senaite.core Documentation

Release 2.0.0rc3

Riding Bytes
Naralabs

Oct 18, 2020

Contents

Documentation for developers

senaite.core is an add-on for Plone CMS that provides the main logic, contents, behavior and user interface for SENAITE LIMS.

It is intended to be imported automatically as a dependency of senaite.lims Meta Installation Package and/or other SENAITE products, and it should **not** be installed alone.

Table of Contents:

CHAPTER 1

License

CHAPTER 2

Code conventions

This guide explains the code conventions used in `senaite.core`, but extends to any *senaite* add-on.

Note: There is a lot of legacy code that does not adhere to these conventions. Still, refactorings and new code should.

2.1 Catalogs

2.1.1 Indexes naming

Names for catalog indexes don't follow the `CamelCase` naming convention, rather all in lowercase and words separated by _:

- Bad: `getSampleTypeUID`
- Good: `sampletype_uid`

2.1.2 Plural and singular forms

Indexes will be written in singular form. There are a few exceptions, mostly those that come from Plone or Zope (e.g. `allowedRolesAndUsers`). Quite frequently, the plural form is used wrongly because the name of the meta-type index to use leads us to think about the plural form: e.g. `KeywordIndex`, that indexes a sequence of keywords. Is better to think about how the searches against the index work.

For instance, for a given object, a `KeywordIndex` will store a sequence of keywords, but if we do a search for single or multiple keywords against this index, the search will only return those items that have at least one of the keywords. And if there are multiple keyword matches for same item, the system will only return the item once. Since we can query for any index (`FieldIndex`, `KeywordIndex`, etc.) using a list, it does not make sense to use the plural form. In fact, if you are forced to add an index in plural form because a given index with same name, but in singular already exists, probably the index in singular is a `FieldIndex`, that does not allow you to store multiple values. In such case, the best approach is to change the meta-type of the existing index from `FieldIndex` to `KeywordIndex`.

- Bad: `sampletype_titles`
- Good: `sampletype_title`

2.1.3 Metadata fields naming

Metadata fields use the `get` prefix and eventually might follow the `CamelCase` naming convention. The reason is that, at present time, SENAITE still uses Archetypes, and `ATContentType`'s mutators for Schema fields follow this naming convention. Since one would expect the name of the metadata field to match with the name of the function from the object, we keep same convention.

- Bad: *SampleTypeUID*
- Good: *getSampleTypeUID*

2.1.4 Plural and singular forms

For metadata fields, use plural forms when the field returns a list and use singular when the field returns a single value.

CHAPTER 3

Adapters

Adapters are the mechanism that allows to extend and change the behavior of not only senaite.core, but also Plone and Zope. SENAITE comes with plenty of hooks for adapters that can be used for a wide range of purposes, such as changing the default behavior for the generation of IDs, add columns and filters in listings, new types of reports, trap events, etc.

In this chapter, the most commonly used adapters for SENAITE are discussed.

3.1 Listing Searchable Text Index

The Listing Searchable Text Index (`listing_searchable_text`) is mostly used for wide searches in listings. It is a `TextIndexNG3` type index present in most catalogs. To fill this index, SENAITE concatenates the values from all fields registered as metadata columns for the given object and catalog. The value is then converted to unicode and stored. This is the default behavior, but it can also be extended or customized by means of two mechanisms:

- By adding your own indexer and explicitly tell the values to exclude/include
- By setting up an adapter implementing `IListingSearchableTextProvider`

The first mechanism can be achieved by simply calling the function *get_searchable_text_tokens* with the correct parameters. For instance, a new indexer might look like follows:

```
@indexer(IMyContentType, IMyCatalog)
def listing_searchable_text(instance):

    # Metadata fields to not include in the index
    exclude = ["metadata_column_2", ]

    # Additional non-metadata fields to include in the index
    include = ["metadata_column_1", "metadata_column_3", ]

    # Generate the list of terms
    tokens = get_searchable_text_tokens(instance, my_catalog_name,
                                        exclude_field_names=exclude,
```

(continues on next page)

(continued from previous page)

```
                                        include_field_names=include)
    return u" ".join(tokens)
```

The second mechanism involves the creation of an adapter that implements `IListingSearchableTextProvider`. For instance:

```
@adapter(IAnalysisRequest)
@implementer(IListingSearchableTextProvider)
class ListingSearchableTextProvider(object):
    """Adapter for Analysis Request Listing Searchable Text Index
    """

    def __init__(self, context, catalog):
        self.context = context
        self.catalog = catalog

    def get_patient(self):
        return self.context.getField("Patient").get(self.context)

    def get_fullname(self):
        patient = self.get_patient()
        if not patient:
            return ""
        return patient.get_fullname()

    def get_code(self):
        patient = self.get_patient()
        if not patient:
            return ""
        return patient.get_code()

    def __call__(self):
        return [self.get_code(), self.get_fullname()]
```

In this case, the object implementing `IAnalysisRequest` has an additional field "Patient", a `ReferenceField` that relates to another object of type `Patient`. With this adapter, we make the system to include the fullname and the code of the patient to the contents of the searchable text index.

CHAPTER 4

Doctests

4.1 SENAITE LIMS API

The SENAITE LIMS API provides single functions for single purposes. This Test builds completely on the API without any further imports needed.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API
```

4.2 API

The purpose of this API is to help coders to follow the DRY principle (Don't Repeat Yourself). It also ensures that the most effective and efficient method is used to achieve a task.

Import it first:

```
>>> from bika.lims import api
```

4.2.1 Setup the test user

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.2.2 Getting the Portal

The Portal is the SENAITE LIMS root object:

```
>>> portal = api.get_portal()
>>> portal
<PloneSite at /plone>
```

4.2.3 Getting the SENAITE Setup object

The Setup object gives access to all of the SENAITE configuration settings:

```
>>> setup = api.get_setup()
>>> setup
<BikaSetup at /plone/bika_setup>

>>> bika_setup = api.get_bika_setup()
>>> setup
<BikaSetup at /plone/bika_setup>

>>> setup == bika_setup
True
```

4.2.4 Creating new Content

Creating new contents in SENAITE LIMS requires some special knowledge. This function helps to do it right and creates a content for you.

Here we create a new *Client* in the *plone/clients* folder:

```
>>> client = api.create(portal.clients, "Client", title="Test Client")
>>> client
<Client at /plone/clients/client-1>

 >>> client.Title()
 'Test Client'
```

4.2.5 Getting a Tool

There are many ways to get a tool in SENAITE LIMS / Plone. This function centralizes this functionality and makes it painless:

```
>>> api.get_tool("bika_setup_catalog")
<BikaSetupCatalog at /plone/bika_setup_catalog>
```

Trying to fetch an non-existing tool raises a custom *APIError*.

