This page summarizes what has been done recently with ZODB Components and from September 2010 to February 2011, related to portal type classes, ZODB property sheets, and accessors generation.

Contents

Portal type class

What has been done?

- ERP5 objects are converted to portal type instances.
- Portal type classes are instances of a class inside the module "erp5.portal_type". For example, a Person object should now have the "erp5.portal_type.Person" type.
- The last part of the class path is invariably the portal type name. If you want a erp5.portal_type.Banana class to be created, you need to create a portal type (in Types Tool / portal.portal_types) named "Banana".

As a developer, what do I need to pay attention to?

You cannot create instances of documents directly:

```
container._setObject("some_id", MyUberDocumentClass("some_id"))
```

This use should be banned, and will break. Instead, you should use:

```
container.newContent(portal_type="...", id="....")
```

You can experiment to understand the difference: the first use creates a Document instance, the second used creates a portal type class. We only want portal type classes.

- All objects you create must have a portal type object defined in Types Tool. No exception. Tools should have a portal type as well.

What needs to be done?

- Migration: objects that have a classpath different than ERP5Type.Document.XXX are not all migrated. migrateToPortalTypeClass tries a simple, cheap heuristic to migrate at least the items just under the portal, and the contents of Types Tool, but it's not perfect.
- Candidates to non-migration are: items in ERP5Type.Core (Folder, aka Modules, RoleInformaton, ActionInformation, ...) items in top level of a product (ERP5Type.XMLMatrix ?) and generally speaking, all the "bastard" classes that did not belong to a clean Document/ subfolder in your Product.
- Small scripts can be written to recurse in a site and call Base.migrateToPortalTypeClass on objects. But this needs to be done carefully, maybe checking before that a portal type does exist for the object you're migrating.
- Note on auto-migration: be careful if implementing "migrate automatically" things. The checks that you perform on instance bootup must be light. The checks that you perform on ERP5Site.migrateToPortalTypeClass must be light as well.

Hacking advanced notes

- Note that Base_viewDict now prints along the class of the object, to allow easy debugging/checks
- Any doubt on a specific object? Checks its mro (method resolution order): pp context._``_class_``_.mro() in a pdb. The first class should be the portal_type class (erp5.portal_type.Person), the second should be the Document class (ERP5.Document.Person.Person)
- Putting a __setstate__ in the Base class to attempt migration of all objects, wherever they are, does not work. You might know that Zope enforces its very own method resolution order, and as such, a method in Base is not guaranteed to always overwrite the methods from Persistent.Persistent.

ZODB Property Sheets

What has been done?

- Creation of a new tool, Property Sheet Tool: portal.portal_property_sheets that holds property sheets
- This allows removing the old Property Sheets that we had on the filesystem, and putting them in the ZODB instead.

As a developer, what do I need to pay attention to?

- Create new Property sheets as subobjects of Property Sheet Tool.
- You have four portal types depending on the Property you want to define: Standard Property, Acquired Property, Category and Dynamic Category. The first three should have explicit uses, the last one is meant to allow assigning base categories to an object using a Tales expression to select categories.
- When installing a Business Template, or with a new instance, you might need to migrate your filesystem property sheets into ZODB objects. There is an action on the Property Sheet Tool for that.

What needs to be done?

- Validation of property sheets to avoid regenerating accessors on invalid/incomplete changes. A few notes:
  - we can/should define constraints on properties to define how much we have to fill to consider the property to be valid.
  - we could use validation workflow, but it needs exporting objects to bootstrap/portal_*.xml with a "validated" state so that new instances can directly load correct objects without thinking.
  - another way is to filter in interaction workflows: do not trigger accessor regeneration is the property is inconsistent. On the other hand, the solution is imperfect: save an incomplete property, restart your instance, and you're doomed.
- Better way to add new Properties. As the number of portal types/constraints will grow, the "Action" menu will grow too long and it will be annoying.

