everest Documentation

Release 1.0

everestlers

July 19, 2012

CONTENTS

`everest` is an extension of the popular `Pyramid` framework aimed at simplifying the development of REST applications.

CHAPTER

ONE

INSTALLATION

Installing `everest` is simple:

```
> pip install everest
```

CHAPTER

TWO

DOCUMENTATION

everest Tutorial Start here for a tutorial on building `everest` applications.

API Reference The full API Documentation

CHAPTER

THREE

DEVELOPMENT

everest is hosted on github. To contribute, please fork the project and submit a pull request.

3.1 everest Tutorial

This tutorial provides a quick overview of the terms and concepts used in everest and how to put them to work in an application.

Building everest applications

Using everest applications

Glossary of Terms

3.1.1 Building everest applications

In this section, you will find a step-by-step guide on how to build a RESTful application with everest.

1. The application

Suppose you want to write a program that helps a garden designer with composing lists of beautiful perennials and shrubs that she intends to plant in her customer's gardens. Let's call this fancy application "Plant Scribe". In its simplest possible form, this application will have to handle customers, projects (per customer), sites (per project), and plant species (per site).

2. Designing the entity model

everest applications keep their value state in *entity* objects.

Entities and Resources

The entity model implements the *domain logic* of the application by enforcing all value state constraints at all times.
Entities are manipulated through *resource* objects. A resource object provides access either to a single entity object (*member resource*) or to a collection of entities of the same kind (*collection resource*). Resources can call other resources to modify other parts of the entity model, thus implementing the *business logic* of the application. Each collection resource uses an *aggregate* to provide access to its underlying entities. They support slicing, filtering, and ordering operations.

The first step on our way to the Plant Scribe application is therefore to decide which data we want to store in our entity model. We start with the customer:

```
from everest.entities.base import Entity
from everest.entities.utils import slug_from_string

class Customer(Entity):
    def __init__(self, first_name, last_name, **kw):
        Entity.__init__(self, **kw)
        self.first_name = first_name
        self.last_name = last_name

    @property
    def slug(self):
        return slug_from_string("%s-%s" % (self.last_name, self.first_name))
```

In our example, the `Customer` class inherits from the `Entity` class provided by `everest`. This is convenient, but not necessary; any class can participate in the entity model as long as it implements the `everest.entities.interfaces.IEntity` interface. Note, however, that this interface requires the presence of a `slug` attribute, which in the case of the customer entity is composed of the concatenation of the customer's last and first name.

Slugs

A *slug* is a character string that uniquely identifies an entity within its aggregate. `everest` uses the slug as part of the URL for the member resource wrapping an entity, so, ideally, it should ideally be a short, mnemonic expression.

For each customer, we need to be able to handle an arbitrary number of projects:

```
from everest.entities.base import Entity
from everest.entities.utils import slug_from_string

class Project(Entity):
    def __init__(self, name, customer, **kw):
        Entity.__init__(self, **kw)
        self.name = name
        self.customer = customer

    @property
    def slug(self):
        return slug_from_string(self.name)
```

Note that the `name` attribute, which serves as the project entity slug, does not need to be unique among *all* projects, but just among all projects for a given customer.

Another noteworthy observation is that although the project references the customer, we do not (yet) have a way to access the projects associated with a given customer as an attribute of its customer entity. Avoiding such circular references allows us to keep our entity model simple, but we may be missing the convenience they offer. We will return to this issue a little later.

Each project is referenced by one or more planting sites:

```
from everest.entities.base import Entity
from everest.entities.utils import slug_from_string

class Site(Entity):
    def __init__(self, name, project, **kw):
        Entity.__init__(self, **kw)
        self.name = name
```

```
        self.project = project

    @property
    def slug(self):
        return slug_from_string(self.name)
```

The plant species to choose from for each site are modeled as follows:

```
from everest.entities.base import Entity
from everest.entities.utils import slug_from_string

class Species(Entity):
    def __init__(self, species_name, genus_name,
                 cultivar=None, author=None, **kw):
        Entity.__init__(self, **kw)
        self.species_name = species_name
        self.genus_name = genus_name
        self.cultivar = cultivar
        self.author = author

    @property
    def slug(self):
        return slug_from_string(
                    "%s-%s-%s-%s"
                    % (self.genus_name, self.species_name,
                       '' if self.cultivar is None else self.cultivar,
                       '' if self.author is None else self.author))
```

Finally, the information about which plant species to use at which site and in which quantity is modeled as an "incidence" entity:

```
from everest.entities.base import Entity

class Incidence(Entity):
    def __init__(self, species, site, quantity, **kw):
        Entity.__init__(self, **kw)
        self.species = species
        self.site = site
        self.quantity = quantity

    @property
    def slug(self):
        return None if self.species is None else self.species.slug
```

3. Designing and building the resource layer

With the entity model in place, we can now proceed to designing the resource layer. The first step here is to define the marker interfaces that `everest` will use to access the various parts of the resource system. This is very straightforward to do:

```
"""
This file is part of the everest project.
See LICENSE.txt for licensing, CONTRIBUTORS.txt for contributor information.

Created on Jan 9, 2012.
"""
from zope.interface import Interface # pylint: disable=F0401

__docformat__ = 'reStructuredText en'
```

```
__all__ = ['ICustomer',
           'IIncidence',
           'IProject',
           'ISite',
           'ISpecies',
           ]


# no __init__ pylint: disable=W0232
class ICustomer(Interface):
    pass


class IProject(Interface):
    pass


class ISpecies(Interface):
    pass


class ISite(Interface):
    pass


class IIncidence(Interface):
    pass
# pylint: enable=W0232
```

Next, we move on to declaring the resource attributes using `everest`'s resource attribute descriptors. Each resource attribute descriptor maps a single attribute from the resource's entity and makes it available for access from the outside.

Resource Attribute Kinds

There are three kinds of resource attributes in `everest`: Terminal attributes, member attributes, and collection attributes. A *terminal* resource attribute references an object of an atomic type or some other type that is not a resource itself. A *member* resource attribute references another member resource and a *collection* resource attribute references another collection resource. Resource attributes are declared using the `terminal_attribute()`, `member_attribute()`, and `collection_attribute()` descriptor generating functions from the `resources.descriptors` module.

In our example application, the resources mostly declare the public attributes of the underlying entities as attributes:

```
from everest.resources.base import Member
from everest.resources.descriptors import collection_attribute
from everest.resources.descriptors import terminal_attribute
from plantscribe.interfaces import IProject

class CustomerMember(Member):
    relation = 'http://plantscribe.org/relations/customer'
    first_name = terminal_attribute(str, 'first_name')
    last_name = terminal_attribute(str, 'last_name')
    projects = collection_attribute(IProject, backref='customer')
```

```
from everest.resources.base import Member
from everest.resources.descriptors import collection_attribute
from everest.resources.descriptors import member_attribute
```

```
from everest.resources.descriptors import terminal_attribute
from plantscribe.interfaces import ICustomer
from plantscribe.interfaces import ISite

class ProjectMember(Member):
    relation = 'http://plantscribe.org/relations/project'
    name = terminal_attribute(str, 'name')
    customer = member_attribute(ICustomer, 'customer')
    sites = collection_attribute(ISite, backref='project', is_nested=True)
```

```
from everest.resources.base import Member
from everest.resources.descriptors import collection_attribute
from everest.resources.descriptors import member_attribute
from everest.resources.descriptors import terminal_attribute
from plantscribe.interfaces import IIncidence
from plantscribe.interfaces import IProject

class SiteMember(Member):
    relation = 'http://plantscribe.org/relations/site'
    name = terminal_attribute(str, 'name')
    incidences = collection_attribute(IIncidence, backref='site',
                                      is_nested=True)
    project = member_attribute(IProject, 'project')
```

```
from everest.resources.base import Member
from everest.resources.descriptors import terminal_attribute

class SpeciesMember(Member):
    relation = 'http://plantscribe.org/relations/species'
    species_name = terminal_attribute(str, 'species_name')
    genus_name = terminal_attribute(str, 'genus_name')
    cultivar = terminal_attribute(str, 'cultivar')
    author = terminal_attribute(str, 'author')
```

```
from everest.resources.base import Member
from everest.resources.descriptors import member_attribute
from everest.resources.descriptors import terminal_attribute
from plantscribe.interfaces import ISite
from plantscribe.interfaces import ISpecies

class IncidenceMember(Member):
    relation = 'http://plantscribe.org/relations/incidence'
    species = member_attribute(ISpecies, 'species')
    site = member_attribute(ISite, 'site')
    quantity = terminal_attribute(float, 'quantity')
```

In the simple case where the resource attribute descriptor declares a public attribute of the underlying entity, it expects a type or an interface of the target object and the name of the corresponding entity attribute as arguments.

URL resolution

`everest` favors and facilitates object traversal for URL resolution. In particular, all resource attributes that target a member or collection resource can be used directly for URL traversal unless they are specifically set as non-nested resource in the corresponding resource attribute declaration.

For `member_attribute()` and `collection_attribute()` descriptors there is also an optional argument `is_nested` which determines if the URL for the target resource is going to be formed relative to the root (i.e., as an

absolute path) or relative to the parent resource declaring the attribute.

We also have the possibility to declare resource attributes that do not reference the target resource directly through an entity attribute, but indirectly through a "backreferencing" attribute. In the example code, this is demonstrated in the `projects` attribute of the `CustomerMember` resource which allows us to access a customer's projects at the resource level even though the underlying entity does not reference its projects directly.

4. Configuring the application

With the resource layer in place, we can now move on to configuring our application. `everest` applications are based on the `pyramid` framework and everything you learned about configuring `pyramid` applications can be applied here. Rather than duplicating the excellent documentation available on the Pyramid web site, we will focus on a minimal example on how to configure the extra resource functionality that `everest` supplies.

