Bazaar 0.99.0 API Documentation

API Documentation

February 2, 2004

Contents

1 Package bazaar

Bazaar is an easy to use and powerful abstraction layer between relational database and object oriented application.

Features:

- easy to use - define classes and programmer is ready to get and modify application data in object-oriented way (no additional steps such as code generation is required)
- object-oriented programing and feel - using classes, objects and references instead of relations, its columns, primary and foreign keys
- object-oriented database operations:
  - add, update, delete
  - get and reload
  - easy object finding with support for SQL queries
  - association data load and reload
- application class relationships:
  - one-to-one, one-to-many and many-to-many
  - uni-directional and bi-directional
- application objects and association data cache integrated with Python garbage collector:
  - full - load all rows at once from relation
  - lazy - load one row from relation
- configurable - connection string, DB API module, class relations, object and association data cache types, etc.

Requirements:

- Python 2.3
- Python DB API 2.0 module with "format" and "pyformat" parameter style support (tested with psycopg 1.1.10[1])
- RDBMS (tested with PostgreSQL 7.4[2])

This is free software distributed under GNU Lesser General Public License[3]. Download it from project's page[4] on Savannah[5].

Bazaar is easy to use, but is designed for people who know both object-oriented and relational technologies, their advantages, disadvantages and differences between them ("The Object-Relational Impedance Mismatch"[6] reading is recommended).

(section) Using the layer

(section) Creating application classes

Let's consider following diagram:

```
Order < 1 ---- * > OrderItem
OrderItem | * ---- 1 > Article
```

[1] http://initd.org/software/psycopg
[2] http://www.postgresql.org
[3] http://www.gnu.org/licenses/lgpl.html
[4] http://savannah.nongnu.org/projects/bazaar/
[5] http://savannah.nongnu.org
[6] http://www.agiledata.org/essays/impedancemismatch.html

There are three classes and two associations. Both relationships are one to many associations, but first one is bi-directional and second is uni-directional.

Class definition (more about class and relationships defining can be found in `bazaar.conf` module documentation) should be like:

```
# import bazaar module used to create classes
import bazaar.conf

# create class for articles
# class name is specified, relation equals to class name
Article = bazaar.conf.Persistence('Article')

# add class attributes and relation columns
# class attribute name is the same as relation column name
Article.addColumn('name')
Article.addColumn('price')

# create order and order items classes
# class names are different than database relation names
Order = bazaar.conf.Persistence('Order', relation = 'order')
OrderItem = bazaar.conf.Persistence('OrderItem', relation = 'order_item')

Order.addColumn('no')            # order number
OrderItem.addColumn('pos')       # order item position
OrderItem.addColumn('quantity') # article quantity

# define bi-directional association between Order and OrderItem classes
#
# from OrderItem perspective
# attribute name: order
# relation column name: order_fkey
# referenced object's class: Order
# referenced object's class attribute name: items
OrderItem.addColumn('order', 'order_fkey', Order, vattr = 'items')

# from Order perspective
# attribute name: items
# referenced object's class: OrderItem
# referenced relation column name: order_fkey
# referenced object's class attribute name: order
Order.addColumn('items', vcls = OrderItem, vcol = 'order_fkey', vattr = 'order')

# define uni-directional association between OrderItem and Article classes
#
# attribute name: article
# relation column name: article_fkey
# referenced object's class: Article
OrderItem.addColumn('article', 'article_fkey', Article)
```

Now, SQL schema can be created:

```
# primary key values generator
```

```
create sequence order_seq;
create table "order" (
    # every application object is identified with __key__ attribute
    __key__     integer,
    no           integer not null unique,
    finished     boolean not null,
    primary key (__key__)
);

create sequence article_seq;
create table article (
    __key__     integer,
    name         varchar(20) not null,
    price        numeric(10,2) not null,
    unique (name),
    primary key (__key__)
);

create sequence order_item_seq;
create table order_item (
    __key__     integer,
    order_fkey   integer,
    pos          integer not null,
    article_fkey integer not null,
    quantity     numeric(10,3) not null,
    primary key (__key__),
    unique (order_fkey, pos),

    # association between Order and OrderItem
    foreign key (order_fkey) references "order"(__key__),

    # association between OrderItem and Article
    foreign key (article_fkey) references article(__key__)
);
```

(section) Application code

Application must import Bazaar core module:

```
import bazaar.core
import psycopg
```

DB API module is imported, too. However, it is not obligatory because it can be specified in config file, see `bazaar.config` module documentation for details.

Create Bazaar layer instance. There are several parameters (`bazaar.core.Bazaar`), but now in this example only list of application classes and DB API module are specified:

```
bzr = bazaar.core.Bazaar((Article, Order, OrderItem), dbmod = psycopg)
```

Connect to database:

```
bzr.connectDB('dbname = ord')
```

Connection string is standard database source name (dsn) described in DB API 2.0 specification. Connection can be established with `bazaar.core.Bazaar` class constructor, too.

Create application object:

```
apple = Article()
apple.name = 'apple'
apple.price = 2.33
```

Object constructor can initialize object attributes:

```
oi1 = OrderItem(pos = 1, quantity = 10)
oi1.article = apple

peach = Article()
peach.name = 'peach'
peach.price = 2.34

oi2 = OrderItem(article = peach)
oi2.pos = 2
oi2.quantity = 40
```

Create new order:

```
ord = Order(no = 1)
```

Append order items to order. It can be made in two ways (it is bi-directional relationship):

```
ord.items.append(oi1)
oi2.order = ord
```

Finally, add created objects data into database:

```
bazaar.add(apple)
bazaar.add(peach)
bazaar.add(oi1)
bazaar.add(oi2)
bazaar.add(ord)
```

Objects can be updated and deleted, too (`bazaar.core.Bazaar`).

Now, let's play with some objects. Remove second order item from order no 1:

```
del ord.items[oi2]
```

And update association:

```
ord.items.update()
```

Add second order item again:

```
ord.items.append(oi2)
ord.items.update()
```

Change apple price:

```
apple.price = 2.00
```

And update database data:

```
bzr.update(apple)
```

Print all orders:

```
for ord in bzr.getObjects(Order):
    print ord
```

Find order number 1:

```
bzr.find(Order, {'no': 1})
```

Find order items for article "apple":

```
bzr.find(OrderItem,  {'article': apple})
```

Finally, commit transaction:

```
bzr.commit()
```

1.1 Modules

- **assoc**: Association classes.
  (Section 2, p. 9)
- **cache**: Cache and reference buffer classes.
  (Section 3, p. 24)
- **conf**: Provides classes for mapping application classes to database relations.
  (Section 4, p. 33)
- **config**: Module contains basic classes for Bazaar layer configuration.
  (Section 5, p. 39)
- **core**: This module contains basic Bazaar implementation.
  (Section 6, p. 43)
- **exc**: Bazaar exceptions.
  (Section 7, p. 50)
- **motor**: Data convertor and database access classes.
  (Section 8, p. 53)

2 Module bazaar.assoc

Association classes.

There are several types of associations:

- one-to-one
- one-to-many
- many-to-many

All of them can be:

- uni-directional
- bi-directional

Defining associations between application classes is described in documentation of `bazaar.conf` module.

Referenced objects are accessed in object-oriented manner:

```
# create objects
ord = Order()
oi  = OrderItem()
art = Article()

# assign reference
oi.article = art

# append object reference to list of objects of one-to-many association
ord.items.append(oi)

for oi in ord.items:
    print oi.article
```

Getting object reference (`oi.article`) or iterator of referenced objects (`ord.items`) is performed with descriptors (see below). Iterator of objects is implemented with `bazaar.assoc.ObjectIterator` class. The class makes possible operating on objects, i.e. appending objects into relationship.