```
>>> api.get_tool("NotExistingTool")
Traceback (most recent call last):
[...]
APIError: No tool named 'NotExistingTool' found.
```

This error can also be used for custom methods with the *fail* function:

```
>>> api.fail("This failed badly")
Traceback (most recent call last):
[...]
APIError: This failed badly
```

4.2.6 Getting an Object

Getting the object from a catalog brain is a common task.

This function provides an unified interface to portal objects **and** brains. Furthermore it is idempotent, so it can be called multiple times in a row:

We will demonstrate the usage on the client object we created above:

```
>>> api.get_object(client)
<Client at /plone/clients/client-1>

>>> api.get_object(api.get_object(client))
<Client at /plone/clients/client-1>
```

Now we show it with catalog results:

```
>>> portal_catalog = api.get_tool("portal_catalog")
>>> brains = portal_catalog(portal_type="Client")
>>> brains
[<Products.ZCatalog.Catalog.mybrains object at 0x...>]

>>> brain = brains[0]

>>> api.get_object(brain)
<Client at /plone/clients/client-1>

>>> api.get_object(api.get_object(brain))
<Client at /plone/clients/client-1>
```

The function also accepts a UID:

```
>>> api.get_object(api.get_uid(brain))
<Client at /plone/clients/client-1>
```

And returns the portal object when UID==”0”

```
>>> api.get_object("0")
<PloneSite at /plone>
```

No supported objects raise an error:

```
>>> api.get_object(object())
Traceback (most recent call last):
[...]
APIError: <object object at 0x...> is not supported.

>>> api.get_object("i_am_not_an_uid")
Traceback (most recent call last):
[...]
APIError: 'i_am_not_an_uid' is not supported.
```

However, if a *default* value is provided, the default will be returned in such a case instead:

```
>>> api.get_object(object(), default=None) is None
True
```

To check if an object is supported, e.g. is an ATCT, Dexterity, ZCatalog or Portal object, we can use the *is_object* function:

```
>>> api.is_object(client)
True

>>> api.is_object(brain)
True

>>> api.is_object(api.get_portal())
True

>>> api.is_object(None)
False

>>> api.is_object(object())
False
```

4.2.7 Checking if an Object is the Portal

Sometimes it can be handy to check if the current object is the portal:

```
>>> api.is_portal(portal)
True

>>> api.is_portal(client)
False

>>> api.is_portal(object())
False
```

4.2.8 Checking if an Object is a Catalog Brain

Knowing if we have an object or a brain can be handy. This function checks this for you:

```
>>> api.is_brain(brain)
True

>>> api.is_brain(api.get_object(brain))
False

>>> api.is_brain(object())
False
```

4.2.9 Checking if an Object is a Dexterity Content

This function checks if an object is a *Dexterity* content type:

```
>>> api.is_dexterity_content(client)
False

>>> api.is_dexterity_content(portal)
False
```

We currently have no *Dexterity* contents, so testing this comes later...

4.2.10 Checking if an Object is an AT Content

This function checks if an object is an *Archetypes* content type:

```
>>> api.is_at_content(client)
True

>>> api.is_at_content(portal)
False

>>> api.is_at_content(object())
False
```

4.2.11 Getting the Schema of a Content

The schema contains the fields of a content object. Getting the schema is a common task, but differs between *ATContentType* based objects and *Dexterity* based objects. This function brings it under one umbrella:

```
>>> schema = api.get_schema(client)
>>> schema
<Products.Archetypes.Schema.Schema object at 0x...>
```

Catalog brains are also supported:

```
>>> api.get_schema(brain)
<Products.Archetypes.Schema.Schema object at 0x...>
```

4.2.12 Getting the Fields of a Content

The fields contain all the values that an object holds and are therefore responsible for getting and setting the information.

This function returns the fields as a dictionary mapping of *{"key": value}*:

```
>>> fields = api.get_fields(client)
>>> fields.get("ClientID")
<Field ClientID(string:rw)>
```

Catalog brains are also supported:

```
>>> api.get_fields(brain).get("ClientID")
<Field ClientID(string:rw)>
```

4.2.13 Getting the ID of a Content

Getting the ID is a common task in SENAITE LIMS. This function takes care that catalog brains are not woken up for this task:

```
>>> api.get_id(portal)
'plone'

>>> api.get_id(client)
'client-1'

>>> api.get_id(brain)
'client-1'
```

4.2.14 Getting the Title of a Content

Getting the Title is a common task in SENAITE LIMS. This function takes care that catalog brains are not woken up for this task:

```
>>> api.get_title(portal)
'SENAITE LIMS'

>>> api.get_title(client)
'Test Client'

>>> api.get_title(brain)
'Test Client'
```

4.2.15 Getting the Description of a Content

Getting the Description is a common task in SENAITE LIMS. This function takes care that catalog brains are not woken up for this task:

```
>>> api.get_description(portal)
''

>>> api.get_description(client)
''

>>> api.get_description(brain)
''
```

4.2.16 Getting the UID of a Content

Getting the UID is a common task in SENAITE LIMS. This function takes care that catalog brains are not woken up for this task.

The portal object actually has no UID. This funciton defines it therefore to be *0*:

```
>>> api.get_uid(portal)
'0'

>>> uid_client = api.get_uid(client)
```

(continues on next page)

(continued from previous page)

```
>>> uid_client_brain = api.get_uid(brain)
>>> uid_client is uid_client_brain
True
```

If a UID is passed to the function, it will return the value unchanged:

```
>>> api.get_uid(uid_client) == uid_client
True
```

4.2.17 Getting the URL of a Content

Getting the URL is a common task in SENAITE LIMS. This function takes care that catalog brains are not woken up for this task:

```
>>> api.get_url(portal)
'http://nohost/plone'

>>> api.get_url(client)
'http://nohost/plone/clients/client-1'

>>> api.get_url(brain)
'http://nohost/plone/clients/client-1'
```

4.2.18 Getting the Icon of a Content

```
>>> api.get_icon(client)
'<img width="16" height="16" src="http://nohost/plone/senaite_theme/icon/client"␣
↪title="Test Client" />'
```

```
>>> api.get_icon(brain)
'<img width="16" height="16" src="http://nohost/plone/senaite_theme/icon/client"␣
↪title="Test Client" />'
```

```
>>> api.get_icon(client, html_tag=False)
'http://nohost/plone/senaite_theme/icon/client'
```

```
>>> api.get_icon(client, html_tag=False)
'http://nohost/plone/senaite_theme/icon/client'
```

4.2.19 Getting a catalog brain by UID

This function finds a catalog brain by its uinique ID (UID):

```
>>> api.get_brain_by_uid(api.get_uid(client))
<Products.Archetypes.UIDCatalog.plugbrains object at ...>
```

4.2.20 Getting an object by UID

This function finds an object by its uinique ID (UID). The portal object with the defined UId of '0' is also supported:

```
>>> api.get_object_by_uid('0')
<PloneSite at /plone>

>>> api.get_object_by_uid(uid_client)
<Client at /plone/clients/client-1>

>>> api.get_object_by_uid(uid_client_brain)
<Client at /plone/clients/client-1>
```

If a default value is provided, the function will never fail. Any exception or error will result in the default value being returned:

```
>>> api.get_object_by_uid('invalid uid', 'default')
'default'

>>> api.get_object_by_uid(None, 'default')
'default'
```

4.2.21 Getting an object by Path

This function finds an object by its physical path:

```
>>> api.get_object_by_path('/plone')
<PloneSite at /plone>

>>> api.get_object_by_path('/plone/clients/client-1')
<Client at /plone/clients/client-1>
```

Paths outside the portal raise an error:

```
>>> api.get_object_by_path('/root')
Traceback (most recent call last):
[...]
APIError: Not a physical path inside the portal.
```

Any exception returns default value:

```
>>> api.get_object_by_path('/invaid/path', 'default')
'default'

>>> api.get_object_by_path(None, 'default')
'default'
```

4.2.22 Getting the Physical Path of an Object

The physical path describes exactly where an object is located inside the portal. This function unifies the different approaches to get the physical path and does so in the most efficient way:

```
>>> api.get_path(portal)
'/plone'

>>> api.get_path(client)
'/plone/clients/client-1'
```

(continues on next page)

(continued from previous page)

```
>>> api.get_path(brain)
'/plone/clients/client-1'

>>> api.get_path(object())
Traceback (most recent call last):
[...]
APIError: <object object at 0x...> is not supported.
```

4.2.23 Getting the Physical Parent Path of an Object

This function returns the physical path of the parent object:

```
>>> api.get_parent_path(client)
'/plone/clients'

>>> api.get_parent_path(brain)
'/plone/clients'
```

However, this function goes only up to the portal object:

```
>>> api.get_parent_path(portal)
'/plone'
```

Like with the other functions, only portal objects are supported:

```
>>> api.get_parent_path(object())
Traceback (most recent call last):
[...]
APIError: <object object at 0x...> is not supported.
```

4.2.24 Getting the Parent Object

This function returns the parent object:

```
>>> api.get_parent(client)
<ClientFolder at /plone/clients>
```

Brains are also supported:

```
>>> api.get_parent(brain)
<ClientFolder at /plone/clients>
```

The function can also use a catalog query on the *portal_catalog* and return a brain, if the passed parameter *catalog_search* was set to true.

```
>>> api.get_parent(client, catalog_search=True)
<Products.ZCatalog.Catalog.mybrains object at 0x...>

>>> api.get_parent(brain, catalog_search=True)
<Products.ZCatalog.Catalog.mybrains object at 0x...>
```

However, this function goes only up to the portal object:

```
>>> api.get_parent(portal)
<PloneSite at /plone>
```

Like with the other functions, only portal objects are supported:

```
>>> api.get_parent(object())
Traceback (most recent call last):
[...]
APIError: <object object at 0x...> is not supported.
```

4.2.25 Searching Objects

Searching in SENAITE LIMS requires knowledge in which catalog the object is indexed. This function unifies all SENAITE LIMS catalog to a single search interface:

```
>>> results = api.search({'portal_type': 'Client'})
>>> results
[<Products.ZCatalog.Catalog.mybrains object at 0x...>]
```

Multiple content types are also supported:

```
>>> results = api.search({'portal_type': ['Client', 'ClientFolder'], 'sort_on': 'getId
↪'})
>>> map(api.get_id, results)
['client-1', 'clients']
```

Now we create some objects which are located in the *bika_setup_catalog*:

```
>>> instruments = bika_setup.bika_instruments
>>> instrument1 = api.create(instruments, "Instrument", title="Instrument-1")
>>> instrument2 = api.create(instruments, "Instrument", title="Instrument-2")
>>> instrument3 = api.create(instruments, "Instrument", title="Instrument-3")

>>> results = api.search({'portal_type': 'Instrument', 'sort_on': 'getId'})
>>> len(results)
3

>>> map(api.get_id, results)
['instrument-1', 'instrument-2', 'instrument-3']
```

Queries which result in multiple catalogs will be refused, as it would require manual merging and sorting of the results afterwards. Thus, we fail here:

```
>>> results = api.search({'portal_type': ['Client', 'ClientFolder', 'Instrument'],
↪'sort_on': 'getId'})
Traceback (most recent call last):
[...]
APIError: Multi Catalog Queries are not supported!
```

Catalog queries w/o any *portal_type*, default to the *portal_catalog*, which will not find the following items:

```
>>> analysiscategories = bika_setup.bika_analysiscategories
>>> analysiscategory1 = api.create(analysiscategories, "AnalysisCategory", title="AC-1
↪")
>>> analysiscategory2 = api.create(analysiscategories, "AnalysisCategory", title="AC-2
↪")
```

(continues on next page)

(continued from previous page)

```
>>> analysiscategory3 = api.create(analysiscategories, "AnalysisCategory", title="AC-3
↪")

>>> results = api.search({"id": "analysiscategory-1"})
>>> len(results)
0
```

Would we add the *portal_type*, the search function would ask the *archetype_tool* for the right catalog, and it would return a result:

```
>>> results = api.search({"portal_type": "AnalysisCategory", "id": "analysiscategory-1
↪"})
>>> len(results)
1
```

We could also explicitly define a catalog to achieve the same:

```
>>> results = api.search({"id": "analysiscategory-1"}, catalog="bika_setup_catalog")
>>> len(results)
1
```

To see inactive or dormant items, we must explicitly query them of filter them afterwars manually:

```
>>> results = api.search({"portal_type": "AnalysisCategory", "id": "analysiscategory-1
↪"})
>>> len(results)
1
```

Now we deactivate the item:

```
>>> analysiscategory1 = api.do_transition_for(analysiscategory1, 'deactivate')
>>> api.is_active(analysiscategory1)
False
```

The search will still find the item:

```
>>> results = api.search({"portal_type": "AnalysisCategory", "id": "analysiscategory-1
↪"})
>>> len(results)
1
```

Unless we filter it out manually:

```
>>> len(filter(api.is_active, results))
0
```

Or provide a correct query:

```
>>> results = api.search({"portal_type": "AnalysisCategory", "id": "analysiscategory-1
↪", "is_active": False})
>>> len(results)
1
```

4.2.26 Getting the registered Catalogs

SENAITE LIMS uses multiple catalogs registered via the Archetype Tool. This function returns a list of registered catalogs for a brain or object:

```
>>> api.get_catalogs_for(client)
[...]

>>> api.get_catalogs_for(instrument1)
[...]

>>> api.get_catalogs_for(analysiscategory1)
[...]
```

4.2.27 Getting an Attribute of an Object

This function handles attributes and methods the same and returns their value. It also handles security and is able to return a default value instead of raising an *Unauthorized* error:

```
>>> uid_brain = api.safe_getattr(brain, "UID")
>>> uid_obj = api.safe_getattr(client, "UID")

>>> uid_brain == uid_obj
True

>>> api.safe_getattr(brain, "review_state")
'active'

>>> api.safe_getattr(brain, "NONEXISTING")
Traceback (most recent call last):
[...]
APIError: Attribute 'NONEXISTING' not found.