The minimal `.ini` file for the `plantscribe` application is quite simple:

```
[DEFAULT]

[app:main]
paste.app_factory = plantscribe.run:app_factory

[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 6543
```

The only purpose of the `.ini` file is to specify a `Paster` application factory which is responsible for creating and setting up the application registry and for instantiating a WSGI application.

The `.zcml` configuration file - which is loaded through the application factory - is more interesting:

```
<configure xmlns="http://pylonshq.com/pyramid">

    <!-- Include special directives. -->

    <include package="everest.includes" />

    <!-- Repositories. -->

    <!-- Resource declarations. -->

    <include file="resources.zcml" />

</configure>
```

Note the `include` directive at the top of the file; this not only pulls in the `everest`-specific ZCML directives, but also the Pyramid directives as well.

The most important of the `everest`-specific directives is the `resource` directive. This sets up the connections between the various parts of the resource subsystem, using our marker interfaces as the glue. At the minimum, you need to specify

- A marker interface for your resource;
- An entity class for the resource;
- A member class class for the resource; and
- A name for the root collection.

The aggregate and collection objects needed by the resource subsystem (cf. xxx) are created automatically; you may, however, supply a custom collection class that inherits from `everest.resources.base.Collection`. If you

do not plan on exposing the collection for this resource to the outside, you can set the `expose` flag to `false`, in which case you do not need to provide a root collection name. Non-exposed resources will still be available as a root collection internally, but access through the service as well as the generation of absolute URLs will not work.

5. Running the application

To see our little application in action, we can use the `pshell` interactive shell that comes with `Pyramid`. First, install the `plantscribe` package by issuing

```
$ pip install -e .
```

inside the `docs/demoapp/v0` folder of the `everest` source tree. This presumes you have followed the instructions of installing `everest` and use a `virtualenv` with the `pip` installer (cf. xxx).

Now, still from the same directory, you start the `Pyramid` `pshell` like this:

```
$ pshell plantscribe.ini
Python 2.7.2 (v2.7.2:8527427914a2, Jun 11 2011, 15:22:34)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help" for more information.

Environment:
  app          The WSGI application.
  registry     Active Pyramid registry.
  request      Active request object.
  root         Root of the default resource tree.
  root_factory Default root factory used to create `root`.

>>>
```

The `root` object that is available in the `pshell` environment is the service object that provides access to all public root collections by name:

```
>>> c = root['customers']
>>> c
<CustomerMemberCollection name:customers parent:Service(started)>
```

We can now start adding members to the collection and retrieve them back from the collection:

```
>>> from plantscribe.entities.customer import Customer
>>> ent = Customer('Peter', 'Fox')
>>> m = c.create_member(ent)
>>> m.__name__
'fox-peter'
>>> c.get('fox-peter').__name__
'fox-peter'
```

6. Adding persistency

With the application running, we now turn our attention to persistency. `everest` uses a *repository* to load and save resources from and to a storage backend. To use a filesystem-based repository as the default for our application, we could use the following ZCML declaration:

```
<filesystem_repository
   directory="data"
   content_type="everest.mime.CsvMime"
   make_default="true" />
```

This tells `everest` to use the `data` directory (relative to the `plantscribe` package) to persist representations of the root collections of all resources as `.csv` (Comma Separated Value) files. When the application is initialized,

the root collections are loaded from these representation files and during each `commit` operation at the end of a transaction, all modified root collections are written back to their corresponding representation files.

The filesystem-based repository does not perform well with complex or high volume data structures or in cases where several processes need to access the same persistency backend. In these situations, we need to switch to a an ORM-based repository. `everest` uses xxx `SQLAlchemy` as ORM. What follows is a highly simplified account of what is needed to instruct `SQLAlchemy` to persist the entities of an `everest` application; for an explanation of the terms and concepts used in this section, please refer to the excellent documentation on the *SQLAlchemy http://sqlalchemy.org* web site.

In a first step, we need to initialize the ORM. The following ZCML declaration makes the ORM the default resource repository:

```
<orm_repository
    metadata_factory="everest.tests.testapp_db.db.create_metadata"
    make_default="true"/>
```

The metadata factory setting references a callable that takes an `SQLAlchemy` engine as a parameter and returns a fully initialized metadata instance. For our simple application, this function looks like this:

```
"""
This file is part of the everest project.
See LICENSE.txt for licensing, CONTRIBUTORS.txt for contributor information.

Created on Mar 27, 2012.
"""
from everest.orm import as_slug_expression
from everest.orm import mapper
from plantscribe.entities.customer import Customer
from plantscribe.entities.incidence import Incidence
from plantscribe.entities.project import Project
from plantscribe.entities.site import Site
from plantscribe.entities.species import Species
from sqlalchemy import Column
from sqlalchemy import Float
from sqlalchemy import ForeignKey
from sqlalchemy import Integer
from sqlalchemy import MetaData
from sqlalchemy import String
from sqlalchemy import Table
from sqlalchemy.orm import relationship
from sqlalchemy.sql import literal
from sqlalchemy.sql import select

__docformat__ = 'reStructuredText en'
__all__ = []


def customer_slug(cls):
    return as_slug_expression(cls.last_name + literal('-') + cls.first_name)


def project_slug(cls):
    return as_slug_expression(cls.name)


def species_slug(cls):
    return as_slug_expression(cls.genus_name + literal('-') +
                              cls.species_name + literal('-') +
```

```
                                cls.cultivar + literal('-') +
                                cls.author)


def site_slug(cls):
    return as_slug_expression(cls.name)


def incidence_slug(cls):
    return \
        select([Species.slug]).where(cls.species_id == Species.id).as_scalar()


def create_metadata(engine):
    # Create metadata.
    metadata = MetaData()
    # Define a database schema..
    customer_tbl = \
        Table('customer', metadata,
              Column('customer_id', Integer, primary_key=True),
              Column('first_name', String, nullable=False),
              Column('last_name', String, nullable=False),
              )
    project_tbl = \
        Table('project', metadata,
              Column('project_id', Integer, primary_key=True),
              Column('name', String, nullable=False),
              Column('customer_id', Integer,
                     ForeignKey(customer_tbl.c.customer_id),
                     nullable=False),
              )
    site_tbl = \
        Table('site', metadata,
              Column('site_id', Integer, primary_key=True),
              Column('name', String, nullable=False),
              Column('project_id', Integer,
                     ForeignKey(project_tbl.c.project_id),
                     nullable=False),
              )
    species_tbl = \
        Table('species', metadata,
              Column('species_id', Integer, primary_key=True),
              Column('species_name', String, nullable=False),
              Column('genus_name', String, nullable=False),
              Column('cultivar', String, nullable=False, default=''),
              Column('author', String, nullable=False),
              )
    incidence_tbl = \
        Table('incidence', metadata,
              Column('site_id', Integer,
                     ForeignKey(site_tbl.c.site_id),
                     primary_key=True, index=True, nullable=False),
              Column('species_id', Integer,
                     ForeignKey(species_tbl.c.species_id),
                     primary_key=True, index=True, nullable=False),
              Column('quantity', Float, nullable=False),
              )
    # Map tables to entity classes.
```

```
98      mapper(Customer, customer_tbl,
99             id_attribute='customer_id', slug_expression=customer_slug)
100     mapper(Project, project_tbl,
101            id_attribute='project_id', slug_expression=project_slug,
102            properties=dict(customer=relationship(Customer, uselist=False)))
103     mapper(Site, site_tbl,
104            id_attribute='site_id', slug_expression=site_slug,
105            properties=dict(project=relationship(Project, uselist=False)))
106     mapper(Species, species_tbl,
107            id_attribute='species_id', slug_expression=species_slug)
108     mapper(Incidence, incidence_tbl,
109            slug_expression=incidence_slug,
110            properties=dict(species=relationship(Species, uselist=False),
111                            site=relationship(Site, uselist=False)))
112     # Configure and initialize metadata.
113     metadata.bind = engine
114     metadata.create_all()
115     return metadata
```

The function first creates a database schema and then maps our entity classes to this schema. Note that a special mapper is used which provides a convenient way to map the special *id* and *slug* attributes required by `everest` to the ORM layer.

To use an engine other than the default in-memory SQLite database engine, you need to supply a `db_string` setting in the paster application `.ini` file. For example:

Different resorces may use different repositories, but any given resource can only be assigned to one repository.

3.1.2 Using `everest` applications

Querying with GET

One of the main features of the collection resources in `everest` are their advanced querying capabilities. Query strings have to conform to

An incoming query through a GET request is processed by `everest` in two steps: First, the query string submitted by the client is parsed into a query specification; and second, this query specification is translated to an object that can be applied to the collection resource acting as the query context.

Collection Query Language (CQL)

`everest` supports a custom Collection Query Language (CQL) for querying collection resources.
CQL query expressions are composed of one or more query criteria separated by the tilde ("~") character. Each criterion consists of three parts separated by a colon (":") character :

1. resource attribute name The name of the resource attribute to query. You can specify dotted names to query nested resources.
2. operator The operator to apply.
3. value The value to query for. It is possible to supply multiple values in a comma separated list, which will be interpreted as a Boolean "OR" operation on all given values.

Supported query criterion value types are:
String Arbitrary string enclosed in double quotes.
Number Integer or floating point, scientific notation allowed.
Boolean Case insensitive string **true** or **false**.
Date/Time ISO 8601 encoded string enclosed in double quotes.
Resource URL referencing a resource.

As an example, querying a collection resource “”

If a query contains multiple criteria with different resource attribute names, the criteria are interpreted as a Boolean “AND” operation.