One side of specific relationship is realized with one class descriptor:

```
+-----------------------------------------------------------------------+
| Association  |   Uni-directional   |          Bi-directional          |
+-----------------------------------------------------------------------+
|              |    A     |    B     |        A        |        B        |
+-----------------------------------------------------------------------+
| one-to-one   | OneToOne | OneToOne | BiDirOneToOne   | BiDirOneToOne   |
| one-to-many  | OneToOne | ---      | OneToMany       | BiDirOneToOne   |
| many-to-many | List     | List     | BiDirManyToMany | BiDirManyToMany |
+-----------------------------------------------------------------------+
```

where:

```
A < 1 ------ 1 > B         one-to-one
A < 1 ------ * > B         one-to-many
A < * ------ * > B         many-to-many
```

2.1 Functions

juggle(*obj*, *value*, *app*, *rem*)

Dictionaries `app` and `rem` contain sets of referenced objects indexed by application objects `obj`.
Function appends referenced object `value` to set `app[obj]` and removes it from `rem[obj]`. If set `rem[obj]` contains no values, then it is deleted.

2.2 Variables

Name	Description
log	**Value:** `<logging.Logger instance at 0x404585ec>`

2.3 Class AssociationReferenceProxy

```
__builtin__.object ─┐
                    AssociationReferenceProxy
```

Known Subclasses: List, OneToOne

Association reference proxy class for application objects.

Reference proxy allows to get (upon foreign key value of object's column) and set (upon primary key value of referenced object) reference to other application object (referenced object).

There should be one reference proxy object per association between application classes.

It is allowed to set reference to:

- application object with primary key
- application object without primary key (object is not completed nor stored in database)
- None (NULL) value

When referenced object has no primary key, then reference proxy buffers the object as value with reference buffer.

Application class attribute `col` defines parameters of association.

See Also: `bazaar.cache.ReferenceBuffer` `bazaar.cache.ListReferenceBuffer` `bazaar.conf.Persistence` `bazaar.conf.Column`

2.3.1 Methods

__init__(*self*, *col*, *ref_buf*=`None`)

Create association reference proxy.
Brokers are not initialized with the constructor. Instead, they are set when Bazaar layer is started up.

Parameters

`col`: Application object's class attribute.

Overrides: __builtin__.object.__init__
See Also: `bazaar.core.Bazaar.__init__` `bazaar.conf.Persistence` `bazaar.conf.Column`

save(*self*, *obj*, *value*)

Assign referenced object.
If primary key value of referenced object is not defined, then it is stored in reference buffer, otherwise it's set with `saveForeignKey` method.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

See Also: `saveForeignKey` `bazaar.cache.ReferenceBuffer` `bazaar.cache.ListReferenceBuffer`

saveForeignKey(*self*, *obj*, *vkey*)

Abstract method to save referenced object's primary key value.

Parameters

`obj:`	Application object.
`vkey:`	Referenced object primary key value.

See Also: `bazaar.assoc.List` `bazaar.assoc.OneToOne`

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__

2.3.2 Instance Variables

Name	Description
association	Referenced class' association object of bi-directional association.
broker	Broker of application class.
col	Application object's class attribute.
vbroker	Broker of referenced application objects' class.

2.4 Class BiDirList

```
__builtin__.object ─┐
                    │
bazaar.assoc.AssociationReferenceProxy ─┐
                                        │
                     bazaar.assoc.List ─┐
                                        │
                                    BiDirList
```

Known Subclasses: BiDirManyToMany, OneToMany

Basic bi-directional one-to-many and many-to-many association descriptor.

2.4.1 Methods

append(*self*, *obj*, *value*)

Append referenced object to association and integrate association data.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

Overrides: bazaar.assoc.List.append

integrateRemove(*self*, *obj*, *value*)

Integrate association data when referenced object is removed from association.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

integrateSave(*self*, *obj*, *value*)

Integrate association data when referenced object is appended to association.

remove(*self*, *obj*, *value*)

Remove referenced object from association and integrate association data.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

Overrides: bazaar.assoc.List.remove

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__
Inherited from AssociationReferenceProxy: save
Inherited from List: __init__, __get__, __set__, addAscData, appendKey, contains, delAscData, getAllKeys, iterObjects, justRemove, len, loadData, reloadData, saveForeignKey, update, updateableAscData

2.4.2 Instance Variables

Name	Description
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)*	
Inherited from List: appended *(p. 15)*, cache *(p. 15)*, reload *(p. 15)*, removed *(p. 15)*	

2.5 Class BiDirManyToMany

```
__builtin__.object ─┐
                    │
bazaar.assoc.AssociationReferenceProxy ─┐
                                        │
                  bazaar.assoc.List ─┐
                                     │
                  bazaar.assoc.BiDirList ─┐
                                          │
                                          BiDirManyToMany
```

Bi-directional many-to-many association descriptor.

2.5.1 Methods

appendKey(*self*, *okey*, *vkey*)

Append referenced object relational data to association data and update association data of referenced class.

Parameters

`okey`: Application object's primary key value.
`vkey`: Referenced object's primary key value.

Overrides: bazaar.assoc.List.appendKey

loadData(*self*)

Load association data from database.

Overrides: bazaar.assoc.List.loadData
See Also: `reloadData` `appendKey`

reloadData(*self*, *now*=`False`)

Request reloading association relational data. Referenced class' association data are reloaded, too. Association data are removed from memory. If `now` is set to true, then relationship data are loaded from database immediately.

Parameters

`now`: Reload relationship data immediately.

Overrides: bazaar.assoc.List.reloadData

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__
Inherited from AssociationReferenceProxy: save
Inherited from BiDirList: append, integrateRemove, integrateSave, remove
Inherited from List: __init__, __get__, __set__, addAscData, contains, delAscData, getAllKeys, iterObjects, justRemove, len, saveForeignKey, update, updateableAscData

2.5.2 Instance Variables

Name	Description
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)* **Inherited from List:** appended *(p. 15)*, cache *(p. 15)*, reload *(p. 15)*, removed *(p. 15)*	

2.6 Class BiDirOneToOne

```
__builtin__.object ─┐
                    │
bazaar.assoc.AssociationReferenceProxy ─┐
                                        │
                    bazaar.assoc.OneToOne ─┐
                                           │
                                      BiDirOneToOne
```

Bi-directional one-to-one association descriptor.

See Also: `AssociationReferenceProxy` `OneToOne`

2.6.1 Methods

__set__(*self*, *obj*, *value*)

Descriptor method to set application object's attribute and foreign key values.
The method keeps data integrity of bi-directional one-to-one association.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

Overrides: bazaar.assoc.OneToOne.__set__

integrateRemove(*self*, *obj*, *value*)

Keep bi-directional association data integrity when removal of reference is performed.
Application and referenced objects cannot be `None`.
Method is called by second association object from bi-directional relationship.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

integrateSave(*self*, *obj*, *value*)

Keep bi-directional association data integrity when setting reference is performed.
Application and referenced objects cannot be `None`.
Method is called by second association object from bi-directional relationship.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

Inherited from type: __new__
Inherited from AssociationReferenceProxy: __init__, save
Inherited from OneToOne: __get__, saveForeignKey

2.6.2 Instance Variables

Name	Description
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)*	

2.7 Class List

```
__builtin__.object ─┐
                    │
bazaar.assoc.AssociationReferenceProxy ─┐
                                        │
                                      List
```

Known Subclasses: BiDirList

Basic descriptor for one-to-many and many-to-many associations.

2.7.1 Methods

__init__(*self*, *col*)

Create descriptor for one-to-many and many-to-many associations.

Parameters

`col`: Referenced application object's class column.

Overrides: bazaar.assoc.AssociationReferenceProxy.__init__
See Also: `bazaar.assoc.AssociationReferenceProxy.__init__` `bazaar.core.Bazaar.__init__` `bazaar.conf.Persistence` `bazaar.conf.Column`

__get__(*self*, *obj*, *cls*)

Descriptor method to get iterator of referenced objects.
For example, to get list of all referenced objects of specific order `ord` (items is the descriptor):

```
order_item_list = list(ord.items)
```

or to get a set:

```
order_item_set = sets.Set(ord.items)
```

Parameters

`obj`: Application object.
`cls`: Application class.

Return Value

Iterator of referenced objects, when `obj` is not null, otherwise descriptor object.

