublin core:

```
>>> from zope.dublincore.interfaces import IZopeDublinCore
>>> print(IZopeDublinCore(obj).title)
My title
```

Even if a dublin core property is a list property we can set and get the property as scalar type:

```
>>> obj.author = u'me'
>>> print(obj.author)
me
```

`DCListProperty` acts on the list:

```
>>> obj.authors == (u'me',)
True
```

```
>>> obj.authors = [u'I', u'others']
>>> obj.authors == (u'I', u'others')
True
>>> print(obj.author)
I
```

Dublin Core metadata as content data

Sometimes we want to include data in content objects which mirrors one or more Dublin Core fields. In these cases, we want the Dublin Core structures to use the data in the content object rather than keeping a separate value in the annotations typically used. What fields we want to do this with can vary, however, and we may not want the Dublin Core APIs to constrain our choices of field names for our content objects.

To deal with this, we can use specialized adapter implementations tailored to specific content objects. To make this a bit easier, there is a factory for such adapters.

Let's take a look at the simplest case of this to start with. We have some content object with a *title* attribute that should mirror the Dublin Core *title* field:

```
>>> @implementer(IAttributeAnnotatable)
... class Content(object):
...     title = u""
...     description = u""
```

To avoid having a discrepancy between the *title* attribute of our content object and the equivalent Dublin Core field, we can provide a specific adapter for our object:

```
>>> from zope.dublincore import annotatableadapter
>>> factory = annotatableadapter.partialAnnotatableAdapterFactory(
...     ["title"])
```

This creates an adapter factory that maps the Dublin Core *title* field to the *title* attribute on instances of our *Content* class. Multiple mappings may be specified by naming the additional fields in the sequence passed to *partialAnnotatableAdapterFactory()*. (We'll see later how to use different attribute names for Dublin Core fields.)

Let's see what happens when we use the adapter.

When using the adapter to retrieve a field set to use the content object, the value stored on the content object is used:

```
>>> content = Content()
>>> adapter = factory(content)
>>> print(adapter.title)

>>> content.title = u'New Title'
>>> print(adapter.title)
New Title
```

If we set the relevant Dublin Core field using the adapter, the content object is updated:

```
>>> adapter.title = u'Adapted Title'
>>> print(content.title)
Adapted Title
```


Dublin Core fields which are not specifically mapped to the content object do not affect the content object:

```
>>> adapter.description = u"Some long description."
>>> print(content.description)

>>> print(adapter.description)
Some long description.
```

Using arbitrary field names

We've seen the simple approach, allowing a Dublin Core field to be stored on the content object using an attribute of the same name as the DC field. However, we may want to use a different name for some reason. The *partialAnnotatableAdapterFactory()* supports this as well.

If we call *partialAnnotatableAdapterFactory()* with a mapping instead of a sequence, the mapping is used to map Dublin Core field names to attribute names on the content object.

Let's look at an example where we want the *abstract* attribute on the content object to be used for the *description* Dublin Core field:

```
>>> @implementer(IAttributeAnnotatable)
... class Content(object):
...     abstract = u""
```

We can create the adapter factory by passing a mapping to *partialAnnotatableAdapterFactory()*:

```
>>> factory = annotatableadapter.partialAnnotatableAdapterFactory(
...     {"description": "abstract"})
```

We can check the effects of the adapter as before:

```
>>> content = Content()
>>> adapter = factory(content)

>>> print(adapter.description)

>>> content.abstract = u"What it's about."
>>> print(adapter.description)
What it's about.

>>> adapter.description = u'Change of plans.'
>>> print(content.abstract)
Change of plans.
```

Limitations

The current implementation has a number of limitations to be aware of; hopefully these can be removed in the future.

- Only simple string properties, like *title*, are supported. This is largely because other field types have not been given sufficient thought. Attempting to use this for other fields will cause a *ValueError* to be raised by *partialAnnotatableAdapterFactory()*.
- The CMF-like APIs are not supported in the generated adapters. It is not clear that these APIs are used, but content object implementations should be aware of this limitation.