The following table shows the available operators and data types in CQL:

Operator	Data Type				
Name	String	Number	Boolean	Date/Time	Resource
starts-with	x				
not-starts-with	x				
ends-with	x				
not-ends-with	x				
contains	x				
not-contains	x				
contained	x				
not-contained	x				
equal-to					
not-equal-to					
less-than					
less-than-or-equal-to					
greater-than					
greater-than-or-equal-to					
in-range					

All attributes that are used to compose a query expression need to be mapped column properties in the ORM. Aliases are supported, CompositeProperties are not. All queried entities must have an “id” attribute.

It is by design that the power of CQL to express complex queries is far behind that of SQL.

3.1.3 Glossary of Terms

entity An object in the domain model holding value state.

domain logic The set of rules and constraints governing the value state of the application.

resource xxx

member resource xxx

collection resource xxx

business logic The set of rules and constraints governing the behavior of the application.

slug A character string that uniquely identifies a *member resource* within its collection.

repository xxx

aggregate xxx

3.2 API Reference

3.2.1 Entities

everest.entities.aggregates

Continued on next page

Table 3.1 – continued from previous page

everest.entities.attributes
everest.entities.base
everest.entities.interfaces
everest.entities.repository
everest.entities.system
everest.entities.utils

everest.entities.aggregates

Aggregate implementations.

class everest.entities.aggregates.**MemoryAggregate**(*entity_class*, *session_factory*)
Bases: everest.entities.base.Aggregate

In-memory implementation for aggregates.

Note When "blank" entities without an ID and a slug are added to a memory aggregate, they can not be retrieved using the `get_by_id()` or `get_by_slug()` methods since there is no mechanism to autogenerate IDs or slugs.

class everest.entities.aggregates.**OrmAggregate**(*entity_class*, *session_factory*, *search_mode=False*)
Bases: everest.entities.base.Aggregate

ORM implementation for aggregates.

everest.entities.attributes

Entity attributes.

everest.entities.base

Entity and aggregate base classes.

class everest.entities.base.**Aggregate**(*entity_class*, *session_factory*)
Bases: object

Abstract base class for all aggregates.

An aggregate is an accessor for a set of entities of the same type which are held in some repository.

The wrapped entity set may be a "root" set of all entities in the repository or a "relation" set defined by a relationship to entities of some other type.

Supports filtering, sorting, slicing, counting, iteration as well as retrieving, adding and removing entities.

__init__(*entity_class*, *session_factory*)
Constructor:

Parameters

- **entity_class** (a class implementing everest.entities.interfaces.IEntity) – the entity class (type) of the entities in this aggregate.
- **session** – Session object.

__weakref__
list of weak references to the object (if defined)

add(*entity*)
Adds an entity to the aggregate.

If the entity has an ID, it must be unique within the aggregate.

Parameters **entity** (object implementing `everest.entities.interfaces.IEntity`) – entity (domain object) to add

Raises ValueError if an entity with the same ID exists

clone()
Returns a clone of this aggregate.

count()
Returns the total number of entities in the underlying aggregate. If specified, filter specs are applied. A specified slice key is ignored.

Returns number of aggregate members (`int`)

classmethod create(*entity_class*, *session_factory*)
Factory class method to create a new aggregate.

entity_class = None
Entity class (type) of the entities in this aggregate.

get_by_id(*id_key*)
Returns an entity by ID from the underlying aggregate or *None* if the entity is not found.

Note if a filter is set which matches the requested entity, it will not be found.

Parameters **id_key** (*int* or *str*) – ID value to look up

Raises `everest.exceptions.DuplicateException` if more than one entity is found for the given ID value.

Returns specified entity or *None*

Returns a single entity from the underlying aggregate by ID.

get_by_slug(*slug*)
Returns an entity by slug or *None* if the entity is not found.

Parameters **slug** (*str*) – slug value to look up

Raises `everest.exceptions.DuplicateException` if more than one entity is found for the given ID value.

Returns entity or *None*

iterator()
Returns an iterator for the entities contained in the underlying aggregate.

If specified, filter, order, and slice settings are applied.

Returns an iterator for the aggregate entities

remove(*entity*)
Removes an entity from the aggregate.

Parameters **entity** (object implementing `everest.entities.interfaces.IEntity`) – entity (domain object) to remove

Raises ValueError entity was not found

set_relationship(*relationship*)
Sets a relationship for this aggregate.

Parameters **relationship** – instance of `thelma.relationsip.Relationship`.

class `everest.entities.base.`**`Entity`**(*id=None*)
Bases: `object`

Abstract base class for all model entities.

All entities have an ID which is used as the default value for equality comparison. The object may be initialized without an ID.

`__weakref__`
list of weak references to the object (if defined)

`slug`
Returns a human-readable and URL-compatible string that is unique within all siblings of this entity.

everest.entities.interfaces

Interfaces for entity and aggregate classes.

everest.entities.repository

Entity repository.

class `everest.entities.repository.`**`EntityRepository`**(*entity_store*, *aggregate_class*)
Bases: `everest.repository.Repository`

The entity repository manages entity accessors (aggregates).

In addition to creating and caching aggregates, the entity repository also provides facilities to interact with the aggregate implementation registry. This makes it possible to switch the implementation used for freshly created aggregates at runtime.

`aggregate_class` = **None**
The class to use when creating new aggregates.

everest.entities.system

System entities.

everest.entities.utils

Entity related utilities.

`everest.entities.utils.`**`get_entity_class`**(*rc*)
Returns the entity class registered for the given registered resource.

Parameters **member** – registered resource

Returns entity class (class implementing *everest.entities.interfaces.IEntity*)

`everest.entities.utils.`**`get_root_aggregate`**(*rc*)
Returns an aggregate from the root entity repository for the given registered resource.

`everest.entities.utils.`**`get_stage_aggregate`**(*rc*)
Returns an aggregate from the stage entity repository for the given registered resource.

`everest.entities.utils.`**`identifier_from_slug`**(*slug*)
Converts the given slug into an identifier string.

Parameters **slug** (*str*) – slug string

everest.entities.utils.**slug_from_identifier**(*id_string*)
Converts the given identifier string into a slug.

Parameters **id_string** (*str*) – identifier string

everest.entities.utils.**slug_from_integer**(*integer*)
Slugs are mnemonic string identifiers for resources for use in URLs.

This function converts an integer into a string slug.

everest.entities.utils.**slug_from_string**(*string*)
Slugs are mnemonic string identifiers for resources for use in URLs.

This function replaces characters that are not allowed to occur in a URL with allowed characters.

3.2.2 Querying

everest.querying.base

Querying operators, expressions, visitors, builders, directors.

class everest.querying.base.**BinaryOperator**
Bases: everest.querying.base.Operator

Binary querying operator.

class everest.querying.base.**CqlExpression**
Bases: object

Single CQL expression.

CQL expressions can be converted to a string and support the conjunction (AND) operation.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**CqlExpressionList**(*expressions*)
Bases: object

List of CQL expressions.

Like a single CQL expression, CQL expression lists can be converted to a string and joined with the conjunction (AND) operation.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**Operator**
Bases: object

Base class for querying operators.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**Specification**
Bases: object

Abstract base classs for all specifications.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**SpecificationBuilder**(*spec_factory*)
Bases: object

Base class for specification builders.

__weakref__
list of weak references to the object (if defined)

specification
Returns the built specification.

class everest.querying.base.**SpecificationDirector**(*parser*, *builder*)
Bases: object

Abstract base class for specification directors.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**SpecificationVisitor**
Bases: everest.querying.base.SpecificationVisitorBase

Base class for all specification visitors.

class everest.querying.base.**SpecificationVisitorBase**
Bases: object

Base class for specification visitors.

__weakref__
list of weak references to the object (if defined)

class everest.querying.base.**UnaryOperator**
Bases: everest.querying.base.Operator

Unary querying operator.

everest.querying.filtering

Filter specification builder, visitor, director classes.

class everest.querying.filtering.**CqlFilterSpecificationVisitor**
Bases: everest.querying.filtering.FilterSpecificationVisitor

Filter specification visitor building a CQL expression.

class everest.querying.filtering.**EvalFilterSpecificationVisitor**
Bases: everest.querying.filtering.FilterSpecificationVisitor

Filter specification visitor building an evaluator for in-memory filtering.

class everest.querying.filtering.**FilterSpecificationBuilder**(*spec_factory*)
Bases: everest.querying.base.SpecificationBuilder

Filter specification builder.

The filter specification builder is responsible for building concrete specs with build methods dispatched by the director and for forming disjunction specs when a) multiple values are given in a single criterion; or b) the same combination of attribute name and operator is encountered multiple times.

class everest.querying.filtering.**FilterSpecificationDirector**(*parser*, *builder*)
Bases: everest.querying.base.SpecificationDirector

Director for filter specifications.

class everest.querying.filtering.**FilterSpecificationVisitor**
Bases: everest.querying.base.SpecificationVisitor

Base class for filter specification visitors.

class everest.querying.filtering.**SqlFilterSpecificationVisitor**(*entity_class*, *custom_clause_factories=None*)
Bases: everest.querying.filtering.FilterSpecificationVisitor

Filter specification visitor building a SQL expression.

__init__(*entity_class*, *custom_clause_factories=None*)
Constructs a SqlFilterSpecificationVisitor

Parameters

- **entity_class** – an entity class that is mapped with SQLAlchemy
- **custom_clause_factories** – a map containing custom clause factory functions for selected (attribute name, operator) combinations.

everest.querying.filterparser

Filter CQL criteria expression parser.

everest.querying.filterparser.**parse_filter**(*criteria_string*)
Parses the given filter criteria string.

everest.querying.interfaces

Interfaces for specifications and related classes.

everest.querying.operators

Custom querying operators.

class everest.querying.operators.**CQL_FILTER_OPERATORS**
Bases: object

Static container for all CQL filtering operators.