See Also: `bazaar.assoc.ObjectIterator`

__set__(*self*, *obj*, *value*)

Assigning list of referenced objects is not implemented yet.

addAscData(*self*, *pairs*)

Add pair of application object's and referenced object's primary key values into database.

See Also: `update`

append(*self*, *obj*, *value*)

Append referenced object to association.

Parameters

`obj:`	Application object.
`value:`	Referenced object.

appendKey(*self*, *okey*, *vkey*)

Append referenced object relational data to association data.

Parameters

`okey:`	Application object's primary key value.
`vkey:`	Referenced object's primary key value.

contains(*self*, *obj*, *value*)

Check if object is referenced by application object.

Parameters

`obj:`	Application object.
`value:`	Object to check.

Return Value

True if object is referenced by application object.

delAscData(*self*, *pairs*)

Remove pair of application object's and referenced object's primary key values from database.

See Also: `update`

getAllKeys(*self*)

Return tuple of application object's and referenced object's primary key values.
Referenced object's primary key value is taken from database with appropriate convertor methods.

See Also: `bazaar.motor.Convertor.getAscData`

iterObjects(*self*, *obj*)

Return iterator of all referenced objects by application object.

Parameters

`obj`: Application object.

Return Value

Iterator of all referenced objects.

justRemove(*self*, *obj*, *value*)

Remove referenced object from association.

Parameters

`obj`: Application object.
`value`: Referenced object.

len(*self*, *obj*)

Return amount of all referenced objects by application object.

loadData(*self*)

Load association data from database.

See Also: `reloadData` `appendKey`

reloadData(*self*, *now*=`False`)

Request reloading association relational data.
Association data are removed from memory. If `now` is set to true, then relationship data are loaded from database immediately.

Parameters

`now`: Reload relationship data immediately.

remove(*self*, *obj*, *value*)

Remove referenced object from association and update information about data removal.

Parameters

`obj`: Application object.
`value`: Referenced object.

saveForeignKey(*self*, *obj*, *vkey*)

Save referenced object's primary key value.
Primary key value is appended to the set of referenced objects' primary key values.

Parameters

`obj`: Application object.
`vkey`: Referenced object's primary key value.

Overrides: bazaar.assoc.AssociationReferenceProxy.saveForeignKey

update(*self*, *obj*)

Update in database relational data of association of given application object.

Parameters
`obj`: Application object.

See Also: `bazaar.assoc.OneToMany.updateReferencedObjects` `bazaar.assoc.OneToMany.addReferencedObjects` `bazaar.assoc.OneToMany.delReferencedObjects`

updateableAscData(*self*, *obj*, *value*)

Return pair of application object's and referenced object's primary key values.
The data will be used to update relationship in database.

See Also: `update`

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__
Inherited from AssociationReferenceProxy: save

2.7.2 Instance Variables

Name	Description
appended	Sets of referenced objects appended to association.
cache	Association data cache - sets of referenced objects's primary key values per application object.
reload	If true, then association data will be loaded from database.
removed	Sets of referenced objects removed from association.
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)*	

2.8 Class ObjectIterator

```
__builtin__.object ─┐
                    │
              ObjectIterator
```

Iterator of referenced objects of one-to-many and many-to-many associations.

The iterator is used to append, remove and update referenced objects, which are associated with application object.

For example, to print articles of order's items:

```
items = order.items    # get ObjectIterator object
for oi in items:
    print oi.article
```

Several operators are supported

- len: `len(items)`
- in: `oi in items`
- del: `del items[oi]`

See Also: `AssociationReferenceProxy`

2.8.1 Methods

__init__(*self*, *obj*, *association*)

Create iterator of referenced objects.

Parameters

`obj:`	Application object.
`association:`	One-to-many or many-to-many association object.

Overrides: __builtin__.object.__init__

__contains__(*self*, *value*)

Check if object is referenced in the relationship.

Parameters

`value:` Object to check.

Return Value

True if object is referenced.

__delitem__(*self*, *value*)

Remove referenced object from association.
Method works for bi-directional associations, too:

```
   order.items.remove(oi)        # oi.order == None
```

or:

```
   del order.items[oi]           # oi.order == None
```

Referenced object cannot be `None`.

Parameters

`value:` Referenced object.

__iter__(*self*)

Iterator interface method.

Return Value

Iterator of all referenced objects got from association object.

__len__(*self*)

Return amount of referenced objects.

append(*self*, *value*)

Associate referenced object with application object.
Method works for bi-directional associations, too:
```
oi = OrderItem()
order.items.append(oi)        # oi.order == oi
```
Referenced object cannot be `None`.

Parameters
`value`: Referenced object.

remove(*self*, *value*)

Remove referenced object from association.
Method works for bi-directional associations, too:
```
order.items.remove(oi)        # oi.order == None
```
or:
```
del order.items[oi]           # oi.order == None
```
Referenced object cannot be `None`.

Parameters
`value`: Referenced object.

update(*self*)

Update association data in database.

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__

2.8.2 Instance Variables

Name	Description
association	Association object.
obj	Application object.

2.9 Class OneToMany

```
__builtin__.object
bazaar.assoc.AssociationReferenceProxy
                    bazaar.assoc.List
                    bazaar.assoc.BiDirList
                                        OneToMany
```

One-to-many association descriptor.

One-to-many association defined on "one" side is always bi-directional relationship.

2.9.1 Methods

__init__(*self*, *col*)

Create descriptor for one-to-many associations.

Parameters

`col`:	Referenced application object's class column.

Overrides: bazaar.assoc.List.__init__
See Also: `bazaar.assoc.AssociationReferenceProxy.__init__`

addReferencedObjects(*self*, *pairs*)

Add referenced objects into database.
The method is used as `addAscData` method with one-to-many associations when updating relationship.

See Also: `delReferencedObjects` `updateReferencedObjects` `update`

append(*self*, *obj*, *value*)

Append referenced object to association and integrate association data.

Parameters

`obj`:	Application object.
`value`:	Referenced object.

Overrides: bazaar.assoc.BiDirList.append

delReferencedObjects(*self*, *pairs*)

Delete referenced objects from database.
The method is used as `delAscData` method with one-to-many associations when updating relationship.

See Also: `addReferencedObjects` `updateReferencedObjects` `update`

getAllKeys(*self*)

Return tuple of application object's and referenced object's primary key values.
Referenced object is taken from referenced class broker.

Overrides: bazaar.assoc.List.getAllKeys

reloadData(*self*, *now*=`False`)

Request reloading association relational data. Referenced objects are reloaded, too.
Association data are removed from memory. If `now` is set to true, then relationship data are loaded from database immediately.

Parameters

`now`:	Reload relationship data immediately.

Overrides: bazaar.assoc.List.reloadData

updateableAscData(*self*, *obj*, *value*)

Return pair of application object and referenced object.
The data will be used to update association in database.

Overrides: bazaar.assoc.List.updateableAscData
See Also: `update`

updateReferencedObjects(*self*, *pairs*)

Update referenced objects in database.
The method is used as `addAscData` and as `delAscData` with one-to-many associations when updating relationship.

See Also: `addReferencedObjects` `delReferencedObjects` `update`

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__
Inherited from AssociationReferenceProxy: save
Inherited from BiDirList: integrateRemove, integrateSave, remove
Inherited from List: __get__, __set__, addAscData, appendKey, contains, delAscData, iterObjects, justRemove, len, loadData, saveForeignKey, update

2.9.2 Instance Variables

Name	Description
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)*	
Inherited from List: appended *(p. 15)*, cache *(p. 15)*, reload *(p. 15)*, removed *(p. 15)*	

2.10 Class OneToOne

```
__builtin__.object ─┐
                    │
bazaar.assoc.AssociationReferenceProxy ─┐
                                        │
                                    OneToOne
```

Known Subclasses: BiDirOneToOne

Class for uni-directional one-to-one association descriptors.

See Also: `bazaar.assoc.AssociationReferenceProxy` `bazaar.assoc.BiDirOneToOne`

2.10.1 Methods

__get__(*self*, *obj*, *cls*)

Descriptor interface method to retrieve reference of referenced object for application object.