Hacking advanced notes

- The structure of property sheet tool itself is not complicated. But accessor generation with property sheets is generally tricky, as it requires bootstrapping an instance from nothing, knowing that we need to fetch most of our information from ZODB. Chicken and eggs everywhere.
- With our through the web era, we introduce new issues, and change the way we're used to update our instances. We're used to somehow update products first, then zodb. But when editing "vital" content in the zodb, it seems that the point of focus for updates should be the zodb content first. Think about someone changing a portal type or a property sheet, and then changing the underlying document code: you need to update the ZODB/business templates first; but

to do so we need to have good enough products that allow restarting an instance in an "old" state. We have no mechanism whatsoever to provide this in ERP5; this is important to think about it or we (you) will run into big troubles in the future. Upgrader approach is good for projects, but less than ideal for developers, as we cannot write upgraders for each revisions, nor can we write everywhere heavy code to think about "what if developer updates like that"... Maybe portals need a kind of revision number attribute, allowing us to tell easily how old is the core part of an instance, on an extremely cheap manner: relying on template tool or any kind of complex introspection is too slow to be acceptable.

Constraints

What has been done?

- Constraints have always been part of Property Sheets. They now need to be defined in Property Sheet Tool, just as any other Property.
- But, if previously we had a classes in Constraint/ folder of products, we now need to define a Document (and a portal type) for each constraint. In Property Sheet view, you will see that several constraints can be added as a subobject: there is one Document, and one Portal Type for each constraint type.
- Constraints implement the Predicate API.

As a developer, what do I need to pay attention to?

- Projects need to rewrite their Constraint classes as Documents. See the difference between ERP5``Type.Constraint.Property``Existence and ERP5Type.Core.Property``Existence``Constraint for an idea of the simple changes that have to be done. You need to derive from constraint mixin, and override the _checkConsistency method
- Business Templates that have Constraint``Template``Item items should be rewritten, to move each Constraint``Template``Item as one Document``Template``Item (Constraint Document class) and one new Portal Type
- Parameters of fixConsistency(): filter={'id': ...} should be changed to filter={'reference': ...}.

Hacking advanced notes

- Performances of Predicate API for constraints needs to be assessed carefully.
- Since Constraints are now Predicates, they are indexed in predicate SQL table: some trunk tests had been failing because Predicate searchs were returning constraints instead of (rules, other predicates... your pick). Your project's tests might need to be refined.

ZODB Components

This only explains how to use ZODB Components from a developer point of view and what has changed since the presentation at Europython 2012.

What has been done?

Documents, Extensions and Tests in bt5 are no longer stored in instance home but directly into ZODB. This provides several benefits, such as:

- Changes are propagated in a single transaction on all the nodes, so when you install a bt5, everything is installed on all the clients at the same time, whereas it was required to copy the files around before.
- Edit through the web.

What I should know as a developer?

The general technical implementation of ZODB Components for ERP5 is documented in the slides written for Europython 2012. You can find the slides and notes there and the video there. Even though there has been some changes since then, the general idea still stands so this document is still worth reading.

Note that TYPE, used in this document, refers to ZODB Component type and may be currently equal to document, extension or test.

Here is what you should know when using ZODB Components:

- Unless a ZODB Component is in validated or modified validation_state, it will never be used (on filesystem, it would mean that the file does not exist at all).

  A Component is in modified state if it has been previously validated, but some errors were found after it has been saved. Until these errors have been fixed, reference, version and source code set when the Component was validated will be used.

Also, as ZODB Component are lazily loaded, no error may be displayed nor reported until you actually access the object.

- Only the reference and version matters to lookup for a Component to be loaded. Even though ID could be set to anything, it should only follow naming convention defined in the section below.
- Generally speaking, any ZODB Component can be imported like any Python module (but for most of them you should not do that except in tests, see the per-Components sections below for further information), for example:
  - Import Component of the version with the highest priority:

    ```
    import erp5.component.TYPE.Foo
    ```
  - Import specifically Component of version PROJECT:

    ```
    import erp5.component.TYPE.PROJECT_version.Foo
    ```
- For security reasons, a Developer Role has been introduced which is not available through the UI. Only users with Developer Role can modify ZODB Components. Similarly to Class Tool which requires creating a file in ERP5Type Product, you must edit zope.conf on the filesystem to add users to Developer Role. For instance, to add zope users to Developer Role:

  ```
  %import Products.ERP5Type
  <ERP5Type erp5>
    developers zope
  </ERP5Type>
  ```
- Upon export on the filesystem, a Component is split up into 2 files: metadata (xml) and source code (.py).
- Likewise ZODB Property Sheets and Portal Type as Classes, when a ZODB Component is modified, this reset Components and Portal Type as Classes as well as inheritance may have changed or Property Sheets.
- By default, textarea is used to edit source code through the web browser, but Ace Editor (provided in erp5_ace_editor) provides a much better UI along with maximize and fullscreen modes, jumping to line and column where an error or warning has been found... After installing erp5_ace_editor bt5, you can enable it as Source Code Editor in User Interface tab in Site/User Preference. Later on, Ace Editor will also be useable from ZMI (Jérôme).
- You can also edit ZODB Components through WebDAV or FTP (actually supported by Zope directly with a few code to make it work for ZODB Components). You must modify zope.conf and add the following section:

  ```
  <webdav-source-server>
     address IP:PORT
     force-connection-close off
  </webdav-source-server>
  ```

  For example to mount your ERP5 instance and edit ZODB Components with davfs2 (on Debian, the package is davfs2, read mount.davfs(8), umount.davfs(8) and davfs2.conf(5)):

  ```
  mount -t davfs -o uid=UID,gid=GID,username=USERNAME http://IP:PORT/erp5/portal_components /MOUNT/DIRECTORY
  ```

ID Naming convention

Even though ID could be anything, it should be in the following format:

TYPE.VERSION.REFERENCE

Migrating bt5 Documents, Extensions and Tests from filesystem actually follows this naming convention.

Adding custom version

For projects, you should add at least one specific version, which can be achieved by the following steps:

1. Add a version and its priority in Portal Properties, for example:project | 60.0
2. Add this version to Registered Version Priority Selection field in Business Template view.

For ERP5 Components, there is already erp5 version defined in erp5_core, so you don``t need to add anything.

Migration of existing bt5 Documents, Extensions, Tests and Products

Except for ERP5-specific version, you should create a new version, see previous section for that.

You can migrate Business Template thanks to Migrate Components from Filesystem action in Business Template view. In the next screen, you can specify versions of Components to be migrated.

Note that the migration is all or nothing and ZODB Components will not be automatically validated.

Also, Products in bt5 are deprecated, instead you must either migrate your Products to Documents or move them to normal Products, through your SlapOS Software Release recipe.

Extension Component (erp5.component.extension)

When adding an External Method, you can specify Module Name exactly as you used to do.

For example, a ZODB Extension Component whose version is project, reference is Bar and ID is extension.project.Bar, you must only specify Bar. Unless you have an Extension Component with the same reference and whose version has an higher priority, then it will be used automatically. From an implementation point of view, this will actually import erp5.component.extension.Bar, equivalent to erp5.component.extension.HIGHEST_PRIORITY_VERSION_version.Bar.

By default, when specifying Bar as Module Name, ZODB Components will be lookup and if there is no such Components, then it will fallback on the filesystem.

Document Component (erp5.component.document)

Likewise filesystem bt5 Document, ZODB Document Components in bt5 must **only** be used as Portal Types Type Class. But if you use these documents in tests for example, you must use erp5.component.document instead of erp5.document.

Test Component (erp5.component.test)

Basically, a Test Component behaves like a Document Component. However, as a Test Component is within a bt5, there is a chicken & egg issue with runUnitTest command for installation of bt5 dependencies because in current Unit Test, the list of required bt5s is defined in getBusinessTemplateList() class method which requires to load the Component. However, it cannot be loaded until the bt5 (and its dependencies) have been installed as it may depend on Document or other Test Components.

One solution would have been to fiddle with sys.path and implement workaround to load the Component without installing any bt5, but that would be hackish and would not work when trying to import Document Components.

Therefore, the solution implemented is to specify through runUnitTest command line the bt5 where the test can be found. This bt5 will be installed as well as its dependencies using bt5list file (so you *must* make sure that this file is up-to-date before doing running any test).