__weakref__
list of weak references to the object (if defined)

class everest.querying.operators.**CQL_ORDER_OPERATORS**
Bases: object

Static container for all CQL ordering operators.

__weakref__
list of weak references to the object (if defined)

everest.querying.orderparser

Order CQL expression parser.

everest.querying.orderparser.**parse_order**(*criteria_string*)
Parses the given order criteria string.

everest.querying.specifications

Specifications.

This file is part of the everest project. See LICENSE.txt for licensing, CONTRIBUTORS.txt for contributor information.

The central idea of a Specification is to separate the statement of how to match a candidate from the candidate object that it is matched against.

class everest.querying.specifications.**CompositeFilterSpecification**(*left_spec*, *right_spec*)
Bases: everest.querying.specifications.FilterSpecification

Abstract base class for specifications that are composed of two other specifications.

__init__(*left_spec*, *right_spec*)
Constructs a CompositeFilterSpecification

Parameters

- **left_spec** (FilterSpecification) – the left part of the composite specification
- **right_spec** (FilterSpecification) – the right part of the composite specification

class everest.querying.specifications.**ConjunctionFilterSpecification**(*left_spec*, *right_spec*)
Bases: everest.querying.specifications.CompositeFilterSpecification

Concrete conjuction specification.

class everest.querying.specifications.**CriterionFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.LeafFilterSpecification

Abstract base class for specifications representing filter criteria.

__init__(*attr_name*, *attr_value*)
Constructs a filter specification for a query criterion.

Parameters

- **operator** (everest.querying.operators.Operator) – operator
- **attr_name** (*str*) – the candidate's attribute name
- **attr_value** – the value that satisfies the specification

class everest.querying.specifications.**DisjuctionFilterSpecification**(*left_spec*, *right_spec*)
Bases: everest.querying.specifications.CompositeFilterSpecification

Concrete disjuction specification.

class everest.querying.specifications.**FilterSpecification**
Bases: everest.querying.base.Specification

Abstract base class for all filter specifications.

and_(*other*)
Generative method to create a ConjunctionFilterSpecification.

Parameters **other** (FilterSpecification) – the other specification

Returns a new conjuction specification

Return type ConjunctionFilterSpecification

is_satisfied_by(*candidate*)
Tells if the given candidate object matches this specification.

Parameters **candidate** (*object*) – the candidate object

Returns True if the specification is met by the candidate.

Return type bool

not_()
Generative method to create a NegationFilterSpecification

Returns a new negation specification

Return type NegationFilterSpecification

or_(*other*)
Generative method to create a DisjuctionFilterSpecification

Parameters **other** (FilterSpecification) – the other specification

Returns a new disjuction specification

Return type DisjuctionFilterSpecification

class everest.querying.specifications.**FilterSpecificationFactory**
Bases: object

Filter specification factory.

__weakref__
list of weak references to the object (if defined)

class everest.querying.specifications.**LeafFilterSpecification**
Bases: everest.querying.specifications.FilterSpecification

Abstract base class for specifications that represent leaves in a specification tree.

class everest.querying.specifications.**NaturalOrderSpecification**(*attr_name*)
Bases: everest.querying.specifications.ObjectOrderSpecification

See http://www.codinghorror.com/blog/2007/12/sorting-for-humans-natural-sort-order.html

class everest.querying.specifications.**NegationFilterSpecification**(*wrapped_spec*)
Bases: everest.querying.specifications.FilterSpecification

Concrete negation specification.

__eq__(*other*)
Equality operator

__init__(*wrapped_spec*)
Constructs a NegationFilterSpecification

Parameters **wrapped** (FilterSpecification) – the wrapped specification

__ne__(*other*)
Inequality operator

class everest.querying.specifications.**OrderSpecificationFactory**
Bases: object

Order specification factory.

__weakref__
list of weak references to the object (if defined)

class everest.querying.specifications.**ValueContainedFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value contained in a list of values specification

class everest.querying.specifications.**ValueContainsFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value contains specification

class everest.querying.specifications.**ValueEndsWithFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value ends with specification

class everest.querying.specifications.**ValueEqualToFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value equal to specification

class everest.querying.specifications.**ValueGreaterThanFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value greater than specification

class everest.querying.specifications.**ValueGreaterThanOrEqualToFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value greater than or equal to specification

class everest.querying.specifications.**ValueInRangeFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete specification for a range of values

class everest.querying.specifications.**ValueLessThanFilterSpecification**(*attr_name*, *attr_value*)
Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value less than specification

class everest.querying.specifications.**ValueLessThanOrEqualToFilterSpecification**(*attr_name*, *attr_value*)

Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value less than or equal to specification

class everest.querying.specifications.**ValueStartsWithFilterSpecification**(*attr_name*, *attr_value*)

Bases: everest.querying.specifications.CriterionFilterSpecification

Concrete value starts with specification

everest.querying.utils

Querying utilities.

class everest.querying.utils.**OrmAttributeInspector**

Bases: object

Helper class inspecting class attributes mapped by the ORM.

__weakref__

list of weak references to the object (if defined)

static inspect(*orm_class*, *attribute_name*)

Parameters **attribute_name** – name of the mapped attribute to inspect.

Returns list of 2-tuples containing information about the inspected attribute (first element: mapped entity attribute kind; second attribute: mapped entity attribute)

everest.querying.utils.**get_filter_specification_factory**()

Returns the object registered as filter specification factory utility.

Returns object implementing everest.querying.interfaces.IFilterSpecificationFactory

everest.querying.utils.**get_order_specification_factory**()

Returns the object registered as order specification factory utility.

Returns object implementing everest.querying.interfaces.IOrderSpecificationFactory

3.2.3 Representers

everest.representers.atom
everest.representers.attributes
everest.representers.base
everest.representers.config
everest.representers.converters
everest.representers.csv
everest.representers.dataelements
everest.representers.interfaces
everest.representers.mapping
everest.representers.traversal
everest.representers.urlloader
everest.representers.utils
everest.representers.xml

everest.representers.atom

ATOM representers.

everest.representers.atom.**AtomRepresenterConfiguration**
alias of XmlRepresenterConfiguration

everest.representers.attributes

Mapped resource attributes.

class everest.representers.attributes.**MappedAttribute**(*attr*, *options=None*)
Bases: object

Represents an attribute mapped from a class into a representation.

Wraps a (read-only) resource attribute and mapping options which can be configured dynamically.

__init__(*attr*, *options=None*)

Parameters **attr** – Resource attribute.

__weakref__
list of weak references to the object (if defined)

everest.representers.base

Representer base classes.

class everest.representers.base.**Representer**
Bases: object

Base class for all representers.

A representer knows how to convert an object into a representation of a particular MIME type (content type) and vice versa.

__weakref__
list of weak references to the object (if defined)

class everest.representers.base.**RepresenterRegistry**
Bases: object

Registry for representer classes and representer factories.

__weakref__
list of weak references to the object (if defined)

create(*resource*, *content_type*)
Creates a representer for the given combination of resource and content type. This will also find representer factories that were registered for a base class of the given resource.

register(*resource_class*, *content_type*, *configuration=None*)
Registers a representer factory for the given combination of resource class and content type.

Parameters **configuration** (everest.representers.config.RepresenterConfiguration) – representer configuration. A default instance will be created if this is not given.

class everest.representers.base.**ResourceRepresenter**(*resource_class*, *mapping*)
Bases: everest.representers.base.Representer

Base class for resource representers which know how to convert resource representations into resources and back.

This conversion is performed using four customizable, independent helper objects:

1. The *representation parser* responsible for converting the representation into a data element tree;
2. The *data element parser* responsible for converting a data element tree into a resource;
3. The *data element generator* responsible for converting a resource into a data element tree; and
4. the *representation generator* responsible for converting the data element tree into a representation.

configure(*options=None*, *attribute_options=None*)
Configures the attribute mapping for this representer.

Parameters

- **options** (*dict*) – configuration options for the mapping associated with this representer.
- **attribute_options** (*dict*) – attribute options for the mapping associated with this representer.

data_from_representation(*representation*)
Creates a data element from the given representation.

Returns object implementing `everest.representers.interfaces.IExplicitDataElement`

data_from_resource(*resource*)
Extracts managed attributes from a resource and constructs a data element for serialization from it.

Default representer behavior:

- Top-level member and collections resource attributes are represented as links.
- Nested member resource attributes are represented as links, nested collection resource attributes are ignored (building a link may require iterating over the collection).

The default behavior can be configured with the "representer" directive (`everest.configuration.representer()`) by means of the "write_as_link" and "ignore" options of representer configuration objects (`everest.representers.config.RepresenterConfiguration`).

data_from_stream(*stream*)
Creates a data element reading a representation from the given stream.

Returns object implementing `everest.representers.interfaces.IExplicitDataElement`

representation_from_data(*data_element*)
Creates a representation from the given data element.

Parameters **data_element** –

resource_from_data(*data_element*, *resolve_urls=True*)
Extracts serialized data from the given data element and constructs a resource from it.

Parameters **resolve_urls** – If this is set to *False*, resolving URLs in the data element tree will be delayed until after loading has completed.

Returns object implementing `everest.resources.interfaces.IResource`

everest.representers.base.**data_element_tree_to_string**(*data_element*)
Creates a string representation of the given data element tree.

everest.representers.config

Representer configuration.

everest.representers.converters

Converters resource attribute value <-> representation string.

everest.representers.csv

CSV representers.

class everest.representers.csv.**CsvMappingRegistry**
Bases: everest.representers.mapping.SimpleMappingRegistry

Registry for CSV mappings.

class everest.representers.csv.**CsvRepresentationGenerator**(*stream*, *resource_class*, *mapping*)
Bases: everest.representers.base.RepresentationGenerator

A CSV writer for resource data.