Parameters

`obj`: Application object.
`cls`: Application class.

Return Value

Referenced object when `obj` is not null, otherwise descriptor object.

__set__(*self*, *obj*, *value*)

Descriptor interface method to set application object's attribute and foreign key values.
This method is optimized for uni-directional one-to-one association.

Parameters

`obj`: Application object.
`value`: Referenced object.

saveForeignKey(*self*, *obj*, *vkey*)

Save referenced object's primary key value.
Application object's foreign key value is set to referenced object's primary key value.

Parameters

`obj`: Application object.
`vkey`: Referenced object primary key value.

Overrides: bazaar.assoc.AssociationReferenceProxy.saveForeignKey

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__
Inherited from AssociationReferenceProxy: __init__, save

2.10.2 Instance Variables

Name	Description
Inherited from AssociationReferenceProxy: association *(p. 10)*, broker *(p. 10)*, col *(p. 10)*, vbroker *(p. 10)*	

3 Module bazaar.cache

Cache and reference buffer classes.

Cache classes are used to buffer objects and association data loaded from database. There are two types of cache:

- full:
  - objects - all objects of their class are loaded from database at once
  - association data - all data (for all application objects of given relationship between two classes) are loaded from database at once
- lazy:
  - objects - only one object is loaded from database
  - association data - data are loaded for given application object

Cache and buffer classes are dictionaries. A dictionary contains pairs of primary key value and object identified by the primary key (object cache) or application object and set of primary key values of referenced objects (one-to-many and many-to-many association data cache).

Reference buffers contains objects, which does not have priamry key values (are not in database).

Every class and association has its own cache, which is configurable, see `bazaar.config` module documentation.

3.1 Variables

Name	Description
log	**Value:** `<logging.Logger instance at 0x403f406c>`

3.2 Class Cache

Known Subclasses: Full, Lazy

Abstract, basic class for different data caches.

3.2.1 Methods

__init__(*self*, *owner*)

Create cache object.

Parameters

`owner:` Owner of the cache - object broker or association object.

__getitem__(*self*, *param*)

Return referenced object or association data.

Parameters

`param:` Referenced object primary key value or application object.

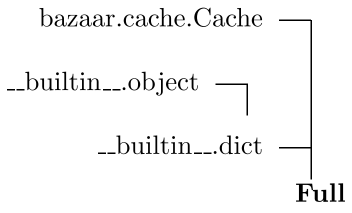
load(*self*, *key*)

Load referenced objects or association data from database.

3.2.2 Instance Variables

Name	Description
owner	Owner of the cache - object broker or association object.

3.3 Class Full



Known Subclasses: FullAssociation, FullObject

Abstract, basic cache class for loading all objects and association data.

See Also: `getEmpty` `bazaar.cache.FullObject` `bazaar.cache.FullAssociation`

3.3.1 Methods

__init__(*self*, *param*)
Overrides: bazaar.cache.Cache.__init__

__getitem__(*self*, *param*)

Return referenced object or association data.

Parameters
`param`: Referenced object primary key value or application object.

Return Value
Referenced object or association data (depends on cache type). If data is not found then `None` or empty set is returned.

Overrides: bazaar.cache.Cache.__getitem__
See Also: `bazaar.cache.FullObject` `bazaar.cache.FullAssociation`

getEmpty(*self*)

Get data which will be returned when object or association data will not be found in the cache.

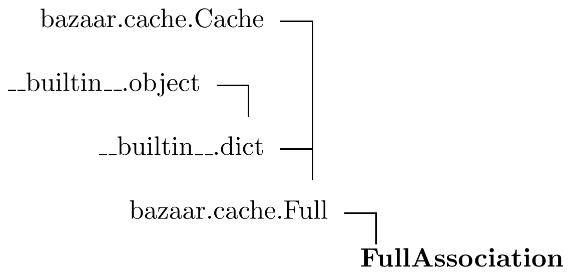
Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__

Inherited from type: __new__, fromkeys
Inherited from Cache: load

3.3.2 Instance Variables

Name	Description
empty	Is returned by `__getitem__` method when referenced object or association data are not found in cache.
Inherited from Cache: owner *(p. 24)*	

3.4 Class FullAssociation



Cache for loading all association data of relationship from database.

3.4.1 Methods

getEmpty(*self*)

Get data which will be returned when object or association data will not be found in the cache.
This method returns empty set.

Overrides: bazaar.cache.Full.getEmpty

load(*self*, *obj*)

Load all association data from database.

Overrides: bazaar.cache.Cache.load
See Also: `bazaar.assoc.List.loadData`

Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: __new__, fromkeys
Inherited from Full: __init__, __getitem__

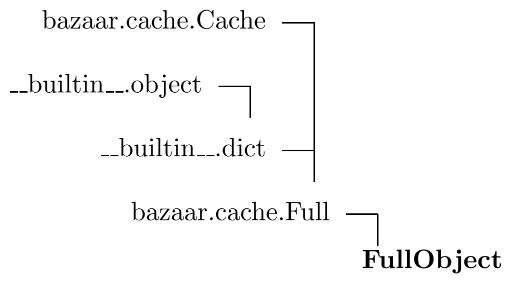
3.4.2 Instance Variables

Name	Description
Inherited from Cache: owner *(p. 24)*	

continued on next page

Name	Description
Inherited from Full: empty *(p. 25)*	

3.5 Class FullObject



Cache class for loading all objects of application class from database.

3.5.1 Methods

getEmpty(*self*)

Get data which will be returned when object or association data will not be found in the cache.
This method returns `None`.

Overrides: bazaar.cache.Full.getEmpty

load(*self*, *key*)

Load all application class objects from database.

Overrides: bazaar.cache.Cache.load
See Also: `bazaar.core.Broker.loadObjects`

Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: __new__, fromkeys
Inherited from Full: __init__, __getitem__

3.5.2 Instance Variables

Name	Description
Inherited from Cache: owner *(p. 24)*	
Inherited from Full: empty *(p. 25)*	

3.6 Class Lazy

bazaar.cache.Cache
Lazy

Known Subclasses: LazyAssociation, LazyObject

Abstract, basic cache class for lazy objects and association data loading.

3.6.1 Methods

__getitem__(*self*, *param*)

Return referenced object or association data.

Parameters

`param`: Referenced object primary key value or application object.

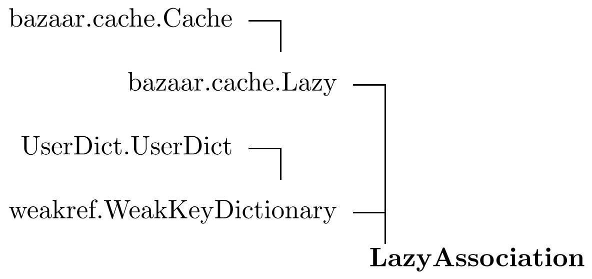
Overrides: bazaar.cache.Cache.__getitem__

Inherited from Cache: __init__, load

3.6.2 Instance Variables

Name	Description
dicttype	Weak dictionary superclass, i.e. `WeakValueDictionary` or `WeakKeyDictionary`.
Inherited from Cache: owner *(p. 24)*	

3.7 Class LazyAssociation



Cache for lazy loading of association data from database.

3.7.1 Methods

__init__(*self*, *owner*)

Create object lazy cache.

Parameters

`owner`: Owner of the cache - object broker or association object.

Overrides: bazaar.cache.Cache.__init__

load(*self*, *obj*)

Load association data from database for application object `obj`.

Parameters

`obj`: Application object.

Return Value

Loaded association data from database.