Time annotators

Time annotators store the creation resp. last modification time of an object. We will use a simple `Content` class as our example.

```
>>> class Content(object):
...     created = None
...     modified = None
```

The annotations are stored on the `IZopeDublinCore` adapter. This dummy adapter reads and writes from/to the context object.

```
>>> from zope.component import provideAdapter
>>> from zope.dublincore.interfaces import IZopeDublinCore
>>> class DummyDublinCore(object):
...     def __init__(self, context):
...         self.__dict__['context'] = context
...
...     def __getattr__(self, name):
...         return getattr(self.context, name)
...
...     def __setattr__(self, name, value):
...         setattr(self.context, name, value)
>>> provideAdapter(DummyDublinCore, (Content,), IZopeDublinCore)
```

Created annotator

The created annotator sets creation and modification time to current time.

```
>>> content = Content()
```

It is registered for the `ObjectCreatedEvent`:

```
>>> from zope.dublincore import timeannotators
>>> timeannotators._NOW = 'NOW'
>>> from zope.component import provideHandler
>>> from zope.dublincore.timeannotators import CreatedAnnotator
>>> from zope.lifecycleevent.interfaces import IObjectCreatedEvent
>>> provideHandler(CreatedAnnotator, (IObjectCreatedEvent,))
>>> from zope.event import notify
>>> from zope.lifecycleevent import ObjectCreatedEvent
>>> notify(ObjectCreatedEvent(content))
```

Both `created` and `modified` get set:

```
>>> content.created
'NOW'
>>> content.modified
'NOW'
```

The created annotator can also be registered for `(object, event)`:

```
>>> from zope.component import subscribers
>>> provideHandler(CreatedAnnotator, (None, IObjectCreatedEvent,))
```

```
>>> content = Content()
>>> ignored = subscribers((content, ObjectCreatedEvent(content)), None)
```

Both created and modified get set this way, too:

```
>>> content.created
'NOW'
>>> content.modified
'NOW'
```

Modified annotator

The modified annotator only sets the modification time to current time.

```
>>> content = Content()
```

It is registered for the ObjectModifiedEvent:

```
>>> from zope.dublincore.timeannotators import ModifiedAnnotator
>>> from zope.lifecycleevent.interfaces import IObjectModifiedEvent
>>> provideHandler(ModifiedAnnotator, (IObjectModifiedEvent,))

>>> from zope.lifecycleevent import ObjectModifiedEvent
>>> notify(ObjectModifiedEvent(content))
```

Only modified gets set:

```
>>> print(content.created)
None
>>> content.modified
'NOW'
```

The modified annotator can also be registered for (object, event):

```
>>> provideHandler(ModifiedAnnotator, (None, IObjectModifiedEvent,))
>>> content = Content()
>>> ignored = subscribers((content, ObjectModifiedEvent(content)), None)
```

modified gets set, this way, too:

```
>>> print(content.created)
None
>>> content.modified
'NOW'
```


Dublin Core interfaces

interface `zope.dublincore.interfaces.IDublinCoreElementItem`

A qualified dublin core element

value

Value

The element value

qualification

Qualification

The element qualification

interface `zope.dublincore.interfaces.IGeneralDublinCore`

Dublin-core data access interface

The Dublin Core, <http://dublincore.org/>, is a meta data standard that specifies a set of standard data elements. It provides flexibility of interpretation of these elements by providing for element qualifiers that specialize the meaning of specific elements. For example, a date element might have a qualifier, like “creation” to indicate that the date is a creation date. In addition, any element may be repeated. For some elements, like subject, and contributor, this is obviously necessary, but for other elements, like title and description, allowing repetitions is not very useful and adds complexity.

This interface provides methods for retrieving data in full generality, to be compliant with the Dublin Core standard. Other interfaces will provide more convenient access methods tailored to specific element usage patterns.

getQualifiedCreators ()

Return a sequence of Creator IDublinCoreElementItem.

getQualifiedSources ()

Return a sequence of Source IDublinCoreElementItem.

getQualifiedRights ()

Return a sequence of Rights IDublinCoreElementItem.

getQualifiedSubjects ()
Return a sequence of Subject *IDublinCoreElementItem*.

getQualifiedRelations ()
Return a sequence of Relation *IDublinCoreElementItem*.

getQualifiedLanguages ()
Return a sequence of Language *IDublinCoreElementItem*.

getQualifiedTitles ()
Return a sequence of Title *IDublinCoreElementItem*.

getQualifiedContributors ()
Return a sequence of Contributor *IDublinCoreElementItem*.

getQualifiedIdentifiers ()
Return a sequence of Identifier *IDublinCoreElementItem*.

getQualifiedDescriptions ()
Return a sequence of Description *IDublinCoreElementItem*.

getQualifiedTypes ()
Return a sequence of Type *IDublinCoreElementItem*.

getQualifiedFormats ()
Return a sequence of Format *IDublinCoreElementItem*.

getQualifiedDates ()
Return a sequence of Date *IDublinCoreElementItem*.

getQualifiedCoverages ()
Return a sequence of Coverage *IDublinCoreElementItem*.

getQualifiedPublishers ()
Return a sequence of Publisher *IDublinCoreElementItem*.

interface `zope.dublincore.interfaces.IWritableGeneralDublinCore`
Provide write access to dublin core data

This interface augments *IStandardDublinCore* with methods for writing elements.

setQualifiedPublishers (*qualified_publishers*)
Set the qualified Publishers elements.