Handles linked resources and nested member and collection resources.

everest.representers.dataelements

Data elements.

class everest.representers.dataelements.**CollectionDataElement**
Bases: everest.representers.dataelements.DataElement

Abstract base class for collection data elements.

__len__()
Returns the number of member data elements in this collection data element.

add_member(*data_element*)
Adds the given member data element to this collection data element.

get_members()
Returns all member data elements added to this collection data element.

class everest.representers.dataelements.**DataElement**
Bases: object

Abstract base class for data element classes.

Data elements manage value state during serialization and deserialization. Implementations may need to be adapted to the format of the external representation they manage.

__weakref__
list of weak references to the object (if defined)

classmethod create()
Basic factory method.

classmethod create_from_resource(*resource*)
(Abstract) factory method taking a resource as input.

mapping = None
Static attribute mapping.

class everest.representers.dataelements.**DataElementAttributeProxy**(*data_element*)
Bases: object

Convenience proxy for accessing data from data elements.

The proxy allows you to transparently access terminal, member, and collection attributes. Nested access is also supported.

Example:

```
prx = DataElementAttributeProxy(data_element)
de_id = prx.id                              # terminal access
de_parent = prx.parent                      # member access
de_child = prx.children[0]                  # collection access
de_grandchild = prx.children[0].children[0] # nested collection access
```

__weakref__
list of weak references to the object (if defined)

class everest.representers.dataelements.**LinkedDataElement**
Bases: everest.representers.dataelements.DataElement

Data element managing a linked resource during serialization and deserialization.

class everest.representers.dataelements.**MemberDataElement**
Bases: everest.representers.dataelements.DataElement

Abstract base class for member data element classes.

get_mapped_nested(*attr*)
Returns the mapped nested resource attribute (either a member or a collection resource attribute).

Returns object implementing *:class:IDataelement* or *None* if no nested resource is found for the given attribute name.

get_mapped_terminal(*attr*)
Returns the value for the given mapped terminal resource attribute.

Parameters **attr** (everest.representers.attributes.MappedAttribute) – attribute to retrieve.

Returns attribute value or *None* if no value is found for the given attribute name.

set_mapped_nested(*attr*, *data_element*)
Sets the value for the given mapped nested resource attribute (either a member or a collection resource attribute).

Parameters **data_element** – a :class:DataElement or :class:LinkedDataElement object containing nested resource data.

set_mapped_terminal(*attr*, *value*)
Sets the value for the given mapped terminal resource attribute.

Parameters **value** – value of the attribute to set.

class everest.representers.dataelements.**SimpleCollectionDataElement**
Bases: everest.representers.dataelements._SimpleDataElementMixin, everest.representers.dataelements.CollectionDataElement

Basic implementation of a collection data element.

class everest.representers.dataelements.**SimpleLinkedDataElement**
Bases: everest.representers.dataelements.LinkedDataElement

Basic implementation of a linked data element.

class everest.representers.dataelements.**SimpleMemberDataElement**
Bases: everest.representers.dataelements._SimpleDataElementMixin, everest.representers.dataelements.MemberDataElement

Basic implementation of a member data element.

get_nested(*attr_name*)
Returns the (raw) value of the specified attribute.

Parameters **attr_name** (*str*) – name of the attribute to retrieve.

get_terminal(*attr_name*)
Returns the (raw) value of the specified attribute.

Parameters **attr_name** (*str*) – name of the attribute to retrieve.

set_nested(*attr_name*, *data_element*)
Sets the (raw) value of the specified attribute.

Parameters

- **attr_name** (*str*) – name of the attribute to set.
- **data_element** – a DataElement or LinkedDataElement object containing nested resource data.

set_terminal(*attr_name*, *value*)
Sets the (raw) value of the specified attribute.

Parameters

- **attr_name** (*str*) – name of the attribute to set.
- **value** (*str*) – value of the attribute to set.

everest.representers.interfaces

Interfaces for representers.

everest.representers.mapping

Mapping and mapping registry.

class everest.representers.mapping.**Mapping**(*mapping_registry*, *mapped_class*, *data_element_class*, *configuration*)
Bases: object

Performs configurable resource <-> data element tree <-> representation mappings.

Property mapped_class The resource class mapped by this mapping.

Property data_element_class The data element class for this mapping

__init__(*mapping_registry*, *mapped_class*, *data_element_class*, *configuration*)

Parameters **configuration** – mapping configuration object.

__weakref__
list of weak references to the object (if defined)

configuration
Returns this mapping's configuration object.

get_attribute_map(*mapped_class=None*, *key=None*)
Returns a map of all attributes of the given mapped class.

Parameters **key** – tuple of attribute names specifying a path to a nested attribute in a resource tree. If this is not given, all attributes in this mapping will be returned.

class everest.representers.mapping.**SimpleMappingRegistry**
Bases: everest.representers.mapping.MappingRegistry

Default implementation for a mapping registry using default data element and configuration classes.

collection_data_element_base_class
alias of SimpleCollectionDataElement

linked_data_element_base_class
alias of SimpleLinkedDataElement

member_data_element_base_class
alias of SimpleMemberDataElement

everest.representers.traversal

Resource data tree traversal.

class everest.representers.traversal.**AttributeKey**(*data*)
Bases: object

Value object used as a key during resource data tree traversal.

Each key consists of a tuple of attribute strings that uniquely determine a node's position in the resource data tree.

__weakref__
list of weak references to the object (if defined)

class everest.representers.traversal.**MappingDataElementTreeTraverser**(*root*, *mapping*)
Bases: everest.representers.traversal.MappingResourceDataTreeTraverser

Mapping traverser for data element trees.

class everest.representers.traversal.**MappingResourceDataTreeTraverser**(*root*, *mapping*)
Bases: everest.representers.traversal.DataElementTreeTraverserMixin, everest.representers.traversal.DataTreeTraverser

Abstract base class for resource data tree traversers.

class everest.representers.traversal.**ResourceTreeTraverser**(*root*, *mapping*)
Bases: everest.representers.traversal.MappingResourceDataTreeTraverser

Mapping traverser for resource trees.

everest.representers.urlloader

URL lazy loader.

class everest.representers.urlloader.**LazyAttributeLoaderProxy**(*_loader_map=None*, ***kw*)

Bases: object

Proxy for lazy loading of attributes referencing entities that are loaded through a URL-linked resource.

__weakref__
list of weak references to the object (if defined)

classmethod create(*entity_cls*, *data*)
Factory class method to create a lazy loader for entities linked through resource URLs.

This returns an instance of a new dynamically created subtype of the given entity class which also inherits from this class to add the referenced entity attribute loading functionality. Once all referenced entity attributes have been loaded successfully, the instance's class is reverted to the given entity class.

class everest.representers.urlloader.**LazyUrlLoader**(*url*, *resolver*)
Bases: object

Helper class for lazy loading of URLs.

__weakref__
list of weak references to the object (if defined)

everest.representers.utils

Representer related utilities.

everest.representers.utils.**as_representer**(*resource*, *content_type*)
Adapts the given resource and content type to a representer.

Parameters

- **resource** – resource to adapt.
- **content_type** (*str*) – content (MIME) type to create a representer for.

everest.representers.utils.**get_mapping_registry**(*content_type*)
Returns the data element registry for the given content type (a Singleton).

Note This only works after a representer for the given content type has been created.

everest.representers.xml

XML representers.

class everest.representers.xml.**XmlMappingRegistry**
Bases: everest.representers.mapping.MappingRegistry

Registry for XML mappings.

NS_MAP = {'xsi': 'http://www.w3.org/2001/XMLSchema-instance'}
Static namespace prefix: namespace map.

configuration_class
alias of XmlRepresenterConfiguration

class everest.representers.xml.**XmlRepresenterConfiguration**(*options=None*, *attribute_options=None*)
Bases: everest.representers.config.RepresenterConfiguration

Specialized configuration class for XML representers.

Allowed configuration attribute names:

xml_tag : The XML tag to use for the represented data element class.

xml_schema : The XML schema to use for the represented data element class.

xml_ns : The XML namespace to use for the represented data element class.

xml_prefix : The XML namespace prefix to use for the represented data element class.

3.2.4 Resources

everest.resources.attributes

This file is part of the everest project. See LICENSE.txt for licensing, CONTRIBUTORS.txt for contributor information.

class everest.resources.attributes.**CollectionResourceAttribute**(*name*, *value_type*, *cardinality='ONETOMANY'*, *entity_name=None*, *is_nested=False*)

Bases: everest.resources.attributes._ResourceResourceAttribute

Resource attribute class for collection attribute declarations.

class everest.resources.attributes.**MemberResourceAttribute**(*name*, *value_type*, *cardinality='MANYTOONE'*, *entity_name=None*, *is_nested=False*)

Bases: everest.resources.attributes._ResourceResourceAttribute

Resource attribute class for member attribute declarations.

class everest.resources.attributes.**MetaResourceAttributeCollector**(*mcs*, *name*, *bases*, *class_dict*)

Bases: type

Meta class for member resource classes managing declared attributes.

Extracts relevant information from the resource class descriptors for use e.g. in the representers.

class everest.resources.attributes.**ResourceAttributeKinds**

Bases: object

Static container for resource attribute kind constants.