Overrides: bazaar.cache.Cache.load

Inherited from UserDict: __cmp__, __len__, clear, values
Inherited from Lazy: __getitem__
Inherited from WeakKeyDictionary: __contains__, __delitem__, __iter__, __repr__, __setitem__, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update

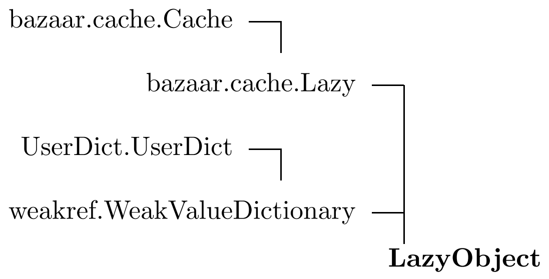
3.7.2 Class Methods

Inherited from UserDict: fromkeys

3.7.3 Instance Variables

Name	Description
Inherited from Cache: owner *(p. 24)*	
Inherited from Lazy: dicttype *(p. 27)*	

3.8 Class LazyObject



Cache for lazy referenced object loading.

3.8.1 Methods

__init__(*self*, *owner*)

Create object lazy cache.

Parameters

`owner`: Owner of the cache - object broker or association object.

Overrides: bazaar.cache.Cache.__init__

load(*self, key*)

Load referenced object with primary key value `key`.

Overrides: bazaar.cache.Cache.load

Inherited from UserDict: __cmp__, __contains__, __delitem__, __len__, clear, has_key, keys
Inherited from Lazy: __getitem__
Inherited from WeakValueDictionary: __iter__, __repr__, __setitem__, copy, get, items, iteritems, iterkeys, itervalues, pop, popitem, setdefault, update, values

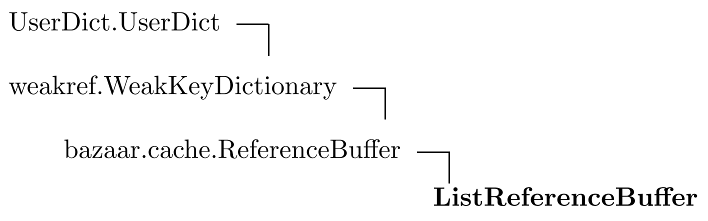
3.8.2 Class Methods

Inherited from UserDict: fromkeys

3.8.3 Instance Variables

Name	Description
Inherited from Cache: owner *(p. 24)*	
Inherited from Lazy: dicttype *(p. 27)*	

3.9 Class ListReferenceBuffer



Reference buffer for set of objects.

It is dictionary with application objects as keys and set of referenced objects as value.

See Also: `bazaar.cache.ReferenceBuffer`

3.9.1 Methods

__contains__(*self*, *item*)

Check if object is in reference buffer. Operator `in` can be used in two ways:

```
# buffer contains minimum one referenced value by application
# object obj (len(ref_buf[obj]) > 0):
obj in ref_buf
# or
(obj, None) in ref_buf

# referenced object value is referenced by obj and exists in
# buffer:
(obj, value) in ref_buf
```

Parameters

`item`: Application object or pair of application object and referenced object.

Overrides: bazaar.cache.ReferenceBuffer.__contains__

__delitem__(*self*, (*obj*, *value*))

Remove referenced object from application object's set of referenced objects.
If the set contains no more referenced objects, it is removed from dictionary.

Parameters

`obj`: Application object.
`value`: Referenced object.

Overrides: bazaar.cache.ReferenceBuffer.__delitem__

__setitem__(*self*, *obj*, *value*)

Add referenced object to the aplication object's set of referenced objects.
The set is created if it does not exist.

Parameters

`obj`: Application object.
`value`: Referenced object.

Overrides: weakref.WeakKeyDictionary.__setitem__

Inherited from UserDict: __cmp__, __len__, clear, values
Inherited from WeakKeyDictionary: __init__, __getitem__, __iter__, __repr__, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update

3.9.2 Class Methods

Inherited from UserDict: fromkeys

3.10 Class ReferenceBuffer

```
UserDict.UserDict ─┐
                   │
weakref.WeakKeyDictionary ─┐
                           │
                      ReferenceBuffer
```

Known Subclasses: ListReferenceBuffer

Simple reference buffer class.

The class is used to save referenced objects, which has no primary key value.

It is dictionary with application objects as keys and referenced objects as values.

See Also: `bazaar.cache.ListReferenceBuffer`

3.10.1 Methods

__contains__(*self*, *item*)

Check if application object is stored in reference buffer.

Parameters

`item`: Tuple of application object and referenced object.

Return Value

Returns true if application object is in reference buffer.

Overrides: weakref.WeakKeyDictionary.__contains__

__delitem__(*self*, (*obj*, *value*))

Remove application object from reference buffer.

Overrides: weakref.WeakKeyDictionary.__delitem__

Inherited from UserDict: __cmp__, __len__, clear, values
Inherited from WeakKeyDictionary: __init__, __getitem__, __iter__, __repr__, __setitem__, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update

3.10.2 Class Methods

Inherited from UserDict: fromkeys

4 Module bazaar.conf

Provides classes for mapping application classes to database relations.

Application class can be defined by standard Python class definition:

```
import bazaar.conf

class Order(bazaar.core.PersistentObject):
    __metaclass__ = bazaar.conf.Persistence
    relation      = ’order’
    columns       = {
        ’no’        : bazaar.conf.Column(’no’),
        ’finished’  : bazaar.conf.Column(’finished’),
        ’birthdate’ : bazaar.conf.Column(’birthdate’),
    }
```

It is possible to create application class by class instantiation:

```
Order = bazaar.conf.Persistence(’order’)
Order.addColumn(’no’)
Order.addColumn(’finished’)
```

Of course, both ideas can be mixed:

```
class Order(bazaar.core.PersistentObject):
    __metaclass__ = bazaar.conf.Persistence
    relation      = ’order’

Order.addColumn(’no’)
Order.addColumn(’finished’)
```

(section) Associations

Method `bazaar.conf.Persistence.addColumn` makes possible to define associations between classes (see `bazaar.assoc` module documentation for implementation details).

(section) One-to-one association

To define one-to-one association between two classes, programmer should specify the application class attribute, relation column and referenced class. For example, to associate department class with its boss (uni-directional relationship):

```
Department.addColumn(’boss’, ’boss_fkey’, Boss)
```

In case of bi-directional association (where boss is aware of department and vice versa):

```
Department.addColumn(’boss’, ’boss_fkey’, Boss, vattr = ’department’)
Boss.addColumn(’department’, ’dep_fkey’, Department, vattr = ’boss’)
```

Defining bi-directional relationship involves specifing attribute (with `vattr` parameter) of opposite class which glues the whole association.

SQL schema of `Department` and `Boss` classes would look like:

```
create table boss (
    __key__      integer,
    name          varchar(10) not null,
    surname       varchar(20) not null,
    phone        varchar(12) not null,
    dep_fkey     integer unique,
    unique (name, surname),
    primary key (__key__)
    --  see below
    --    foreign key (dep_fkey) references department(__key__) initially deferred
);


create sequence department_seq;
create table department (
    __key__      integer,
    boss_fkey    integer unique,
    primary key (__key__),
    foreign key (boss_fkey) references boss(__key__) initially deferred
);

alter table boss add foreign key (dep_fkey) references department(__key__) initially deferred;
```

(section) Many-to-many association

In relational database many-to-many relationships are created with one additional link relation. Therefore, when defining the association, programmer should specify following parameters:

- attribute name
- referenced application class
- link relation and its columns

For example, uni-directional many-to-many association between `Employee` and `Order` classes:

```
Employee.addColumn('orders', 'employee', Order, 'employee_orders', 'order')
```

SQL schema:

```
create sequence order_seq;
create table "order" (
    __key__      integer,
    no           integer not null unique,
    finished     boolean not null,
    primary key (__key__)
);

create sequence employee_seq;
create table employee (
    __key__      integer,
    name         varchar(10) not null,
    surname      varchar(20) not null,
    phone        varchar(12) not null,
    unique (name, surname),
    primary key (__key__)
);

create table employee_orders (
    employee         integer,
    "order"          integer,
    primary key (employee, "order"),
    foreign key (employee) references employee(__key__),
    foreign key ("order") references "order"(__key__)
);
```

To define bi-directional association attribute of opposite class, as in case of bi-directional one-to-one association, code should be written:

```
Employee.addColumn('orders', 'employee' Order, 'employee_orders', 'order', 'employees')
Order.addColumn('employees', 'order', Employee, 'employee_orders', 'employee', 'orders')
```