The argument must be a sequence of Publisher *IDublinCoreElementItem*.

setQualifiedTypes (*qualified_types*)
Set the qualified Types elements.

The argument must be a sequence of Type *IDublinCoreElementItem*.

setQualifiedLanguages (*qualified_languages*)
Set the qualified Languages elements.

The argument must be a sequence of Language *IDublinCoreElementItem*.

setQualifiedContributors (*qualified_contributors*)
Set the qualified Contributors elements.

The argument must be a sequence of Contributor *IDublinCoreElementItem*.

setQualifiedCreators (*qualified_creators*)
Set the qualified Creator elements.

The argument must be a sequence of Creator *IDublinCoreElementItem*.

setQualifiedRelations (*qualified_relations*)

Set the qualified Relations elements.

The argument must be a sequence of Relation *IDublinCoreElementItem*.

setQualifiedCoverages (*qualified_coverages*)

Set the qualified Coverages elements.

The argument must be a sequence of Coverage *IDublinCoreElementItem*.

setQualifiedSources (*qualified_sources*)

Set the qualified Sources elements.

The argument must be a sequence of Source *IDublinCoreElementItem*.

setQualifiedFormats (*qualified_formats*)

Set the qualified Formats elements.

The argument must be a sequence of Format *IDublinCoreElementItem*.

setQualifiedSubjects (*qualified_subjects*)

Set the qualified Subjects elements.

The argument must be a sequence of Subject *IDublinCoreElementItem*.

setQualifiedIdentifiers (*qualified_identifiers*)

Set the qualified Identifiers elements.

The argument must be a sequence of Identifier *IDublinCoreElementItem*.

setQualifiedDates (*qualified_dates*)

Set the qualified Dates elements.

The argument must be a sequence of Date *IDublinCoreElementItem*.

setQualifiedTitles (*qualified_titles*)

Set the qualified Title elements.

The argument must be a sequence of *IDublinCoreElementItem*.

setQualifiedDescriptions (*qualified_descriptions*)

Set the qualified Descriptions elements.

The argument must be a sequence of Description *IDublinCoreElementItem*.

setQualifiedRights (*qualified_rights*)

Set the qualified Rights elements.

The argument must be a sequence of Rights *IDublinCoreElementItem*.

interface `zope.dublincore.interfaces.IDCDescriptiveProperties`

Basic descriptive meta-data properties

description

Description

The first unqualified Dublin Core ‘Description’ element value.

title

Title

The first unqualified Dublin Core ‘Title’ element value.

interface `zope.dublincore.interfaces.IDCTimes`

Time properties

modified

Modification Date

The date and time that the object was last modified in a meaningful way.

created

Creation Date

The date and time that an object is created. This is normally set automatically.

interface `zope.dublincore.interfaces.IDCPublishing`

Publishing properties

expires

Expiration Date

The date and time that the object should become unpublished.

effective

Effective Date

The date and time that an object should be published.

interface `zope.dublincore.interfaces.IDCExtended`

Extended properties

This is a mixed bag of properties we want but that we probably haven't quite figured out yet.

publisher

Publisher

The first unqualified Dublin Core 'Publisher' element value.

contributors

Contributors

The unqualified Dublin Core 'Contributor' element values

subjects

Subjects

The unqualified Dublin Core 'Subject' element values

creators

Creators

The unqualified Dublin Core 'Creator' element values

interface `zope.dublincore.interfaces.ICMFDublinCore`

This interface duplicates the CMF dublin core interface.

Publisher ()

Dublin Core element - resource publisher

Return full formal name of the entity or person responsible for publishing the resource.

The first unqualified Dublin Core *Publisher* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

Identifier ()

Return the URL of the resource.

This value is computed. It is included in the output of `qualifiedIdentifiers` with the qualification 'url'.

Description ()

Return the resource description

Return a natural language description of this object.

The first unqualified Dublin Core *Description* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

Contributors ()

Return the resource contributors

Return any additional collaborators.