Migration steps:

1. This should already be the case but all the bt5 dependencies must be properly defined (dependency_list Business Template property or Dependencies field on Business Template view).

2. For bt5s required specifically to run tests, there is a new property, test_dependency_list (Test Dependencies on Business Template view) where they can be added. Please note that in contrary to filesystem test, the bt5 are not *forced* installed so you must define all dependencies, including solving virtual dependencies (for example, for erp5_full_text_catalog, you can add erp5_full_text_myisam_catalog to Test Dependencies).

3. For customer project, make sure that your SlapOS recipe generates bt5list for your customer bt5s. Also, to your customer tests/__init__.py, add the following path to your tests path:

   ```
   %s/bt5/*/TestTemplateItem/portal_components/test.*.test*.py
   ```

Finally, to execute a Test Component as a Live Tests, you can do through Run Live Tests Component Tool Action. As of runUnitTest command considering that testFoo is in bt5 called hogehoge:

```
runUnitTest hogehoge:testFoo
```

Procedure to upgrade customer project

1. With erp5.git before merge request 1032 (2b8c630500f8a65566cf5ccf76b5215add840e54):
   1. Replace deprecated newTempXXX calls as done in 26e3c68b10be9165318dec9c02184dca0398d3e4.
   2. Migrate all customer Products to ZODB Components to their appropriate bt5s using Migrate Components from Filesystem Business Template Action. This will automatically select classes used in the current Business Template Portal Types and will not delete anything from the filesystem once done. Also, this will fix imports only for the migrated files..
   3. Commit.

1. With current erp5.git:
   1. Delete .pyc files to make sure now deleted Documents are not loaded: git clean -xdf product/
   2. Fix imports (name of the module before migration from the FS is in source_reference property, you can use

https://lab.nexedi.com/nexedi/erp5/uploads/cef2ce5429e7abb9c9d0ab53b75b6593/fix_imports script for now: ./fix_imports /path/to/erp5/repository/ /path/to/customer/repository/
3. Commit.
4. Regenerate bt5list: ./product/ERP5/bin/genbt5list bt5 product/ERP5/bootstrap
5. Update all bt5s.
6. Filesystem Products can now be deleted from FS. Also make sure to remove pyc files: git clean -xdf -e bt5list product/.
7. Commit.

Major changes since the presentation at Europython 2012

This section lists major changes, excluding bootstrap issue, minor bug fixes and UI improvements here and there.

Security

Access to Component Tool has been further restricted (anyone was able to view Components) and is now set through Component Tool class rather than instance, so it can be easily changed anytime, rather than only being set at creation or through an upgrade script.

Pylint

Before, in order to check that the source code was somewhat valid, the code was actually executing, but this approach has the following drawbacks:

- Executing the source code may have side-effect, such as importing module or for monkey-patches.
- Does not detect error in code executed later (function...).
- Only the first error was reported.

The source code is now checked *statically* through Pylint if it can be imported, otherwise it fallbacks on executing the source code as before (please note that Pylint has been added to SlapOS recipe specifically for ZODB Components so you may need to update your environment).

Pylint has been chosen in favor of (faster) other implementation such as pyflakes because it can also check coding style and naming conventions, which will be used in the future. Moreover, it seems to report errors that pyflakes could not find.

As a side note, edition of ZODB Components through Ace Editor has been greatly improved so you can click directly on the errors or warning and it will go the corresponding line and column.

Import lock deadlock

Upon any import in Python < 3.3, the Python global import lock is acquired (to avoid race conditions while checking sys.path, avoid incomplete modules from being seen by other threads or processes and also to avoid a module from being executed twice). Therefore, ZODB Component import hooks (following PEP302) are protected by import lock.

However, there was a deadlock when trying to import Components, as these import hooks tries to fetch properties from ZODB, which may unpickle objects in another thread (for an Exception class with ZEO for example) and thus trying to acquire import lock when importing classes.

From now on, the import lock is released in import hooks until sys.path is actually modified and there is another lock (aq_method_lock, common to Portal Type as Classes and ZODB Property Sheets) to prevent entering import hooks in parallel.