(section) One-to-many association

Following SQL schema describes two one-to-many associations:

```
create sequence article_seq;
create table article (
    __key__      integer,
    name          varchar(20) not null,
    price         numeric(10,2) not null,
    unique (name),
    primary key (__key__)
);

create sequence order_item_seq;
create table order_item (
    __key__      integer,
    order_fkey   integer,
    pos           integer not null,
    article_fkey integer not null,
    quantity      numeric(10,3) not null,
    primary key (__key__),
    unique (order_fkey, pos),
    foreign key (order_fkey) references "order"(__key__),
    foreign key (article_fkey) references article(__key__)
);
```

First one is uni-directional relationship between `Article` and `OrderItem` classes' relations - many order items can be created for one article. The relationship should be defined on "many" side with similar code as in case of uni-directional one-to-one association:

```
OrderItem.addColumn('article', 'article_fkey', Article)
```

There is second relationship. Bi-directional association between `Order` and `OrderItem` classes. The nature of this association due its realization excludes uni-directionality. It is because of `order_fkey` column of `order_item` relation. Definition of such association considers its bi-directionality:

```
Order.addColumn('items', vcls = OrderItem, vcol = 'order_fkey', vattr = 'order')
OrderItem.addColumn('order', 'order_fkey', Order, vattr = 'items')
```

(section) Inheritance

There are two classes defined above. `Boss` class is very similar to `Employee` class. The last one can be reused with inheritance:

```
Boss = bazaar.conf.Persistence('Boss', bases = (Employee,), relation = 'boss')
```

Boss class derives all attributes and associations from `Employee` class.

SQL schema for `Boss` class relation can look like:

```
create table boss (
    dep_fkey     integer,
    foreign key (dep_fkey) references department(__key__) initially deferred
) inherits(employee);
```

4.1 Variables

Name	Description
log	**Value:** `<logging.Logger instance at 0x4048066c>`

4.2 Class Column

Describes application class attribute.

Application class atribute can be simple attribute or can define association (relationship) between application classes.

When class attribute describes association the `vcls` is always defined. Depedning on relationship type (1-1, 1-n, m-n, uni-directional, bi-directional) some of the attributes `vlink`, `vcol`, `vattr` are defined, too.

See Also: bazaar.conf.Persistence.addColumn bazaar.assoc

4.2.1 Methods

__init__(*self*, *attr*, *col*=`None`)

Create application class attribute description.

Parameters

`attr`: Application class attribute name.
`col`: Relation column.

4.2.2 Properties

Name	Description
is_bidir	
is_many	
is_many_to_many	
is_one_to_many	
is_one_to_one	

4.2.3 Instance Variables

Name	Description
association	Association descriptor of given column.

continued on next page

Name	Description
attr	Application class attribute name.
col	Relation column name (equal to `attr` by default).
is_bidir	Class attribute describes bi-directional association.
is_many	Class attribute is one-to-many or many-to-many association.
is_many_to_many	Class attribute is many-to-many association.
is_one_to_many	Class attribute is one-to-many association.
is_one_to_one	Class attribute is one-to-one association.
link	Many-to-many link relation name.
update	Used with 1-n associations. If true, then update referenced objects on relationship update, otherwise add appended objects and delete removed objects.
vattr	Attribute name of referenced object(s).
vcls	Class of referenced object(s).
vcol	Relation column name of referenced object(s).

4.3 Class Persistence

```
__builtin__.object ─┐
                    │
        __builtin__.type ─┐
                          │
                     Persistence
```

Application class metaclass.

Programmer defines application classes with the metaclass. The class is assigned to the database relation. Class name is used as relation name, by default.

4.3.1 Methods

addColumn(*self*, *attr*, *col*=`None`, *vcls*=`None`, *link*=`None`, *vcol*=`None`, *vattr*=`None`, *update*=`True`)

Add attribute description to persistent application class.
This way the application class attributes and relationships between application classes are defined.

Parameters

`attr:`	Application class attribute name.
`col:`	Relation column name (equal to `attr` by default).
`vcls:`	Class of referenced object(s).
`link:`	Many-to-many link relation name.
`vcol:`	Relation column name of referenced object(s).
`vattr:`	Attribute name of referenced object(s).
`update:`	Used with 1-n associations. If true, then update referenced objects on relationship update, otherwise add appended objects and delete removed objects.

See Also: `bazaar.conf.Column`

getColumns(*self*)

Return list of all defined columns including inherited.

Inherited from object: __init__, __reduce__, __reduce_ex__, __str__
Inherited from type: __call__, __cmp__, __delattr__, __getattribute__, __hash__, __repr__, __setattr__, __subclasses__, mro

4.3.2 Static Methods

__new__(*self*, *name*, *bases*=`(<class 'bazaar.core.PersistentObject'>,)`, *data*=`None`, *relation*=`None`, *sequencer*=`None`, *modname*=`'bazaar.conf'`)

Create application class.

Parameters

`relation:`	Database relation name.
`sequencer:`	Name of primary key values generator sequencer.
`modname:`	Module of the application class, i.e. `app.business`.

Overrides: __builtin__.type.__new__

4.3.3 Instance Variables

Name	Description
columns	List of application class attribute descriptions.
relation	Database relation name.
sequencer	Name of primary key values generator sequencer.

4.3.4 Class Variables

Name	Description
Inherited from type: __bases__ *(p. ??)*, __basicsize__ *(p. ??)*, __dictoffset__ *(p. ??)*, __flags__ *(p. ??)*, __itemsize__ *(p. ??)*, __mro__ *(p. ??)*, __name__ *(p. ??)*, __weakrefoffset__ *(p. ??)*	

5 Module bazaar.config

Module contains basic classes for Bazaar layer configuration.

Bazaar layer is configurable. It is possible to specify several parameters in configuration file such as DB-API module, database connection string, cache classes, relations, etc.

All parameters are presented in table below:

Group	Section	Parameter	Default value
basic	bazaar	module	---
		dsn	---
		seqpattern	select nextval for %s
classes	bazaar.cls	<cls>.relation	application class name
		<cls>.sequencer	<cls>.relation + '_seq'
		<cls>.cache	bazaar.cache.FullObject
associations	bazaar.asc	<attr>.cache	bazaar.cache.FullAssociation

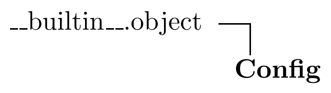
Sample configuration file using `bazaar.config.CPConfig` class:

```
[bazaar]
dsn:        dbname = ord port = 5433
module:     psycopg
seqpattern: select nextval('%s');

[bazaar.cls]
app.Article.sequencer: article_seq
app.Article.relation:  article
app.Article.cache:     bazaar.cache.FullObject
app.OrderItem.cache:   bazaar.cache.LazyObject

[bazaar.asc]
app.Department.boss.cache: bazaar.cache.FullAssociation
app.Order.items.cache:     bazaar.cache.LazyAssociation
```

It is possible to implement different configuration classes. This module contains abstract config class `bazaar.config.Config`. Class `bazaar.config.CPConfig` loads Bazaar configuration with `ConfigParser` class (ini files). Every configuration class method returns an option or `None` if specified parameter is not found in config source.

Of course, it is possible to implement other configuration classes, i.e. for GConf[7] configuration system.

5.1 Class Config



[7] http://www.gnome.org/projects/gconf/

Known Subclasses: CPConfig

Basic, abstract configuration class.

5.1.1 Methods

getAssociationCache(*self*, *attr*)

Get name of association cache.

Parameters

`attr:` Association attribute name, i.e. `Order.items`.

getClassRelation(*self*, *cls*)

Get name of application class' relation.

Parameters

`cls:` Class name.

getClassSequencer(*self*, *cls*)

Get name of sequencer used to get application objects primary key values.

Parameters

`cls:` Class name of application objects.

getDBModule(*self*)

Return Python DB API module.

getDSN(*self*)

Return Python DB API data source name.

getObjectCache(*self*, *cls*)

Get name of application objects cache class.