The unqualified Dublin Core *Contributor* element values are returned as a sequence of unicode strings.

Creator ()

Return the resource creators.

Return the full name(s) of the author(s) of the content object.

The unqualified Dublin Core *Creator* element values are returned as a sequence of unicode strings.

Title ()

Return the resource title.

The first unqualified Dublin Core *Title* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

Rights ()

Return the resource rights.

Return a string describing the intellectual property status, if any, of the resource. for the resource.

The first unqualified Dublin Core *Rights* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

EffectiveDate ()

Return the effective date

The value of the first Dublin Core *Date* element qualified by 'effective' is returned as a unicode string if a qualified element is defined, otherwise, an empty unicode string is returned. The string is formatted 'YYYY-MM-DD H24:MN:SS TZ'.

ModificationDate ()

Date resource last modified.

The value of the first Dublin Core *Date* element qualified by 'modification' is returned as a unicode string if a qualified element is defined, otherwise, an empty unicode string is returned. The string is formatted 'YYYY-MM-DD H24:MN:SS TZ'.

Language ()

Return the resource language.

Return the RFC language code (e.g., 'en-US', 'pt-BR') for the resource.

The first unqualified Dublin Core *Language* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

ExpirationDate ()

Date resource expires.

The value of the first Dublin Core *Date* element qualified by 'expiration' is returned as a unicode string if a qualified element is defined, otherwise, an empty unicode string is returned. The string is formatted 'YYYY-MM-DD H24:MN:SS TZ'.

Date ()

Return the default date

The first unqualified Dublin Core *Date* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned. The string is formatted 'YYYY-MM-DD H24:MN:SS TZ'.

Format ()

Return the resource format.

Return the resource's MIME type (e.g., 'text/html', 'image/png', etc.).

The first unqualified Dublin Core *Format* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

CreationDate ()

Return the creation date.

The value of the first Dublin Core *Date* element qualified by 'creation' is returned as a unicode string if a qualified element is defined, otherwise, an empty unicode string is returned. The string is formatted 'YYYY-MM-DD H24:MN:SS TZ'.

Type ()

Return the resource type

Return a human-readable type name for the resource.

The first unqualified Dublin Core *Type* element value is returned as a unicode string if an unqualified element is defined, otherwise, an empty unicode string is returned.

Subject ()

Return the resource subjects.

The unqualified Dublin Core *Subject* element values are returned as a sequence of unicode strings.

interface `zope.dublincore.interfaces.IZopeDublinCore`

Extends: `zope.dublincore.interfaces.IGeneralDublinCore`, `zope.dublincore.interfaces.ICMFDublinCore`, `zope.dublincore.interfaces.IDCDescriptiveProperties`, `zope.dublincore.interfaces.IDCTimes`, `zope.dublincore.interfaces.IDCPublishing`, `zope.dublincore.interfaces.IDCExtended`

Zope Dublin Core properties

interface `zope.dublincore.interfaces.IWriteZopeDublinCore`

Extends: `zope.dublincore.interfaces.IZopeDublinCore`, `zope.dublincore.interfaces.IWritableGeneralDublinCore`

Zope Dublin Core properties with generate update support

Hacking on `zope.dublincore`

Getting the Code

The main repository for `zope.dublincore` is in the Zope Foundation Github repository:

<https://github.com/zopefoundation/zope.dublincore>

You can get a read-only checkout from there:

```
$ git clone https://github.com/zopefoundation/zope.dublincore.git
```

or fork it and get a writeable checkout of your fork:

```
$ git clone git@github.com:jrandom/zope.dublincore.git
```

The project also mirrors the trunk from the Github repository as a Bazaar branch on Launchpad:

<https://code.launchpad.net/zope.dublincore>

You can branch the trunk from there using Bazaar:

```
$ bzr branch lp:zope.dublincore
```

Working in a `virtualenv`

Installing

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a `virtualenv`. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/hack-zope.dublincore
```

Next, get this package registered as a “development egg” in the environment:

```
$ /tmp/hack-zope.dublincore/bin/python setup.py develop
```

Running the tests

Run the tests using the build-in `setuptools` testrunner:

```
$ /tmp/hack-zope.dublincore/bin/python setup.py test
running test
.....
-----
Ran 80 tests in 0.000s

OK
```

If you have the `nose` package installed in the virtualenv, you can use its testrunner too:

```
$ /tmp/hack-zope.dublincore/bin/easy_install nose
...
$ /tmp/hack-zope.dublincore/bin/nosetests
.....
-----
Ran 80 tests in 0.000s