>>> api.safe_getattr(brain, "NONEXISTING", "")
''
```

4.2.28 Getting the Portal Catalog

This tool is needed so often, that this function just returns it:

```
>>> api.get_portal_catalog()
<CatalogTool at /plone/portal_catalog>
```

4.2.29 Getting the Review History of an Object

The review history gives information about the objects' workflow changes:

```
>>> review_history = api.get_review_history(client)
>>> sorted(review_history[0].items())
[('action', None), ('actor', 'test_user_1_'), ('comments', ''), ('review_state',
↪'active'), ('time', DateTime('...'))]
```

4.2.30 Getting the Revision History of an Object

The review history gives information about the objects' workflow changes:

```
>>> revision_history = api.get_revision_history(client)
>>> sorted(revision_history[0])
['action', 'actor', 'actor_home', 'actorid', 'comments', 'review_state', 'state_title
↪', 'time', 'transition_title', 'type']
>>> revision_history[0]["transition_title"]
u'Create'
```

4.2.31 Getting the assigned Workflows of an Object

This function returns all assigned workflows for a given object:

```
>>> api.get_workflows_for(bika_setup)
('senaite_setup_workflow',)

>>> api.get_workflows_for(client)
('senaite_client_workflow',)
```

This function also supports the portal_type as parameter:

```
>>> api.get_workflows_for(api.get_portal_type(client))
('senaite_client_workflow',)
```

4.2.32 Getting the Workflow Status of an Object

This function returns the state of a given object:

```
>>> api.get_workflow_status_of(client)
'active'
```

It is also able to return the state from a brain without waking it up:

```
>>> api.get_workflow_status_of(brain)
'active'
```

It is also capable to get the state of another state variable:

```
>>> api.get_workflow_status_of(client, "review_state")
'active'
```

Deactivate the client:

```
>>> api.do_transition_for(client, "deactivate")
<Client at /plone/clients/client-1>

>>> api.get_workflow_status_of(client)
'inactive'
```

Reactivate the client:

```
>>> api.do_transition_for(client, "activate")
<Client at /plone/clients/client-1>

>>> api.get_workflow_status_of(client)
'active'
```

4.2.33 Getting the available transitions for an object

This function returns all possible transitions from all workflows in the object's workflow chain.

Let's create a Batch. It should allow us to invoke two different transitions: 'close' and 'cancel':

```
>>> batch1 = api.create(portal.batches, "Batch", title="Test Batch")
>>> transitions = api.get_transitions_for(batch1)
>>> len(transitions)
2
```

The transitions are returned as a list of dictionaries. Since we cannot rely on the order of dictionary keys, we will have to satisfy ourselves here with checking that the two expected transitions are present in the return value:

```
>>> 'Close' in [t['title'] for t in transitions]
True
>>> 'Cancel' in [t['title'] for t in transitions]
True
```

4.2.34 Getting the creation date of an object

This function returns the creation date of a given object:

```
>>> created = api.get_creation_date(client)
>>> created
DateTime('...')
```

4.2.35 Getting the modification date of an object

This function returns the modification date of a given object:

```
>>> modified = api.get_modification_date(client)
>>> modified
DateTime('...')
```

4.2.36 Getting the review state of an object

This function returns the review state of a given object:

```
>>> review_state = api.get_review_status(client)
>>> review_state
'active'
```

It should also work for catalog brains:

```
>>> portal_catalog = api.get_tool("portal_catalog")
>>> results = portal_catalog({"portal_type": "Client", "UID": api.get_uid(client)})
>>> len(results)
1
>>> api.get_review_status(results[0]) == review_state
True
```

4.2.37 Getting the registered Catalogs of an Object

This function returns a list of all registered catalogs within the *archetype_tool* for a given portal_type or object:

```
>>> api.get_catalogs_for(client)
[...]
```

It also supports the *portal_type* as a parameter:

```
>>> api.get_catalogs_for("Analysis")
[...]
```

4.2.38 Transitioning an Object

This function performs a workflow transition and returns the object:

```
>>> client = api.do_transition_for(client, "deactivate")
>>> api.is_active(client)
False

>>> client = api.do_transition_for(client, "activate")
>>> api.is_active(client)
True
```

4.2.39 Getting inactive/cancellation state of different workflows

There are two workflows allowing an object to be set inactive. We provide the is_active function to return False if an item is set inactive with either of these workflows.

In the search() test above, the is_active function's handling of brain states is tested. Here, I just want to test if object states are handled correctly.

For setup types, we use senaite_deactivable_type_workflow:

```
>>> method1 = api.create(portal.methods, "Method", title="Test Method")
>>> api.is_active(method1)
True
>>> method1 = api.do_transition_for(method1, 'deactivate')
>>> api.is_active(method1)
False
```

For transactional types, senaite_cancellable_type_workflow is used:

```
>>> maintenance_task = api.create(instrument1, "InstrumentMaintenanceTask", title=
↪"Maintenance Task for Instrument 1")
>>> api.is_active(maintenance_task)
```

(continues on next page)

(continued from previous page)

```
True
>>> maintenance_task = api.do_transition_for(maintenance_task, "cancel")
>>> api.is_active(maintenance_task)
False
```

But there are custom workflows that can also provide *cancel* transition, like *senaite_batch_workflow*, to which *Batch* type is bound:

```
>>> batch1 = api.create(portal.batches, "Batch", title="Test Batch")
>>> api.is_active(batch1)
True
>>> batch1 = api.do_transition_for(batch1, 'cancel')
>>> api.is_active(batch1)
False
```

4.2.40 Getting the granted Roles for a certain Permission on an Object

This function returns a list of Roles, which are granted the given Permission for the passed in object:

```
>>> api.get_roles_for_permission("Modify portal content", portal)
['LabClerk', 'LabManager', 'Manager', 'Owner']

>>> api.get_roles_for_permission("Modify portal content", bika_setup)
['LabClerk', 'LabManager', 'Manager']
```

4.2.41 Checking if an Object is Versionable

Some contents in SENAITE LIMS support versioning. This function checks this for you.

Instruments are not versionable:

```
>>> api.is_versionable(instrument1)
False
```

Analysisservices are versionable:

```
>>> analysisservices = bika_setup.bika_analysisservices
>>> analysisservice1 = api.create(analysisservices, "AnalysisService", title=
↪"AnalysisService-1")
>>> analysisservice2 = api.create(analysisservices, "AnalysisService", title=
↪"AnalysisService-2")
>>> analysisservice3 = api.create(analysisservices, "AnalysisService", title=
↪"AnalysisService-3")

>>> api.is_versionable(analysisservice1)
True
```

4.2.42 Getting the Version of an Object

This function returns the version as an integer:

```
>>> api.get_version(analysisservice1)
0
```

Calling *processForm* bumps the version:

```
>>> analysisservice1.processForm()
>>> api.get_version(analysisservice1)
1
```

4.2.43 Getting a Browser View

Getting a browser view is a common task in SENAITE LIMS:

```
>>> api.get_view("plone")
<Products.Five.browser.metaconfigure.Plone object at 0x...>


>>> api.get_view("workflow_action")
<Products.Five.browser.metaconfigure.WorkflowActionHandler object at 0x...>
```

4.2.44 Getting the Request

This function will return the global request object:

```
>>> api.get_request()
<HTTPRequest, URL=http://nohost>
```

4.2.45 Getting a Group

Users in SENAITE LIMS are managed in groups. A common group is the *Clients* group, where all users of client contacts are grouped. This function gives easy access and is also idempotent:

```
>>> clients_group = api.get_group("Clients")
>>> clients_group
<GroupData at /plone/portal_groupdata/Clients used for /plone/acl_users/source_groups>

>>> api.get_group(clients_group)
<GroupData at /plone/portal_groupdata/Clients used for /plone/acl_users/source_groups>
```

Non-existing groups are not found:

```
>>> api.get_group("NonExistingGroup")
```

4.2.46 Getting a User

Users can be fetched by their user id. The function is idempotent and handles user objects as well:

```
>>> from plone.app.testing import TEST_USER_ID
>>> user = api.get_user(TEST_USER_ID)
>>> user
```

(continues on next page)

(continued from previous page)

```
<Products.PlonePAS.tools.memberdata.MemberData object at 0x...>

>>> api.get_user(api.get_user(TEST_USER_ID))
<Products.PlonePAS.tools.memberdata.MemberData object at 0x...>
```

Non-existing users are not found:

```
>>> api.get_user("NonExistingUser")
```

4.2.47 Getting User Properties

User properties, like the email or full name, are stored as user properties. This means that they are not on the user object. This function retrieves these properties for you:

```
>>> properties = api.get_user_properties(TEST_USER_ID)
>>> sorted(properties.items())
[('description', ''), ('email', ''), ('error_log_update', 0.0), ('ext_editor', False),
↪ ...]

>>> sorted(api.get_user_properties(user).items())
[('description', ''), ('email', ''), ('error_log_update', 0.0), ('ext_editor', False),
↪ ...]
```

An empty property dict is returned if no user could be found:

```
>>> api.get_user_properties("NonExistingUser")
{}

>>> api.get_user_properties(None)
{}
```

4.2.48 Getting Users by their Roles

```
>>> from operator import methodcaller
```

Roles in SENAITE LIMS are basically a name for one or more permissions. For example, a *LabManager* describes a role which is granted the most permissions.

So first I'll add some users with some different roles:

```
>>> for user in [{'username': 'labmanager_1', 'roles': ['LabManager']},
...              {'username': 'labmanager_2', 'roles': ['LabManager']},
...              {'username': 'sampler_1', 'roles': ['Sampler']},
...              {'username': 'client_1', 'roles': ['Client']}]:
...     member = portal.portal_registration.addMember(
...         user['username'], user['username'],
...         properties={'username': user['username'],
...                     'email': user['username'] + "@example.com",
...                     'fullname': user['username']})
...     setRoles(portal, user['username'], user['roles'])
...     # If user is a LabManager, add Owner local role on clients folder
...     # TODO ask @ramonski, is this still required?
```

(continues on next page)

(continued from previous page)

```
...     if 'LabManager' in user['roles']:
...         portal.clients.manage_setLocalRoles(user['username'], ['Owner'])
```

To see which users are granted a certain role, you can use this function:

```
>>> labmanagers = api.get_users_by_roles(["LabManager"])
>>> sorted(labmanagers, key=methodcaller('getId'))
[<PloneUser 'labmanager_1'>, <PloneUser 'labmanager_2'>]
```

A single value can also be passed into this function:

```
>>> sorted(api.get_users_by_roles("Sampler"), key=methodcaller('getId'))
[<PloneUser 'sampler_1'>]
```

4.2.49 Getting the Current User

Getting the current logged in user:

```
>>> api.get_current_user()
<Products.PlonePAS.tools.memberdata.MemberData object at 0x...
```

4.2.50 Getting the Contact associated to a Plone user

Getting a Plone user previously registered with no contact assigned:

```
>>> user = api.get_user('labmanager_1')
>>> contact = api.get_user_contact(user)
>>> contact is None
True
```

Assign a new contact to this user:

```
>>> labcontacts = bika_setup.bika_labcontacts
>>> labcontact = api.create(labcontacts, "LabContact", Firstname="Lab", Lastname=
↪"Manager")
>>> labcontact.setUser(user)
True
```

And get the contact associated to the user:

```
>>> api.get_user_contact(user)
<LabContact at /plone/bika_setup/bika_labcontacts/labcontact-1>
```

As well as if we specify only *LabContact* type:

```
>>> api.get_user_contact(user, ['LabContact'])
<LabContact at /plone/bika_setup/bika_labcontacts/labcontact-1>
```

But fails if we specify only *Contact* type:

```
>>> nuser = api.get_user_contact(user, ['Contact'])
>>> nuser is None
True
```

4.2.51 Getting the Contact Client

Getting the current client the current user belongs to:

```
>>> api.get_current_client() is None
True
```

And still fails if we use a user that is not associated to a client:

```
>>> api.get_user_client(user) is None
True

>>> api.get_user_client(labcontact) is None
True
```

Try now with a valid contact:

```
>>> client_user = api.get_user('client_1')
>>> contact1 = api.create(client, "Contact", Firstname="Lost", Lastname="Nomad")
>>> contact1.setUser(client_user)
True

>>> api.get_user_client(contact1)
<Client at /plone/clients/client-1>

>>> api.get_user_client(client_user)
<Client at /plone/clients/client-1>
```

4.2.52 Creating a Cache Key

This function creates a good cache key for a generic object or brain:

```
>>> key1 = api.get_cache_key(client)
>>> key1
'Client-client-1-...'
```

This can be also done for a catalog result brain:

```
>>> portal_catalog = api.get_tool("portal_catalog")
>>> brains = portal_catalog({"portal_type": "Client", "UID": api.get_uid(client)})
>>> key2 = api.get_cache_key(brains[0])
>>> key2
'Client-client-1-...'
```

The two keys should be equal:

```
>>> key1 == key2
True
```

The key should change when the object get modified:

```
>>> client.setClientID("TESTCLIENT")
>>> client.processForm()
>>> key3 = api.get_cache_key(client)
>>> key3 != key1
True
```

~~ important:: Workflow changes do not change the modification date! A custom event subscriber will update it therefore.

A workflow transition should also change the cache key:

```
>>> _ = api.do_transition_for(client, transition="deactivate")
>>> api.is_active(client)
False
>>> key4 = api.get_cache_key(client)
>>> key4 != key3
True
```

4.2.53 SENAITE Cache Key decorator

This decorator can be used for *plone.memoize* cache decorators in classes. The decorator expects that the first argument is the class instance (*self*) and the second argument a brain or object:

```
>>> from plone.memoize.volatile import cache

>>> class SENAITEClass(object):
...     @cache(api.bika_cache_key_decorator)
...     def get_very_expensive_calculation(self, obj):
...         print "very expensive calculation"
...         return "calculation result"
```

Calling the (expensive) method of the class does the calculation just once:

```
>>> instance = SENAITEClass()
>>> instance.get_very_expensive_calculation(client)
very expensive calculation
'calculation result'
>>> instance.get_very_expensive_calculation(client)
'calculation result'
```

The decorator can also handle brains:

```
>>> instance = SENAITEClass()
>>> portal_catalog = api.get_tool("portal_catalog")
>>> brain = portal_catalog(portal_type="Client")[0]
>>> instance.get_very_expensive_calculation(brain)
very expensive calculation
'calculation result'
>>> instance.get_very_expensive_calculation(brain)
'calculation result'
```

4.2.54 ID Normalizer

Normalizes a string to be usable as a system ID:

```
>>> api.normalize_id("My new ID")
'my-new-id'
```

```
>>> api.normalize_id("Really/Weird:Name;")
'really-weird-name'
```

```
>>> api.normalize_id(None)
Traceback (most recent call last):
[...]
APIError: Type of argument must be string, found '<type 'NoneType'>'
```

4.2.55 File Normalizer

Normalizes a string to be usable as a file name:

```
>>> api.normalize_filename("My new ID")
'My new ID'
```

```
>>> api.normalize_filename("Really/Weird:Name;")
'Really-Weird-Name'
```

```
>>> api.normalize_filename(None)
Traceback (most recent call last):
[...]
APIError: Type of argument must be string, found '<type 'NoneType'>'
```

4.2.56 Check if an UID is valid

Checks if an UID is a valid 23 alphanumeric uid:

```
>>> api.is_uid("ajw2uw9")
False
```

```
>>> api.is_uid(None)
False
```

```
>>> api.is_uid("")
False
```

```
>>> api.is_uid('0e1dfc3d10d747bf999948a071bc161e')
True
```

Per convention we assume "0" is the uid for portal object (PloneSite):

```
>>> api.is_uid("0")
True
```

Checks if an UID is a valid 23 alphanumeric uid and with a brain:

```
>>> api.is_uid("ajw2uw9", validate=True)
False
```

```
>>> api.is_uid(None, validate=True)
False
```

```
>>> api.is_uid("", validate=True)
False
```

```
>>> api.is_uid('0e1dfc3d10d747bf999948a071bc161e', validate=True)
False
```

```
>>> api.is_uid("0", validate=True)
True
```

```
>>> asfolder = self.portal.bika_setup.bika_analysisservices
>>> serv = api.create(asfolder, "AnalysisService", title="AS test")
>>> serv.setKeyword("as_test")
>>> uid = serv.UID()
>>> api.is_uid(uid, validate=True)
True
```

4.2.57 Check if a Date is valid

Do some imports first:

```
>>> from datetime import datetime
>>> from DateTime import DateTime
```

Checks if a DateTime is valid:

```
>>> now = DateTime()
>>> api.is_date(now)
True
```

```
>>> now = datetime.now()
>>> api.is_date(now)
True
```

```
>>> now = DateTime(now)
>>> api.is_date(now)
True
```

```
>>> api.is_date(None)
False
```

```
>>> api.is_date('2018-04-23')
False
```

4.2.58 Try conversions to Date

Try to convert to DateTime:

```
>>> now = DateTime()
>>> zpdt = api.to_date(now)
>>> zpdt.ISO8601() == now.ISO8601()
True
```

```
>>> now = datetime.now()
>>> zpdt = api.to_date(now)
>>> pydt = zpdt.asdatetime()
```

Note that here, for the comparison between dates, we convert DateTime to python datetime, cause DateTime.strftime() is broken for timezones (always looks at system time zone, ignores the timezone and offset of the DateTime instance itself):

```
>>> pydt.strftime('%Y-%m-%dT%H:%M:%S') == now.strftime('%Y-%m-%dT%H:%M:%S')
True
```

Try the same, but with utcnow() instead:

```
>>> now = datetime.utcnow()
>>> zpdt = api.to_date(now)
>>> pydt = zpdt.asdatetime()
>>> pydt.strftime('%Y-%m-%dT%H:%M:%S') == now.strftime('%Y-%m-%dT%H:%M:%S')
True
```

Now we convert just a string formatted date:

```
>>> strd = "2018-12-01 17:50:34"
>>> zpdt = api.to_date(strd)
>>> zpdt.ISO8601()
'2018-12-01T17:50:34'
```

Now we convert just a string formatted date, but with timezone:

```
>>> strd = "2018-12-01 17:50:34 GMT+1"
>>> zpdt = api.to_date(strd)
>>> zpdt.ISO8601()
'2018-12-01T17:50:34+01:00'
```

We also check a bad date here (note the month is 13):

```
>>> strd = "2018-13-01 17:50:34"
>>> zpdt = api.to_date(strd)
>>> api.is_date(zpdt)
False
```

And with European format:

```
>>> strd = "01.12.2018 17:50:34"
>>> zpdt = api.to_date(strd)
>>> zpdt.ISO8601()
'2018-12-01T17:50:34'
```

```
>>> zpdt = api.to_date(None)
>>> zpdt is None
True
```

Use a string formatted date as fallback:

```
>>> strd = "2018-13-01 17:50:34"
>>> default_date = "2018-01-01 19:30:30"
>>> zpdt = api.to_date(strd, default_date)
>>> zpdt.ISO8601()
'2018-01-01T19:30:30'
```

Use a DateTime object as fallback:

```
>>> strd = "2018-13-01 17:50:34"
>>> default_date = "2018-01-01 19:30:30"
>>> default_date = api.to_date(default_date)
>>> zpdt = api.to_date(strd, default_date)
>>> zpdt.ISO8601() == default_date.ISO8601()
True
```

Use a datetime object as fallback:

```
>>> strd = "2018-13-01 17:50:34"
>>> default_date = datetime.now()
>>> zpdt = api.to_date(strd, default_date)
>>> dzpdt = api.to_date(default_date)
>>> zpdt.ISO8601() == dzpdt.ISO8601()
True
```

Use a non-conversionable value as fallback:

```
>>> strd = "2018-13-01 17:50:34"
>>> default_date = "something wrong here"
>>> zpdt = api.to_date(strd, default_date)
>>> zpdt is None
True
```

4.2.59 Check if floatable

```
>>> api.is_floatable(None)
False
```

```
>>> api.is_floatable("")
False
```

```
>>> api.is_floatable("31")
True
```

```
>>> api.is_floatable("31.23")
True
```

```
>>> api.is_floatable("-13")
True
```

```
>>> api.is_floatable("12,35")
False
```

4.2.60 Convert to a float number

```
>>> api.to_float("2")
2.0
```

```
>>> api.to_float("2.234")
2.234
```

With default fallback:

```
>>> api.to_float(None, 2)
2.0
```

```
>>> api.to_float(None, "2")
2.0
```

```
>>> api.to_float("", 2)
2.0
```

```
>>> api.to_float("", "2")
2.0
```

```
>>> api.to_float(2.1, 2)
2.1
```

```
>>> api.to_float("2.1", 2)
2.1
```

```
>>> api.to_float("2.1", "2")
2.1
```

4.2.61 Convert to an int number

```
>>> api.to_int(2)
2
```

```
>>> api.to_int("2")
2
```

```
>>> api.to_int(2.1)
2
```

```
>>> api.to_int("2.1")
2
```

With default fallback:

```
>>> api.to_int(None, 2)
2
```

```
>>> api.to_int(None, "2")
2
```

```
>>> api.to_int("", 2)
2
```

```
>>> api.to_int("2", 0)
2
```

```
>>> api.to_int(2, 0)
2
```

```
>>> api.to_int("as", None) is None
True
```

```
>>> api.to_int("as", "2")
2
```

4.2.62 Convert to minutes

```
>>> api.to_minutes(hours=1)
60
```

```
>>> api.to_minutes(hours=1.5, minutes=30)
120
```

```
>>> api.to_minutes(hours=0, minutes=0, seconds=0)
0
```

```
>>> api.to_minutes(minutes=120)
120
```

```
>>> api.to_minutes(hours="1", minutes="120", seconds="120")
182
```

```
>>> api.to_minutes(days=3)
4320
```

```
>>> api.to_minutes(minutes=122.4567)
122
```

```
>>> api.to_minutes(minutes=122.4567, seconds=6)
123
```

```
>>> api.to_minutes(minutes=122.4567, seconds=6, round_to_int=False)
122.55669999999999
```

4.2.63 Convert to dhm format

```
>>> api.to_dhm_format(hours=1)
'1h'
```

```
>>> api.to_dhm_format(hours=1.5, minutes=30)
'2h'
```

```
>>> api.to_dhm_format(hours=0, minutes=0, seconds=0)
''
```

```
>>> api.to_dhm_format(minutes=120)
'2h'
```

```
>>> api.to_dhm_format(hours="1", minutes="120", seconds="120")
'3h 2m'
```

```
>>> api.to_dhm_format(days=3)
'3d'
```

```
>>> api.to_dhm_format(days=3, minutes=140)
'3d 2h 20m'
```

```
>>> api.to_dhm_format(days=3, minutes=20)
'3d 0h 20m'
```

```
>>> api.to_dhm_format(minutes=122.4567)
'2h 2m'
```

```
>>> api.to_dhm_format(minutes=122.4567, seconds=6)
'2h 3m'
```

4.2.64 Get a registry record

Fetch a value of a registry record:

```
>>> key = "Products.CMFPlone.i18nl10n.override_dateformat.Enabled"
>>> api.get_registry_record(key)
False
```

If the record is not found, the default is returned:

```
>>> key = "non.existing.key"
>>> api.get_registry_record(key, default="NX_KEY")
'NX_KEY'
```

4.2.65 Create a display list

Static display lists, can look up on either side of the dict, and get them in sorted order. They are used in selection widgets.

The function can handle a list of key->value pairs:

```
>>> pairs = [["a", "A"], ["b", "B"]]
>>> api.to_display_list(pairs)
<DisplayList [('', ''), ('a', 'A'), ('b', 'B')] at ...>
```

It can also handle a single pair:

```
>>> pairs = ["z", "Z"]
>>> api.to_display_list(pairs)
<DisplayList [('', ''), ('z', 'Z')] at ...>
```

It can also handle a single string:

```
>>> api.to_display_list("x")
<DisplayList [('', ''), ('x', 'x')] at ...>
```

It can be sorted either by key or by value:

```
>>> pairs = [["b", 10], ["a", 100]]
>>> api.to_display_list(pairs)
<DisplayList [('', ''), ('a', 100), ('b', 10)] at ...>
```

```
>>> api.to_display_list(pairs, sort_by="value")
<DisplayList [('b', 10), ('a', 100), ('', '')] at ...>
```

4.2.66 Converting a text to HTML

This function converts newline (*n*) escape sequences in plain text to <*br/*> tags for HTML rendering.

The function can handle plain texts:

```
>>> text = "First\r\nSecond\r\nThird"
>>> api.text_to_html(text)
'<p>First\r<br/>Second\r<br/>Third</p>'
```

Unicodes texts work as well:

```
>>> text = u"Ä\r\nÖ\r\nÜ"
>>> api.text_to_html(text)
'<p>\xc3\x83\xc2\x84\r<br/>\xc3\x83\xc2\x96\r<br/>\xc3\x83\xc2\x9c</p>'
```

The outer <*p*> wrap can be also omitted:

```
>>> text = "One\r\nTwo"
>>> api.text_to_html(text, wrap=None)
'One\r<br/>Two'
```

Or changed to another tag:

```
>>> text = "One\r\nTwo"
>>> api.text_to_html(text, wrap="div")
'<div>One\r<br/>Two</div>'
```

Empty strings are returned unchanged:

```
>>> text = ""
>>> api.text_to_html(text, wrap="div")
''
```

4.3 API Analysis

The api_analysis provides single functions for single purposes especifically related with analyses.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_analysis
```

4.3.1 Test Setup

Needed Imports:

```
>>> import re
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.api.analysis import get_formatted_interval
>>> from bika.lims.api.analysis import is_out_of_range
>>> from bika.lims.content.analysisrequest import AnalysisRequest
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils import tmpID
>>> from bika.lims.workflow import doActionFor
>>> from bika.lims.workflow import getCurrentState
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), DuplicateVariation="0.5")
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID(), DuplicateVariation="0.5")
```

(continues on next page)

(continued from previous page)

```
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID(), DuplicateVariation="0.5")
>>> Mg = api.create(bikasetup.bika_analysisservices, "AnalysisService", title=
↪"Magnesium", Keyword="Mg", Price="20", Category=category.UID(), DuplicateVariation=
↪"0.5")
>>> service_uids = [api.get_uid(an) for an in [Cu, Fe, Au, Mg]]
```

Create an Analysis Specification for *Water*:

```
>>> sampletype_uid = api.get_uid(sampletype)
>>> rr1 = {"keyword": "Au", "min": "-5", "max":  "5", "warn_min": "-5.5", "warn_max":
↪"5.5"}
>>> rr2 = {"keyword": "Cu", "min": "10", "max": "20", "warn_min":  "9.5", "warn_max":
↪"20.5"}
>>> rr3 = {"keyword": "Fe", "min":  "0", "max": "10", "warn_min": "-0.5", "warn_max":
↪"10.5"}
>>> rr4 = {"keyword": "Mg", "min": "10", "max": "10"}
>>> rr = [rr1, rr2, rr3, rr4]
>>> specification = api.create(bikasetup.bika_analysisspecs, "AnalysisSpec", title=
↪"Lab Water Spec", SampleType=sampletype_uid, ResultsRange=rr)
>>> spec_uid = api.get_uid(specification)
```