Parameters

`cls:` Class name of application objects.

getSeqPattern(*self*)

Return pattern of SQL query, which is used to get next value of application object's primary key value, i.e. `select nextval('%s')`, where `%s` means name of sequencer.

Inherited from object: __init__, __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__

5.2 Class CPConfig

```
__builtin__.object ─┐
                    │
  bazaar.config.Config ─┐
                        │
                     CPConfig
```

Bazaar configuration using `ConfigParser` module.

5.2.1 Methods

__init__(*self*, *cfg*)

Create instance of configuration.

Parameters

`cfg`: `ConfigParser` object.

Overrides: __builtin__.object.__init__

getAssociationCache(*self*, *attr*)

Get name of association cache.

Parameters

`attr`: Association attribute name, i.e. `Order.items`.

Overrides: bazaar.config.Config.getAssociationCache

getClassRelation(*self*, *cls*)

Get name of application class' relation.

Parameters

`cls`: Class name.

Overrides: bazaar.config.Config.getClassRelation

getClassSequencer(*self*, *cls*)

Get name of sequencer used to get application objects primary key values.

Parameters

`cls`: Class name of application objects.

Overrides: bazaar.config.Config.getClassSequencer

getDBModule(*self*)

Return Python DB API module.

Overrides: bazaar.config.Config.getDBModule

getDSN(*self*)

Return Python DB API data source name.

Overrides: bazaar.config.Config.getDSN

getObjectCache(*self*, *cls*)

Get name of application objects cache class.

Parameters

`cls`: Class name of application objects.

Overrides: bazaar.config.Config.getObjectCache

getSeqPattern(*self*)

Return pattern of SQL query, which is used to get next value of application object's primary key value, i.e. `select nextval('%s')`, where `%s` means name of sequencer.

Overrides: bazaar.config.Config.getSeqPattern

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__

5.2.2 Instance Variables

Name	Description
cfg	`ConfigParser` object.

6 Module bazaar.core

This module contains basic Bazaar implementation.

Every application object should derive from `PersistentObject` class.

Class `Bazaar` is designed to to get, modify, find and perform other operations on objects of any application class.

Brokers (`Broker` class) are responsible for operations on objects of specific application class.

6.1 Variables

Name	Description
log	**Value:** `<logging.Logger instance at 0x4048020c>`

6.2 Class Bazaar

The interface to get, modify, find and perform other tasks on application objects.

See Also: `Broker` `bazaar.motor.Motor`

6.2.1 Methods

__init__(*self*, *cls_list*, *config*=`None`, *dsn*=`''`, *dbmod*=`None`, *seqpattern*=`None`)

Start the Bazaar layer.
If database source name is not empty, then database connection is created.

Parameters

`cls\_list:`	List of application classes.
`config:`	Configuration object.
`dsn:`	Database source name.
`dbmod:`	Python DB API module.
`seqpattern:`	Sequence command pattern.

See Also: `bazaar.core.Bazaar.connectDB`, `bazaar.config`

add(*self*, *obj*)

Add object to database.

Parameters

`obj:` Object to add.

closeDBConn(*self*)

Close database connection.

See Also: `bazaar.core.Bazaar.connectDB`

commit(*self*)

Commit pending database transactions.

connectDB(*self*, *dsn*=`None`)

Make new database connection.
New database connection is created with specified database source name, which must conform to Python DB API Specification, i.e.:

```
bazaar.connectDB('dbname=addressbook host=localhost port=5432 user=bird')
```

It is possible to reconnect with previous database source name, too:

```
bazaar.connectDB()
```

Parameters

`dsn`: Database source name.

See Also: `bazaar.core.Bazaar.closeDBConn`

delete(*self*, *obj*)

Delete object from database.
Object's primary key value is set to `None`.

Parameters

`obj`: Object to delete.

find(*self*, *cls*, *query*, *param*=None, *field*=0)

Find objects of given class in database.
The method can be used in two ways.
First, simple dictionary can be passed as query:

```
# iterator is returned, so its next() method is used to get
# the object
apple = bazaar.find(Article, {'name': 'apple'}).next()
```

Dictionary can contain objects as values, i.e.:

```
bazaar.find(OrderItem, {'article': art})
```

Second, it is possible to use full power of SQL language:

```
# find orders and their articles if order amount is greater
# than 50$

# find orders with the query
query = """
    select O.__key__ from "order" O
    left outer join order_item OI on O.__key__ = OI.order_fkey
    left outer join article A on OI.article_fkey = A.__key__
    group by O.__key__ having sum(A.price) > 50
"""

for ord in  bzr.find(Order, query):
    print ord                     # show order
    for oi in ord.items:          # show order's articles
        print oi.article
```

Parameters

`cls`:	Application class.
`query`:	SQL query or dictionary.
`param`:	SQL query parameters.
`field`:	SQL column number which describes found objects' primary key values.

Return Value

Iterator of found objects.

getObjects(*self*, *cls*)

Get list of application objects.
Only objects of specified class are returned.

Parameters

`cls`: Application class.

See Also: `bazaar.core.Bazaar.reloadObjects`

init(*self*)

Initialize the Bazaar layer.

parseConfig(*self*, *config*)

Parse Bazaar configuration.

Parameters

`config`: Bazaar configuration.

See Also: `setConfig` `bazaar.config.Config` `bazaar.config.CPConfig`

reloadObjects(*self*, *cls*, *now*=`False`)

Reload objects from database.

Parameters

`cls`: Application class.

`now`: Reload objects immediately.

See Also: `bazaar.core.Bazaar.getObjects`

rollback(*self*)

Rollback database transactions.

setConfig(*self*, *config*)

Set Bazaar configuration.

See Also: `parseConfig` `init` `bazaar.config.Config` `bazaar.config.CPConfig`

update(*self*, *obj*)

Update object in database.

Parameters

`obj`: Object to update.

6.2.2 Instance Variables

Name	Description
brokers	Dictionary of brokers. Brokers are mapped with its class application.
cls_list	List of application classes.
dbmod	Python DB API module.
dsn	Python DB API database source name.
motor	Database access object.

6.3 Class Broker

Application class broker.

Application class broker is responsible for taking decision on getting objects from database or cache, loading application objects from database with convertor and manipulating application objects with cache.

See Also: `bazaar.motor.Motor` `bazaar.motor.Convertor` `bazaar.cache`

6.3.1 Methods

__init__(*self*, *cls*, *mtr*, *seqpattern*=`None`)

Create application class broker.
Method initializes object cache and database data convertor. Database objects loading is requested.

Parameters

`cls:` Application class.
`mtr:` Database access object.

---

add(*self*, *obj*)

Add object into database.

Parameters

`obj:` Object to add.

---

delete(*self*, *obj*)

Delete object from database.
Object's primary key value is set to `None`.

Parameters

`obj:` Object to delete.

---

find(*self*, *query*, *param*=`None`, *field*=`0`)

Find objects in database.

Parameters

`query:` SQL query or dictionary.
`param:` SQL query parameters.
`field:` SQL column number which describes found objects' primary key values.

See Also: `bazaar.core.Bazaar.find`

---

get(*self*, *key*)

Get application object.
Object is returned from cache.

Parameters

`key:` Object's primary key value.

Return Value

Object with primary key value equal to `key`.

See Also: `bazaar.cache`

---

getObjects(*self*)

Get list of application objects.
If objects reload has been requested, then objects would be loaded from database before returning objects from the cache.

See Also: `bazaar.core.Broker.loadObjects` `bazaar.core.Broker.reloadObjects`

loadObjects(*self*)

Load application objects from database and put them into cache.

See Also: `bazaar.core.Broker.getObjects` `bazaar.core.Broker.reloadObjects`

reloadObjects(*self*, *now*=`False`)

Request reloading objects from database.
All objects are removed from cache. If `now` is set to true, then objects are loaded from database immediately.

Parameters

`now`: Reload objects immediately.

See Also: `bazaar.core.Broker.loadObjects` `bazaar.core.Broker.getObjects`

update(*self*, *obj*)

Update object in database.