We have three kinds of resource attribute:

MEMBER : a member resource attribute

COLLECTION : a collection resource attribute

TERMINAL : an attribute that is not a resource

__weakref__
list of weak references to the object (if defined)

class everest.resources.attributes.**TerminalResourceAttribute**(*name*, *value_type*, *entity_name=None*)
Bases: everest.resources.attributes._ResourceAttribute

Resource attribute class for terminal attribute declarations.

everest.resources.base

Resources.

class everest.resources.base.**Collection**(*aggregate*, *name=None*)
Bases: everest.resources.base.Resource

This is an abstract base class for all resource collections. A collection is a set of member resources which can be filtered, sorted, and sliced.

__getitem__(*key*)
Gets a member (by name).

Parameters **key** (string or unicode) – the name of the member

Raises everest.exceptions.DuplicateException if more than one member is found for the given key value.

Returns object implementing everest.resources.interfaces.IMemberResource

__init__(*aggregate*, *name=None*)
Constructor:

Parameters

- **name** (string) – the name of the collection
- **aggregate** (everest.entities.aggregates.Aggregate - an object implementing an interface derived from everest.entities.interfaces.IAggregate) – the associated aggregate

__iter__()
Returns an iterator over the (possibly filtered and ordered) collection.

__len__()
Returns the size (count) of the collection.

add(*member*)
Adds the given member to this collection.

Parameters **member** (object implementing everest.resources.interfaces.IMemberResource) – member to add.

Raises ValueError if a member with the same name exists

clone()
Returns a clone of this collection.

classmethod create_from_aggregate(*aggregate*)
Creates a new collection from the given aggregate.

Parameters **aggregate** (everest.entities.aggregates.Aggregate instance) – aggregate containing the entities exposed by this collection resource

create_member(*entity*)
Creates a new member resource from the given entity and adds it to this collection.

default_order = <everest.querying.specifications.AscendingOrderSpecification object at 0x32603d0>
The default order of the collection's members.

description = ''
A description of the collection.

get(*key*, *default=None*)
Returns a member for the given key or the given default value if no match was found in the collection.

get_aggregate()
Returns the aggregate underlying this collection.

Returns an object implementing everest.entities.interfaces.IAggregate.

max_limit = 1000
The maximum number of member that can be shown on one page (superclass default: 1000).

remove(*member*)
Removes the given member from this collection.

Parameters **member** (object implementing everest.resources.interfaces.IMemberResource) – member to add.

Raises ValueError if the member can not be found in this collection

root_name = None
The name for the root collection (used as URL path to the root collection inside the service).

set_relationship(*relationship*)
Sets the relation parent for this collection.

The relation parent affects the expressions built for filter and order operations.

Parameters **relationship** – relation with another resource, encapsulated in a everest.relationship.Relationship instance.

title = None
The title of the collection.

update_from_data(*data_element*)
Updates this collection from the given data element.

This iterates over the members of this collection and checks if a member with the same ID exists in the given update data. If yes, the existing member is updated with the update member; if no, the member is removed. All data elements in the update data that have no ID are added as new members. Data elements with an ID that can not be found in this collection trigger an error.

Parameters **data_element** (object implementing *:class:everest.resources.interfaces.IExplicitDataElement*) – data element (hierarchical) to create a resource from

Raises ValueError when a data element with an ID that is not present in this collection is encountered.

class everest.resources.base.**Member**(*entity*, *name=None*)
Bases: everest.resources.attributes.ResourceAttributeControllerMixin, everest.resources.base.Resource

This is an abstract class for all member resources.

__eq__(*other*)
Equality operator.

Equality is based on a resource's name only.

__init__(*entity*, *name=None*)
Constructor:

Parameters

- **name** (string) – unique name of the member within the collection
- **entity** (an object implementing an interface derived from everest.entities.interfaces.IEntity) – the associated entity (domain object)

__ne__(*other*)
Inequality operator.

classmethod create_from_entity(*entity*)
Class factory method creating a new resource from the given entity.

delete()
Deletes this member.

Deleting a member resource means removing it from its parent resource.

get_entity()
Returns the entity this resource manages.

Returns an object implementing everest.entities.interfaces.IEntity.

update_from_data(*data_element*)
Updates this member from the given data element.

Parameters **data_element** (object implementing *:class:everest.resources.representers.interfaces.IExplicitDataElemen* – data element (hierarchical) to create a resource from

class everest.resources.base.**Resource**
Bases: object

This is the abstract base class for all resources.

__init__()
Constructor:

__weakref__
list of weak references to the object (if defined)

add_link(*link*)
Adds a link to another resource.

Parameters **link** (everest.resources.base.Link) – a resource link

classmethod create_from_data(*data_element*)
Creates a resource instance from the given data element (tree).

Parameters **data_element** (object implementing everest.resources.representers.interfaces.IExp – data element (hierarchical) to create a resource from

description = ''
Detailed description of this resource.

links = None
A set of links to other resources.

path
Returns the path to this resource in the tree of resources.

relation = None
The relation identifier to show in links to this resource. Needs to be specified in derived classes.

title = ''
Descriptive title for this resource.

urn
Returns the URN for this resource (globally unique identifier).

everest.resources.descriptors

Attribute descriptors for resource classes.

everest.resources.descriptors.**attribute_alias**
Descriptor for declaring an alias to another attribute declared by an attribute descriptor.

everest.resources.descriptors.**attribute_base**
Abstract base class for all attribute descriptors.

Variables

- **attr_type** – the type (or interface) of the controlled entity attribute.
- **entity_attr** – the entity attribute the descriptor references. May be *None*.
- **cardinality** – indicates the cardinality of the relationship for non-terminal attributes. This is always *None* for terminal attributes.
- **id** (*int*) – unique sequential numeric ID for this attribute. Since this ID is incremented each time a new resource attribute is declared, it can be used to establish a well-defined sorting order on all attribute declarations of a resource.
- **resource_attr** – the resource attribute this descriptor is mapped to. This is set after instantiation.

everest.resources.descriptors.**collection_attribute**
Descriptor for declaring collection attributes of a resource as attributes from its underlying entity.

everest.resources.descriptors.**member_attribute**
Descriptor for declaring member attributes of a resource as attributes from its underlying entity.

everest.resources.descriptors.**terminal_attribute**
Descriptor for declaring terminal attributes of a resource as attributes from its underlying entity.

A terminal attribute is an attribute that the framework will not look into any further for querying or serialization.

everest.resources.entitystores

Entity stores.

class everest.resources.entitystores.**CachingEntityStore**(*name*, *join_transaction=False*)
Bases: everest.resources.entitystores.EntityStore

An entity store that caches all entities in memory.

copy()
Returns a deep copy of the entire entity cache.

class everest.resources.entitystores.**DataManager**(*session*)
Bases: object

Data manager to plug an InMemorySession into a zope transaction.

__weakref__
list of weak references to the object (if defined)

class everest.resources.entitystores.**EntityStore**(*name*, *join_transaction=False*)
Bases: object

Base class for all entity stores.

An entity store is responsible for configuration and initialization of a storage backend for entities. It also creates and holds a session factory which is used to create a (thread-local) session. The session alone provides access to the entities loaded from the entity store.

__weakref__
list of weak references to the object (if defined)

configuration
Returns a copy of the configuration for this entity store.

class everest.resources.entitystores.**FileSystemEntityStore**(*name*, *join_transaction=True*)
Bases: everest.resources.entitystores.CachingEntityStore

EntityStore using the file system as storage.

On initialization, this entity store loads resource representations from files into the root repository. Each commit operation writes the specified resource back to file.

commit(*session*)
Dump all resources that were modified by the given session back into the store.

class everest.resources.entitystores.**OrmEntityStore**(*name*, *join_transaction=True*)
Bases: everest.resources.entitystores.EntityStore

EntityStore connected to an ORM backend.

everest.resources.interfaces

Interfaces for resources.

everest.resources.io

Input/Output operations on resources.

class everest.resources.io.**ConnectedResourcesSerializer**(*content_type*, *dependency_graph=None*)
Bases: object

Serializer for a graph of connected resources.

__init__(*content_type*, *dependency_graph=None*)

Parameters

- **content_type** (object implementing `everest.interfaces.IMime`.) – MIME content type to use for representations
- **dependency_graph** – graph determining which resource connections to follow when the graph of connected resources for a given resource is built.

__weakref__
list of weak references to the object (if defined)

to_files(*resource*, *directory*)
Dumps the given resource and all resources linked to it into a set of representation files in the given directory.

to_strings(*resource*)
Dumps the all resources reachable from the given resource to a map of string representations using the specified content_type (defaults to CSV).

Returns dictionary mapping resource member classes to string representations

to_zipfile(*resource*, *zipfile*)
Dumps the given resource and all resources linked to it into the given ZIP file.

class `everest.resources.io.`**ResourceGraph**
Bases: `pygraph.classes.digraph.digraph`

Specialized digraph for resource instances.

Nodes are resources, edges represent relationships between resources. Since resources are wrapper objects generated on the fly, the presence of a resource in the graph is determined by its underlying entity, using the entity class and its ID as a key.

`everest.resources.io.`**build_resource_dependency_graph**(*resource_classes*, *include_backrefs=False*)
Builds a graph of dependencies among the given resource classes.

The dependency graph is a directed graph with member resource classes as nodes. An edge between two nodes represents a member or collection attribute.

Parameters

- **resource_classes** (*sequence of registered resources.*) – resource classes to determine inter-dependencies of.
- **include_backrefs** (*bool*) – flag indicating if dependencies introduced by back-references (e.g., a child resource referencing its parent) should be included in the dependency graph.

`everest.resources.io.`**build_resource_graph**(*resource*, *dependency_graph=None*)
Traverses the graph of resources that is reachable from the given resource.

If a resource dependency graph is given, links to other resources are only followed if the dependency graph has an edge connecting the two corresponding resource classes; otherwise, a default graph is built which ignores all direct cyclic resource references.

Resource a `thelma.resources.MemberResource` instance.