A solution would be to introduce a per Component package lock (still coarse grain) or a per Component lock (finest grain we could do) if the performances end up being too bad but it seems to be working well enough as it is.

The best solution would be to use Python 3.3, as there is no more global lock but a lock per module. However, the import machinery has completely changed (implemented in Python and not C anymore), so it would be probably quite difficult to backport...

Export of Workflow State

A Component was automatically validated on Business Template installation, but this meant, among other things, that exchanging bt5 containing Components with errors was not possible.

The *last* Workflow History of Component Validation Workflow is now exported without adding anything, thanks to a new Property introduced in Business Template (before, you could only export the full Workflow History).

What need to be done or could be implemented later on?

Do not propagate changes on other nodes

As requested by a customer, this could be fairly useful to be able to change ZODB Components on one specific node before the changes are actually propagated on other nodes (for example, when fixing a bug on production to allow testing on only one node).

Instropection

Being able to know where a given ZODB Component comes from and where it is currently used (which Portal Type classes, which Property Sheets and so on). This should be common to ZODB Property Sheets and Portal Type as Classes.

Migration of Products

Once bugs found with bt5 ZODB Components and other bugs requiring restart of ERP5 have been fixed, the next milestone is to migrate filesystem Products to ZODB.

Partial reset

For now and likewise ZODB Property Sheets and Portal Type as Classes, everytime a ZODB Component is modified, a reset of *all* ZODB Components is done, but after implementing nicely introspection, it should be possible to only reset the modified Components and its dependencies.

[DONE] Remove ClassTool

ClassTool is no longer necessary and could probably be removed after the merge.

Pylint without intermediate file on the filesystem

Currently, an intermediate file on the filesystem is used to perform static checking, but this should not be necessary. Therefore, pylint should be patched so that it can take a string instead of a filename.

Accessor generation

What has been done?

We're getting rid of Base._aq_dynamic, and accessors are now generated directly from Property Sheet definitions, and put into Accessor Holder: one Accessor Holder for each existing Property Sheet item.

Accessor Holder are classes, and you can see them in the method resolution order of your ERP5 objects. For instance, for a person, person._``_class_``_.mro() is:

```
(<class 'erp5.portal_type.Person'>,
 <class 'Products.ERP5.Document.Person.Person'>,
 <class Products.ERP5.mixin.encrypted_password.EncryptedPasswordMixin at 0xcce42cc>,
 <class 'Products.ERP5Type.XMLObject.XMLObject'>,
 <class 'Products.ERP5Type.Core.Folder.Folder'>,
 <class Products.ERP5Type.CopySupport.CopyContainer at 0xb340b0c>,
 <class 'Products.CMFCore.CMFBTreeFolder.CMFBTreeFolder'>,
 <class 'Products.BTreeFolder2.BTreeFolder2.BTreeFolder2Base'>,
 [...]
 <class 'erp5.accessor_holder.BaseAccessorHolder'>,
 <class 'erp5.accessor_holder.DublinCore'>,
 <class 'erp5.accessor_holder.Task'>,
 <class 'erp5.accessor_holder.Reference'>,
 <class 'erp5.accessor_holder.Person'>,
 <class 'erp5.accessor_holder.DefaultImage'>,
 <class 'erp5.accessor_holder.Mapping'>,
 <class 'erp5.accessor_holder.CategoryCore'>,
 <class 'erp5.accessor_holder.Base'>,
 <class 'erp5.accessor_holder.Login'>,
 <class 'erp5.accessor_holder.XMLObject'>,
 <class 'erp5.accessor_holder.Folder'>,
 <class 'erp5.accessor_holder.SimpleItem'>,
 <type 'ExtensionClass.Base'>,
 <type 'object'>)
```

Each accessor_holder class corresponds to accessors that come directly from a Property Sheet. Note as well accessor_holder.BaseAccessorHolder which contains common methods such as related category getters and portal type group getters.

_aq_reset is gone as well.

As a developer, what do I need to pay attention to?