Parameters

`obj`: Object to update.

6.3.2 Instance Variables

Name	Description
cache	Cache of application objects.
cls	Application class.
convertor	Relational and object data convertor.
reload	If true, then application object's reload has been requested.

6.4 Class PersistentObject

```
__builtin__.object ─┐
                    │
                    PersistentObject
```

Parent class of an application class.

6.4.1 Methods

__init__(*self*, ***data*)

Create persistent object with initial data.

Parameters

`data`: Initial values of object attributes.

Overrides: __builtin__.object.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from type: __new__

6.4.2 Instance Variables

Name	Description
__key__	Object's key.

7 Module bazaar.exc

Bazaar exceptions.

7.1 Class AssociationError

```
exceptions.Exception ─┐
                      │
     bazaar.exc.BazaarError ─┐
                             │
                      AssociationError
```

Association exception.

7.1.1 Methods

__init__(*self*, *msg*, *asc*, *obj*, *value*)

Create association exception.

Parameters

`msg:`	Exception message.
`asc:`	Association object.
`obj:`	Application object.
`value:`	Referenced object.

Overrides: exceptions.Exception.__init__

Inherited from Exception: __getitem__, __str__

7.1.2 Instance Variables

Name	Description
asc	Association object.
obj	Application object.
value	Referenced object.

7.2 Class BazaarError

```
exceptions.Exception ─┐
                      │
                 BazaarError
```

Known Subclasses: AssociationError, MappingError

Abstract, basic class for all Bazaar exceptions.

7.2.1 Methods

Inherited from Exception: __init__, __getitem__, __str__

7.3 Class ColumnMappingError

```
exceptions.Exception ─┐
    bazaar.exc.BazaarError ─┐
        bazaar.exc.MappingError ─┐
                                 ColumnMappingError
```

Relation column mapping exception.

Exception is thrown on mapping relation column to application class attribute error, i.e. empty attribute name.

7.3.1 Methods

__init__(*self*, *msg*, *cls*, *col*)

Create relation column mapping exception.

Parameters

- `msg`: Exception message.
- `cls`: Application class.
- `col`: Application class column object.

Overrides: bazaar.exc.MappingError.__init__

Inherited from Exception: __getitem__, __str__

7.3.2 Instance Variables

Name	Description
col	Application class column object.
Inherited from MappingError: cls *(p. 51)*	

7.4 Class MappingError

```
exceptions.Exception ─┐
    bazaar.exc.BazaarError ─┐
                            MappingError
```

Known Subclasses: ColumnMappingError, RelationMappingError

Abstract, basic class for all mapping exceptions.

See Also: `bazaar.exc.ColumnMappingError` `bazaar.exc.RelationMappingError`

7.4.1 Methods

__init__(*self*, *msg*, *cls*)

Create mapping exception.

Parameters

`msg`: Exception message.
`cls`: Application class.

Overrides: exceptions.Exception.__init__

Inherited from Exception: __getitem__, __str__

7.4.2 Instance Variables

Name	Description
cls	Application class.

7.5 Class RelationMappingError

```
exceptions.Exception ─┐
                      │
  bazaar.exc.BazaarError ─┐
                          │
    bazaar.exc.MappingError ─┐
                             │
                   RelationMappingError
```

Database relation mapping exception.

Exception is thrown on mapping database relation to application class error, i.e. empty relation name.

7.5.1 Methods

Inherited from MappingError: __init__
Inherited from Exception: __getitem__, __str__

7.5.2 Instance Variables

Name	Description
Inherited from MappingError: cls *(p. 51)*	

8 Module bazaar.motor

Data convertor and database access classes.

8.1 Variables

Name	Description
log	**Value:** `<logging.Logger instance at 0x4047990c>`

8.2 Class Convertor

Relational and object data convertor.

The class creates all required SQL queries. It converts relational data to object oriented form and vice versa.

`Motor` class is used to connect and execute commands in database.

8.2.1 Methods

__init__(*self*, *cls*, *mtr*, *seqpattern*=`None`)

Create data convertor object.

Parameters

`cls`: Application class.
`mtr`: `Motor` class object.

add(*self*, *obj*)

Add object to database.

Parameters

`obj`: Object to add.

addAscData(*self*, *asc*, *pairs*)

Add association relational data into database.
Adding the data means adding data into link table of many to many association or updating appropriate column of one to many association.

Parameters

`asc`: Association descriptor object.
`pairs`: List of association data - pairs of primary and foreign key values.

createObject(*self*, *data*)

Create object from relational data.

Parameters
    `data`: Relational data.

Return Value
    Created object.

delAscData(*self*, *asc*, *pairs*)

Delete association relational data from database.
Deleting the data means removing data from link table of many to many association. In case of one to many association it means deleting rows of relation on ”many” side or updating appropriate column of one to many association to None value (it depends on relationship configuration).

Parameters
    `asc`: Association descriptor object.
    `pairs`: List of association data - pairs of primary and foreign key values.

delete(*self*, *obj*)

Delete object from database.

Parameters
    `obj`: Object to delete.

dictToSQL(*self*, *param*)

Convert dictionary into `WHERE` SQL clause.
All dictionary items are glued with `AND` operator.

See Also: `bazaar.core.Bazaar.find`

find(*self*, *query*, *param*=`None`, *field*=`0`)

Find objects in database.

Parameters
    `query`: SQL query or dictionary.
    `param`: SQL query parameters.
    `field`: SQL column number which describes found objects’ primary key values.

See Also: `bazaar.core.Bazaar.find`

get(*self*, *key*)

Load object from database.

Parameters
    `key`: Primary key value of object to load.

getAllAscData(*self*, *asc*)

Get all association data from database.

Parameters

`asc`: Association object.

getAscData(*self*, *asc*, *obj*)

Get association relational data for the application object.

Parameters

`asc`: Association object.
`obj`: Application object.

getData(*self*, *obj*)

Extract relational data from application object.

Parameters

`obj`: Application object.

Return Value

Dictionary of object's relational data.

getKey(*self*)

Create new primary key value with sequencer.

Return Value

New primary key value.

getObjects(*self*)

Load objects from database.

objToData(*self*, *param*)

Convert object oriented parameters to pure relational data.

See Also: `bazaar.core.Bazaar.find`

update(*self*, *obj*)

Update object in database.

Parameters

`obj`: Object to update.

8.2.2 Instance Variables

Name	Description
cls	Application class, which objects are converted.
columns	List of columns used with database queries.
motor	Database access object.

continued on next page

Name	Description
queries	Queries to modify data in database.

8.3 Class Motor

Database access object.

The class depends od database API module - Python DB-API 2.0 in this case.

8.3.1 Methods

__init__(*self*, *dbmod*)

Initialize database access object.

Parameters

`dbmod`: DB-API 2.0 module.

add(*self*, *query*, *data*)

Insert row into database relation.

Parameters

`query`: SQL query.
`data`: Row data to insert.

closeDBConn(*self*)

Close database connection.

See Also: `bazaar.motor.Motor.connectDB`

commit(*self*)

Commit pending database transactions.

connectDB(*self*, *dsn*)

Connect with database.

Parameters

`dsn`: Data source name.

See Also: `bazaar.motor.Motor.closeDBConn`

delete(*self*, *query*, *key*)

Delete row from database relation.

Parameters

`query`: SQL query.
`key`: Key of the row to delete.

executeMany(*self*, *query*, *data_list*)

Execute batch query with list of data parameters.

Parameters

`query:` Query to execute.
`data\_list:` List of query's data.

getData(*self*, *query*, *param*=`None`)

Get list of rows from database.
Method returns dictionary per database relation row. The dictionary keys are relation column names and dictionary values are column values of the relation row.

Parameters

`query:` Database SQL query.

rollback(*self*)

Rollback database transactions.

update(*self*, *query*, *data*, *key*)

Update row in database relation.

Parameters

`query:` SQL query.
`data:` Tuple of new values for the row.
`key:` Key of the row to update.

8.3.2 Instance Variables

Name	Description
conn	Python DB API connection object.
dbmod	Python DB API module.

Index