Returns a `ResourceGraph` instance representing the graph of resources reachable from the given resource.

`everest.resources.io.`**dump_resource**(*resource*, *stream*, *content_type=None*)
Dumps the given resource to the given stream using the specified MIME content type (defaults to CSV).

everest.resources.io.**dump_resource_to_files**(*resource*, *content_type=None*, *directory=None*)
Convenience function. See thelma.resources.io.ConnectedResourcesSerializer.to_files() for details.

If no directory is given, the current working directory is used. The given context type defaults to CSV.

everest.resources.io.**dump_resource_to_zipfile**(*resource*, *zipfile*, *content_type=None*)
Convenience function. See thelma.resources.io.ConnectedResourcesSerializer.to_zipfile() for details.

The given context type defaults to CSV.

everest.resources.io.**find_connected_resources**(*resource*, *dependency_graph=None*)
Collects all resources connected to the given resource and returns a dictionary mapping member resource classes to new collections containing the members found.

everest.resources.io.**load_collection_from_file**(*collection*, *filename*, *content_type=None*, *resolve_urls=True*)
Loads resources from the specified file into the given collection resource.

If no content type is provided, an attempt is made to look up the extension of the given filename in the MIME content type registry.

everest.resources.io.**load_collection_from_stream**(*collection*, *stream*, *content_type*, *resolve_urls=True*)
Loads resources from the given stream into the given collection resource.

everest.resources.io.**load_collection_from_url**(*collection*, *url*, *content_type=None*, *resolve_urls=True*)
Loads a collection resource of the given registered resource type from a representation contained in the given URL.

Returns collection resource

everest.resources.io.**load_collections_from_zipfile**(*collections*, *zipfile*, *resolve_urls=True*)
Loads resources contained in the given ZIP archive into each of the given collections.

The ZIP file is expected to contain a list of file names obtained with the get_collection_filename() function, each pointing to a file of zipped collection resource data.

Parameters

- **collections** – sequence of collection resources
- **zipfile** (*str*) – ZIP file name
- **resolve_urls** (*bool*) – Flag indicating if URLs should be resolved during loading.

everest.resources.kinds

Resource kinds.

class everest.resources.kinds.**ResourceKinds**
Bases: object

Static container for resource kind constants.

We have two kinds of resource:

MEMBER : a member resource

COLLECTION : a collection resource

__weakref__
list of weak references to the object (if defined)

everest.resources.link

Resource link.

class everest.resources.link.**Link**(*linked_resource*, *rel*, *type=None*, *title=None*, *length=None*)
Bases: object

A resource link.

::note: The URL for the linked resource is created lazily; at instantiation time, we may not have a request to generate the URL.

__weakref__
list of weak references to the object (if defined)

everest.resources.repository

Resource repository.

class everest.resources.repository.**ResourceRepository**(*entity_repository*)
Bases: everest.repository.Repository

The resource repository manages resource accessors (collections).

everest.resources.service

Service.

class everest.resources.service.**Service**
Bases: everest.resources.base.Resource

The service resource class.

The service resource is placed at the root of the resource tree and provides traversal (=URL) access to all exposed collection resources.

__getitem__(*key*)
Overrides __getitem__ to return a clone of the requested collection.

Parameters **key** (*str*) – collection name.

Returns object implementing everest.resources.interfaces.ICollectionResource.

register(*irc*)
Registers the given resource interface with this service.

start()
Starts the service.

This adds all registered resource interfaces to the service. Multiple calls to this method will only perform the startup once.

everest.resources.system

System resources.

everest.resources.utils

Resource related utilities.

everest.resources.utils.**as_member**(*entity*, *parent=None*)
Adapts an object to a location aware member resource.

Parameters

- **entity** (an object implementing everest.entities.interfaces.IEntity) – a domain object for which a resource adapter has been registered
- **parent** (an object implementing everest.resources.interfaces.ICollectionResource) – optional parent collection resource to make the new member a child of

Returns an object implementing everest.resources.interfaces.IMemberResource

everest.resources.utils.**get_collection_class**(*rc*)
Returns the registered collection resource class for the given marker interface or member resource class or instance.

Parameters **rc** (*class implementing or instance providing or subclass of a registered resource interface.*) – registered resource

everest.resources.utils.**get_member_class**(*rc*)
Returns the registered member class for the given resource.

Parameters **rc** (*class implementing or instance providing or subclass of a registered resource interface.*) – registered resource

everest.resources.utils.**get_resource_url**(*resource*)
Returns the URL for the given resource.

everest.resources.utils.**get_root_collection**(*rc*)
Returns a clone of the collection from the repository registered for the given registered resource.

Parameters **rc** (*class implementing or instance providing or subclass of a registered resource interface.*) – registered resource

everest.resources.utils.**get_stage_collection**(*rc*)
Returns a clone of the collection in the stage repository matching the given registered resource.

Parameters **rc** (*class implementing or instance providing or subclass of a registered resource interface.*) – registered resource

everest.resources.utils.**is_resource_url**(*url_string*)
Checks if the given URL string is a resource URL.

Currently, this check only looks if the URL scheme is either "http" or "https".

everest.resources.utils.**provides_collection_resource**(*obj*)
Checks if the given type or instance provides the everest.resources.interfaces.ICollectionResource interface.

everest.resources.utils.**provides_member_resource**(*obj*)
Checks if the given type or instance provides the everest.resources.interfaces.IMemberResource interface.

everest.resources.utils.**provides_resource**(*obj*)
Checks if the given type or instance provides the everest.resources.interfaces.IResource interface.

3.2.5 Views

everest.views.base
everest.views.deletemember
everest.views.getcollection
everest.views.getmember
everest.views.interfaces
everest.views.postcollection
everest.views.putmember
everest.views.static
everest.views.utils

everest.views.base

View base classes.

class everest.views.base.**GetResourceView**(*resource*, *request*)
Bases: everest.views.base.ResourceView

Abstract base class for all collection views

exception everest.views.base.**HttpWarningResubmit**(*detail=None*, *headers=None*, *comment=None*)
Bases: paste.httpexceptions.HTTPTemporaryRedirect

Special 307 HTTP Temporary Redirect exception which transports

class everest.views.base.**PutOrPostResourceView**(*resource*, *request*)
Bases: everest.views.base.ResourceView

Abstract base class for all member views

class everest.views.base.**ResourceView**(*context*, *request*)
Bases: object

Abstract base class for all resource views.

Resource views know how to handle a number of things that can go wrong in a REST request.

__weakref__
list of weak references to the object (if defined)

class everest.views.base.**ViewUserMessageChecker**
Bases: everest.messaging.UserMessageChecker

Custom user message checker for views.

check()
Implements user message checking for views.

Checks if the current request has an explicit "ignore-message" parameter (a GUID) pointing to a message with identical text from a previous request, in which case further processing is allowed.

create_307_response()
Creates a 307 "Temporary Redirect" response including a HTTP Warning header with code 299 that contains the user message received during processing the request.

everest.views.deletemember

Delete member view.

class everest.views.deletemember.**DeleteMemberView**(*context*, *request*)
Bases: everest.views.base.ResourceView

A View for processing DELETE requests

The client sends a DELETE request to the URI of a Member Resource. If the deletion is successful, the server responds with a status code of 200.

In a RESTful server DELETE does not always mean "delete a record from the database". See RESTful Web Services and REST in Practice books.

See http://bitworking.org/projects/atom/rfc5023.html#delete-via-DELETE

everest.views.getcollection

Get collection view.

class everest.views.getcollection.**GetCollectionView**(*resource*, *request*)
Bases: everest.views.base.GetResourceView

View for GET requests on collection resources.

everest.views.getmember

Get member view.

class everest.views.getmember.**GetMemberView**(*resource*, *request*)
Bases: everest.views.base.GetResourceView

View for GET requests on member resources.

everest.views.interfaces

Interfaces for views.

everest.views.postcollection

Post collection view.

class everest.views.postcollection.**PostCollectionView**(*resource*, *request*)
Bases: everest.views.base.PutOrPostResourceView

View for POST requests on collection resources.

The client POSTs a representation of the member to the URI of the collection. If the new member resource was created successfully, the server responds with a status code of 201 and a Location header that contains the IRI of the newly created resource and a representation of it in the body of the response.

See http://bitworking.org/projects/atom/rfc5023.html#post-to-create

everest.views.putmember

Put member view.

class everest.views.putmember.**PutMemberView**(*resource*, *request*)
Bases: everest.views.base.PutOrPostResourceView

View for PUT requests on member resources.

The client sends a PUT request to store a representation of a Member Resource. If the request is successful, the server responds with a status code of 200.

See http://bitworking.org/projects/atom/rfc5023.html#edit-via-PUT

everest.views.static

Static view.

everest.views.utils

View related utilities.

everest.views.utils.**accept_csv_only**(*context*, *request*)
This can be used as a custom predicate for view configurations with a CSV renderer that should only be invoked if this has been explicitly requested in the ACCEPT header by the client.

3.2.6 Core Modules

everest.batch
everest.configuration
everest.directives
everest.exceptions
everest.ini
everest.interfaces
everest.messaging
everest.mime
everest.orm
everest.relationship
everest.renderers
everest.repository
everest.root
everest.testing
everest.traversal
everest.url
everest.utils

everest.batch

Batch.

class everest.batch.**Batch**(*start*, *size*, *total_size*)
Bases: object

Helper class to manage batches in a sequence.

__init__ (*start*, *size*, *total_size*)

Parameters

- **start** (*int*) – start index for this batch.
- **size** (*int*) – batch size.
- **total_size** (*int*) – total size of the batched sequence.

__weakref__
list of weak references to the object (if defined)

first
Returns the first batch for the batched sequence.