- mostly related to test writing: _aq_reset calls were replaced by compatible resetDynamicDocuments calls. They are compatible, and do work, but are slow. It's good for performance if you can check wether or not any of those _aq_reset can be replaced by resetDynamicDocumentsAtTransactionBoundary that delay the reset at the end of the transaction. See the section in hacker notes for details.

What needs to be done?

- cleanups, cleanups, cleanups. Base.py contains a lot of unused code.
- Documentation``Helper code is probably quite broken (has always been?) even if tests do not reflect that
- Two XML files are used to install sites, in ERP5/bootstrap. (portal_types and portal_property_sheets). There is no easy way to export those or regenerate them from an user point of view. What I did was writing a simple test fixture, save a new fresh site, editing/adding the new portal types, and exporting the XML. The problem is that we probably do not want to export ALL the content of the tools, but only a restricted set of portal types/property sheets.

Hacking advanced notes

Performances

The effective tradeoff of this change is the following: we trade dynamic lazy generation for static generation plus a few mro()-deep lookups. Check for example the Base._edit code, where we have to lookup in a class mro() to fetch the list of restricted methods. This kind of places where we have to walk one's mro() are costly places. On the other hand, it's EASY to optimize them. With lazy aq_dynamic, environment was constantly changing, and we had no guarantees that everything was generated. But with portal type classes/accessor holders, nothing ever changes once the class has been generated once: at the end of loadClass() (ERP5.dynamic.lazy_class) nothing will ever happen to the class anymore, data is "static". So it means that all deep computations we do can be safely cached on the class object for later. Back to our _edit example, the list of method ids that are restricted can be, and should probably be computed once and stored on the portal type class

A new performance test can now be written. On a tiny instance, that only has erp5_core:

```
portal.portal_types.resetDynamicDocuments()
for property_sheet in portal.portal_property_sheets.contentValues():
  property_sheet.createAccessorHolder()
```

And time this loop.

The impact of accessor generation is now easy to measure and improve, instead of being a giant octopus with tentacles that unfold at every dynamic call.

Once you've looped over this list, you're mostly done, and the rest of the code only gathers useful accessor holders and puts them on the right classes. Cherry picking with workflow twists, as you still need to wrap accessors as WorkflowMethod on the portal type class. We may start with a relatively higher cost, but that's easier to improve, easier to profile, easier to optimize.

If then, you want to assess the cost of Workflow method generation, you can do something like:

```
portal.portal_types.resetDynamicDocuments()
for portal_type_id in portal.portal_types.objectIds():
  getattr(erp5.portal_type, portal_type_id).loadClass()
```

And once again, time it.

Memory Cost

Generally speaking, we generate things less blindly, and after cleanups the memory usage should drop to a lower figure than with aq_dynamic.

The globals in Utils are evil, and cache too much. I suppose that removing them or emptying them WILL save a lot of memory. Similarly, I'm questioning the validity and use of the workflow_method_registry attributes on portal type/property holder/accessor holder classes

resetDynamicDocuments vs resetDynamicDocumentsAtTransactionBoundary

There was something relatively bad in the way we were using _aq_reset. Scenario:

```
# some_portal_type
portal_type.edit(type_class="Foo", type_base_category_list=["source",])
```

This edition triggers two workflow triggers, one for the class change, and one for the base category change. Each trigger used to cause an _aq_reset call.

Generally speaking, if during one transaction we had N property changes on M different objects, we would trigger N*M times _aq_reset. That begs the question: is it absolutely compulsory to reset accessors immediately after one's action?

If we think about it, 100% of the actions that can trigger accessor regeneration are user-triggered. Meaning that transactions will be short-lived, and that in case of a success, a commit() will happen under a short time.

So can we delay the reset at commit time? Yes, it seemed so.

It has a few nice properties:

- if one edit triggers several workflow triggers, only one reset will happen.
- In tests, if we do pay attention at what we're doing, we can group portal types / accessor / base category setups and minimize the number of resets

Why did I care so much about the number of resets? With new accessor generation, we do a bit more during generation; and especially the generation of basic properties are very costly. So chaining several resets *is* costly, much more than two aq_resets.