OK
```

If you have the `coverage` package installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/hack-zope.dublincore/bin/easy_install nose coverage
...
$ /tmp/hack-zope.dublincore/bin/nosetests --with coverage
running nosetests
.....
Name                                                    Stmt  Miss  Cover  Missing
-----
zope/dublincore.py                                     0      0  100%
zope/dublincore/annotatableadapter.py                 73     73  100%
zope/dublincore/browser.py                            0      0  100%
zope/dublincore/browser/metadataedit.py              21     21  100%
zope/dublincore/creatorannotator.py                  24     24  100%
zope/dublincore/dcsv.py                               92     92  100%
zope/dublincore/dcterms.py                           65     65  100%
zope/dublincore/interfaces.py                        72     72  100%
zope/dublincore/property.py                          66     66  100%
zope/dublincore/timeannotators.py                    27     27  100%
zope/dublincore/xmlmetadata.py                       173    173  100%
zope/dublincore/zopedublincore.py                   198    198  100%
-----
TOTAL                                                    811    811  100%
-----
Ran 86 tests in 0.000s

OK
```

Building the documentation

zope.dublincore uses the nifty Sphinx documentation system for building its docs. Using the same virtualenv you set up to run the tests, you can build the docs:

```
$ /tmp/hack-zope.dublincore/bin/easy_install Sphinx
...
$ bin/sphinx-build -b html -d docs/_build/doctrees docs docs/_build/html
...
build succeeded.
```

You can also test the code snippets in the documentation:

```
$ bin/sphinx-build -b doctest -d docs/_build/doctrees docs docs/_build/doctest
...
Doctest summary
=====
    73 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the \
  results in _build/doctest/output.txt.
```

Using zc.buildout

Setting up the buildout

zope.dublincore ships with its own buildout.cfg file and bootstrap.py for setting up a development buildout:

```
$ /path/to/python2.6 bootstrap.py
...
Generated script '../bin/buildout'
$ bin/buildout
Develop: '/home/jrandom/projects/Zope/BTK/dublincore/.'
...
Generated script '../bin/sphinx-quickstart'.
Generated script '../bin/sphinx-build'.
```

Running the tests

Run the tests:

```
$ bin/test --all
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Ran 400 tests with 0 failures and 0 errors in 0.366 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

Using tox

Running Tests on Multiple Python Versions

`tox` is a Python-based test automation tool designed to run tests against multiple Python versions. It creates a `virtualenv` for each configured version, installs the current package and configured dependencies into each `virtualenv`, and then runs the configured commands.

`zope.dublincore` configures the following `tox` environments via its `tox.ini` file:

- The `py26`, `py27`, `py33`, `py34`, and `pypy` environments builds a `virtualenv` with `pypy`, installs `zope.dublincore` and dependencies, and runs the tests via `python setup.py test -q`.
- The `coverage` environment builds a `virtualenv` with `python2.6`, installs `zope.dublincore`, installs `nose` and `coverage`, and runs `nosetests` with statement coverage.
- The `docs` environment builds a `virtualenv` with `python2.6`, installs `zope.dublincore`, installs `Sphinx` and dependencies, and then builds the docs and exercises the `doctest` snippets.

This example requires that you have a working `python2.6` on your path, as well as installing `tox`:

```
$ tox -e py26
GLOB sdist-make: .../zope.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
.....
-----
Ran 400 tests in 0.152s

OK
_____ summary _____
py26: commands succeeded
congratulations :)
```

Running `tox` with no arguments runs all the configured environments, including building the docs and testing their snippets:

```
$ tox
GLOB sdist-make: .../zope.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
...
Doctest summary
=====
 73 tests
  0 failures in tests
  0 failures in setup code
  0 failures in cleanup code
build succeeded.
_____ summary _____
py26: commands succeeded
py27: commands succeeded
py32: commands succeeded
pypy: commands succeeded
coverage: commands succeeded
docs: commands succeeded
congratulations :)
```

Contributing to zope.dublincore

Submitting a Bug Report

zope.dublincore tracks its bugs on Github:

<https://github.com/zopefoundation/zope.dublincore/issues>

Please submit bug reports and feature requests there.

Sharing Your Changes

Note: Please ensure that all tests are passing before you submit your code. If possible, your submission should include new tests for new features or bug fixes, although it is possible that you may have tested your new code by updating existing tests.

If have made a change you would like to share, the best route is to fork the Github repository, check out your fork, make your changes on a branch in your fork, and push it. You can then submit a pull request from your branch:

<https://github.com/zopefoundation/zope.dublincore/pulls>

If you branched the code from Launchpad using Bazaar, you have another option: you can “push” your branch to Launchpad:

```
$ bzr push lp:~jrandom/zope.dublincore/cool_feature
```

After pushing your branch, you can link it to a bug report on Launchpad, or request that the maintainers merge your branch using the Launchpad “merge request” feature.

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

Z

`zope.dublincore.interfaces`, 9

- C**
- contributors (zope.dublincore.interfaces.IDCExtended attribute), 12
 - Contributors() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 - created (zope.dublincore.interfaces.IDCTimes attribute), 12
 - CreationDate() (zope.dublincore.interfaces.ICMFDublinCore method), 14
 - Creator() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 - creators (zope.dublincore.interfaces.IDCExtended attribute), 12
- D**
- Date() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 - description (zope.dublincore.interfaces.IDCDescriptiveProperties attribute), 11
 - Description() (zope.dublincore.interfaces.ICMFDublinCore method), 12
- E**
- effective (zope.dublincore.interfaces.IDCPublishing attribute), 12
 - EffectiveDate() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 - ExpirationDate() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 - expires (zope.dublincore.interfaces.IDCPublishing attribute), 12
- F**
- Format() (zope.dublincore.interfaces.ICMFDublinCore method), 14
- G**
- getQualifiedContributors() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedCoverages() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedCreators() (zope.dublincore.interfaces.IGeneralDublinCore method), 9
 - getQualifiedDates() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedDescriptions() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedFormats() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedIdentifiers() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedLanguages() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedPublishers() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedRelations() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedRights() (zope.dublincore.interfaces.IGeneralDublinCore method), 9
 - getQualifiedSources() (zope.dublincore.interfaces.IGeneralDublinCore method), 9
 - getQualifiedSubjects() (zope.dublincore.interfaces.IGeneralDublinCore method), 9
 - getQualifiedTitles() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
 - getQualifiedTypes() (zope.dublincore.interfaces.IGeneralDublinCore method), 10
- I**
- ICMFDublinCore (interface in zope.dublincore.interfaces), 12
 - IDCDescriptiveProperties (interface in zope.dublincore.interfaces), 11
 - IDCExtended (interface in zope.dublincore.interfaces), 12
 - IDCPublishing (interface in zope.dublincore.interfaces), 12
 - IDCTimes (interface in zope.dublincore.interfaces), 11

Identifier() (zope.dublincore.interfaces.ICMFDublinCore method), 12
 IDublinCoreElementItem (interface zope.dublincore.interfaces), 9
 IGeneralDublinCore (interface zope.dublincore.interfaces), 9
 IWritableGeneralDublinCore (interface zope.dublincore.interfaces), 10
 IWriteZopeDublinCore (interface zope.dublincore.interfaces), 14
 IZopeDublinCore (interface zope.dublincore.interfaces), 14

L

Language() (zope.dublincore.interfaces.ICMFDublinCore method), 13

M

ModificationDate() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 modified (zope.dublincore.interfaces.IDCTimes attribute), 11

P

publisher (zope.dublincore.interfaces.IDCEExtended attribute), 12
 Publisher() (zope.dublincore.interfaces.ICMFDublinCore method), 12

Q

qualification (zope.dublincore.interfaces.IDublinCoreElementItem attribute), 9

R

Rights() (zope.dublincore.interfaces.ICMFDublinCore method), 13

S

setQualifiedContributors() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 setQualifiedCoverages() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedCreators() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 setQualifiedDates() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedDescriptions() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedFormats() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedIdentifiers() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedLanguages() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 setQualifiedPublishers() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 setQualifiedRelations() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 setQualifiedRights() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedSources() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedSubjects() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedTitles() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 11
 setQualifiedTypes() (zope.dublincore.interfaces.IWritableGeneralDublinCore method), 10
 Subject() (zope.dublincore.interfaces.ICMFDublinCore method), 14
 subjects (zope.dublincore.interfaces.IDCEExtended attribute), 12

T

title (zope.dublincore.interfaces.IDCDescriptiveProperties attribute), 11
 Title() (zope.dublincore.interfaces.ICMFDublinCore method), 13
 Type() (zope.dublincore.interfaces.ICMFDublinCore method), 14

value (zope.dublincore.interfaces.IDublinCoreElementItem attribute), 9

Z

zope.dublincore.interfaces (module), 9