Return type `Batch` instance.

index
Returns the index of this batch in the batched sequence.

Return type integer

last
Returns the last batch for the batched sequence.

Return type `Batch` instance.

next
Returns the next batch for the batched sequence or *None*, if this batch is already the last batch.

Return type `Batch` instance or *None*.

number
Returns the number of batches the batched sequence contains.

Return type integer.

previous
Returns the previous batch for the batched sequence or *None*, if this batch is already the first batch.

Return type `Batch` instance or *None*.

everest.configuration

Configurator for everest.

class `everest.configuration.`**Configurator**(*registry=None*, *package=None*, *filter_specification_factory=None*, *order_specification_factory=None*, *service=None*, *filter_builder=None*, *filter_director=None*, *cql_filter_specification_visitor=None*, *sql_filter_specification_visitor=None*, *eval_filter_specification_visitor=None*, *order_builder=None*, *order_director=None*, *cql_order_specification_visitor=None*, *sql_order_specification_visitor=None*, *eval_order_specification_visitor=None*, *url_converter=None*, ***kw*)

Bases: `pyramid.configuration.Configurator`

Configurator for everest.

get_registered_utility(**args*, ***kw*)
Convenience method for obtaining a utility from the registry.

query_registered_utilities(**args*, ***kw*)
Convenience method for querying a utility from the registry.

everest.directives

ZCML directives for everest.

class everest.directives.**ResourceDirective**(*context*, *interface*, *member*, *entity*, *collection=None*, *collection_root_name=None*, *collection_title=None*, *repository=None*, *expose=True*)
Bases: zope.configuration.config.GroupingContextDecorator

Directive for registering a resource. Calls everest.configuration.Configurator.add_resource().

class everest.directives.**ResourceRepresenterDirective**(*context*, *content_type*, *kind=None*)
Bases: zope.configuration.config.GroupingContextDecorator

Grouping directive for registering a representer for a given resource(s) and content type combination. Delegates the work to a everest.configuration.Configurator.

everest.directives.**filesystem_repository**(*_context*, *name=None*, *make_default=False*, *aggregate_class=None*, *entity_store_class=None*, *directory=None*, *content_type=None*)
Directive for registering a file-system based repository.

everest.directives.**messaging**(*_context*, *repository*, *reset_on_start=True*)
Directive for setting up the user message resource in the appropriate repository.

Parameters **repository** (*str*) – The repository to create the user messages resource in.

everest.directives.**orm_repository**(*_context*, *name=None*, *make_default=False*, *aggregate_class=None*, *entity_store_class=None*, *db_string=None*, *metadata_factory=None*)
Directive for registering an ORM based repository.

everest.exceptions

Custom exceptions.

exception everest.exceptions.**DuplicateException**
Bases: exceptions.Exception

Raised when more than one item was found where one was expected.

__weakref__
list of weak references to the object (if defined)

exception everest.exceptions.**UnsupportedOperationException**
Bases: exceptions.Exception

Raise this to indicate that the requested operation is not supported.

__weakref__
list of weak references to the object (if defined)

everest.interfaces

Interfaces for everest.

everest.messaging

Message notification and handling.

class everest.messaging.**UserMessageChecker**
Bases: object

Abstract base class for user message checkers.

User message checkers can be used to decide if further processing should be stopped in response to a non-critical event reported through a user message.

__weakref__
list of weak references to the object (if defined)

class everest.messaging.**UserMessageNotifier**
Bases: object

Notifier for user messages.

__weakref__
list of weak references to the object (if defined)

class everest.messaging.**UserMessageHandlingContextManager**(*checker*)
Bases: object

A context which sets up a user message checker as a subscriber to user messages.

__init__(*checker*)
Constructor.

Parameters **checker** (everest.messaging.UserMessageChecker instance.) – The user message checker to subscribe to user messages.

__weakref__
list of weak references to the object (if defined)

everest.mime

MIME (content) types.

everest.orm

ORM related services.

class everest.orm.**OrderClauseList**(**clauses*, ***kwargs*)
Bases: sqlalchemy.sql.expression.ClauseList

Custom clause list for ORDER BY clauses.

Suppresses the grouping parentheses which would trigger a syntax error.

everest.orm.**Session = <sqlalchemy.orm.scoping.ScopedSession object at 0x313b350>**
The scoped session maker. Instantiate this to obtain a thread local session instance.

everest.orm.**as_slug_expression**(*attr*)
Converts the given instrumented string attribute into an SQL expression that can be used as a slug.

Slugs are identifiers for members in a collection that can be used in an URL. We create slug columns by replacing non-URL characters with dashes and lower casing the result. We need this at the ORM level so that we can use the slug in a query expression.

everest.orm.**clear_mappers**()
Clears all mappers set up by SA and also clears all custom "id" and "slug" attributes inserted by the `mapper()` function in this module.

This should only ever bee needed in a testing context.

everest.orm.**commit_veto**(*environ*, *status*, *headers*)
Strict commit veto to use with the transaction manager.

Unlike the default commit veto supplied with the transaction manager, this will veto all commits for HTTP status codes other than 2xx unless a commit is explicitly requested by setting the "x-tm" response header to "commit".

everest.orm.**empty_metadata**(*engine*)
The default metadata factory.

everest.orm.**mapper**(*class_*, *local_table=None*, *id_attribute='id'*, *slug_expression=None*, **args*, ***kwargs*)
Convenience wrapper around the SA mapper which will set up the hybrid "id" and "slug" attributes required by everest after calling the SA mapper.

If you (e.g., for testing purposes) want to clear mappers created with this function, use the `clear_mappers()` function in this module.

Parameters

- **id_attribute** (*str*) – the name of the column in the table to use as ID column (will be aliased to a new "id" attribute in the mapped class)
- **slug_expression** – function to generate a slug SQL expression given the mapped class as argument.

everest.relationship

Parent/child relationship between entities or resources.

class everest.relationship.**Relationship**(*parent*, *children=None*, *backref=None*)
Bases: object

Represents a nested relationship between a parent object and a collection of child objects.

This is used for deferred access of child objects and for dynamic creation of a filter specification for the children.

Variables

- **parent** – parent object
- **children** – child object collection
- **backref** – name of the attribute referencing the parent in each child object.

__weakref__
list of weak references to the object (if defined)

everest.renderers

Renderers.

everest.repository

Repository base class.

class everest.repository.**Repository**
Bases: object

Abstract base class for repositories.

The repository creates accessors on the fly, caches them, and returns a clone.

__weakref__
list of weak references to the object (if defined)

clear(*rc*)
Clears the accessor for the given registered resource.

clear_all()
Clears all accessors.

configure(***config*)
Configures this repository.

get(*rc*)
Returns an accessor for the given registered resource.

If this is the first request, an instance is created on the fly using the new() method and cached. The method always returns a clone of the cached accessor; this clone can later be used to look up the repository it was obtained from using the get_repository() class method.

initialize()
Initializes this repository.

new(*rc*)
Returns a new accessor for the given registered resource.

set(*rc*, *obj*)
Makes the given accessor the one to use for the given registered resource.

everest.repository.**as_repository**(*rc*)
Adapts the given registered resource to its configured repository.

Returns object implementing everest.resources.interfaces.IRepository.

everest.root

Root factory.

everest.traversal

Custom resource object tree traverser.

class everest.traversal.**SuffixResourceTraverser**(*root*)
Bases: pyramid.traversal.ResourceTreeTraverser

A custom model traverser that allows us to specify the representation for resources with a suffix as in *http://everest/racks.csv*.

Rather than to reproduce the functionality of the *__call__* method, we check if base part of the current view name (*racks* in the example) can be retrieved as a child resource from the context. If yes, we set the context to the resource and the view name to the extension part of the current view name (*csv* in the example); if no, nothing is changed.

everest.url

URL <-> resource conversion.

class everest.url.**ResourceUrlConverter**(*request*)
Bases: object

Performs URL <-> resource instance conversion.

See http://en.wikipedia.org/wiki/Query_string for information on characters supported in query strings.

__weakref__
list of weak references to the object (if defined)

url_to_resource(*url*)
Converts the given url into a resource.

Parameters **url** (*str*) – URL to convert

Returns member or collection resource

::note [If the query string in the URL has multiple values for a] query parameter, the last definition in the query string wins.

everest.utils

General purpose utilities.

class everest.utils.**BidirectionalLookup**(*init_map=None*, *map_type=<type 'dict'>*)
Bases: object

Bidirectional mapping between a left and a right collection of items.

Each element of the left collection is mapped to exactly one element of the right collection; both collections contain unique elements.

__init__(*init_map=None*, *map_type=<type 'dict'>*)

Parameters

- **init_map** – map-like object to initialize this instance with
- **map_type** – type to use for the left and right item maps (dictionary like)

__weakref__
list of weak references to the object (if defined)

everest.utils.**check_email**()
match(string[, pos[, endpos]]) –> match object or None. Matches zero or more characters at the beginning of the string

everest.utils.**classproperty**
Property descriptor for class objects.

everest.utils.**get_filter_specification_visitor**(*name*)
Returns a the class registered as the filter specification visitor utility under the given name (one of the everest.querying.base.EXPRESSION_KINDS constants).

Returns class implementing everest.interfaces.IFilterSpecificationVisitor

everest.utils.**get_order_specification_visitor**(*name*)
Returns the class registered as the order specification visitor utility under the given name (one of the everest.querying.base.EXPRESSION_KINDS constants).

Returns class implementing everest.interfaces.IOrderSpecificationVisitor

everest.utils.**get_repository_manager**()
Registers the object registered as the repository manager utility.

Returns object implementing everest.interfaces.IRepositoryManager

CHAPTER

FOUR

INDICES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

e