Create a Reference Definition for blank:

```
>>> blankdef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",␣
↪title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': Au.UID(), 'result': '0', 'min': '0', 'max': '0'},]
>>> blankdef.setReferenceResults(blank_refs)
```

And for control:

```
>>> controldef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': Au.UID(), 'result': '10', 'min': '9.99', 'max': '10.01'},
...                 {'uid': Cu.UID(), 'result': '-0.9','min': '-1.08', 'max': '-0.72'}
↪,]
>>> controldef.setReferenceResults(control_refs)
```

```
>>> blank = api.create(supplier, "ReferenceSample", title="Blank",
...                    ReferenceDefinition=blankdef,
...                    Blank=True, ExpiryDate=date_future,
...                    ReferenceResults=blank_refs)
>>> control = api.create(supplier, "ReferenceSample", title="Control",
...                      ReferenceDefinition=controldef,
...                      Blank=False, ExpiryDate=date_future,
...                      ReferenceResults=control_refs)
```

Create an Analysis Request:

```
>>> values = {
...     'Client': api.get_uid(client),
...     'Contact': api.get_uid(contact),
...     'DateSampled': date_now,
...     'SampleType': sampletype_uid,
...     'Specification': spec_uid,
...     'Priority': '1',
... }
```

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> success = doActionFor(ar, 'receive')
```

Create a new Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> analyses = map(api.get_object, ar.getAnalyses())
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add a duplicate for *Cu*:

```
>>> position = worksheet.get_slot_position(ar, 'a')
>>> duplicates = worksheet.addDuplicateAnalyses(position)
>>> duplicates.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

Add a blank and a control:

```
>>> blanks = worksheet.addReferenceAnalyses(blank, service_uids)
>>> blanks.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
>>> controls = worksheet.addReferenceAnalyses(control, service_uids)
>>> controls.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

4.3.2 Check if results are out of range

First, get the analyses from slot 1 and sort them asc:

```
>>> analyses = worksheet.get_analyses_at(1)
>>> analyses.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

Set results for analysis *Au* (min: -5, max: 5, warn_min: -5.5, warn_max: 5.5):

```
>>> au_analysis = analyses[0]
>>> au_analysis.setResult(2)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> au_analysis.setResult(-2)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> au_analysis.setResult(-5)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> au_analysis.setResult(5)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> au_analysis.setResult(10)
>>> is_out_of_range(au_analysis)
(True, True)
```

```
>>> au_analysis.setResult(-10)
>>> is_out_of_range(au_analysis)
(True, True)
```

Results in shoulders?:

```
>>> au_analysis.setResult(-5.2)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(-5.5)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(-5.6)
>>> is_out_of_range(au_analysis)
(True, True)
```

```
>>> au_analysis.setResult(5.2)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(5.5)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(5.6)
>>> is_out_of_range(au_analysis)
(True, True)
```

4.3.3 Check if results for duplicates are out of range

Get the first duplicate analysis that comes from *Au*:

```
>>> duplicate = duplicates[0]
```

A Duplicate will be considered out of range if its result does not match with the result set to the analysis that was duplicated from, with the Duplicate Variation in % as the margin error. The Duplicate Variation assigned in the Analysis Service *Au* is 0.5%:

```
>>> dup_variation = au_analysis.getDuplicateVariation()
>>> dup_variation = api.to_float(dup_variation)
>>> dup_variation
0.5
```

Set an in-range result (between -5 and 5) for routine analysis and check all variants on it's duplicate. Given that the duplicate variation is 0.5, the valid range for the duplicate must be *Au +-0.5%*:

```
>>> result = 2.0
>>> au_analysis.setResult(result)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> duplicate.setResult(result)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> dup_min_range = result - (result*(dup_variation/100))
>>> duplicate.setResult(dup_min_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_min_range - 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

```
>>> dup_max_range = result + (result*(dup_variation/100))
>>> duplicate.setResult(dup_max_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_max_range + 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

Set an out-of-range result, but within shoulders, for routine analysis and check all variants on it's duplicate. Given that the duplicate variation is 0.5, the valid range for the duplicate must be *Au +-0.5%*:

```
>>> result = 5.5
>>> au_analysis.setResult(result)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> duplicate.setResult(result)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> dup_min_range = result - (result*(dup_variation/100))
>>> duplicate.setResult(dup_min_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_min_range - 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

```
>>> dup_max_range = result + (result*(dup_variation/100))
>>> duplicate.setResult(dup_max_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_max_range + 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

Set an out-of-range and out-of-shoulders result, for routine analysis and check all variants on it's duplicate. Given that the duplicate variation is 0.5, the valid range for the duplicate must be *Au +-0.5%*:

```
>>> result = -7.0
>>> au_analysis.setResult(result)
>>> is_out_of_range(au_analysis)
(True, True)
```

```
>>> duplicate.setResult(result)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> dup_min_range = result - (abs(result)*(dup_variation/100))
>>> duplicate.setResult(dup_min_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_min_range - 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

```
>>> dup_max_range = result + (abs(result)*(dup_variation/100))
>>> duplicate.setResult(dup_max_range)
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(dup_max_range + 0.5)
>>> is_out_of_range(duplicate)
(True, True)
```

4.3.4 Check if results for Reference Analyses (blanks + controls) are out of range

Reference Analyses (controls and blanks) do not use the result ranges defined in the specifications, rather they use the result range defined in the Reference Sample they have been generated from. In turn, the result ranges defined in Reference Samples can be set manually or acquired from the Reference Definition they might be associated with. Another difference from routine analyses is that reference analyses don't expect a valid range, rather a discrete value, so shoulders are built based on % error.

Blank Analyses

The first blank analysis corresponds to *Au*:

```
>>> au_blank = blanks[0]
```

For *Au* blank, as per the reference definition used above, the expected result is 0 +/- 0.1%. Since the expected result is 0, no shoulders will be considered regardless of the % of error. Thus, result will always be "out-of-shoulders" when out of range.

```
>>> au_blank.setResult(0.0)
>>> is_out_of_range(au_blank)
(False, False)
```

```
>>> au_blank.setResult("0")
>>> is_out_of_range(au_blank)
(False, False)
```

```
>>> au_blank.setResult(0.0001)
>>> is_out_of_range(au_blank)
(True, True)
```

```
>>> au_blank.setResult("0.0001")
>>> is_out_of_range(au_blank)
(True, True)
```

```
>>> au_blank.setResult(-0.0001)
>>> is_out_of_range(au_blank)
(True, True)
```

```
>>> au_blank.setResult("-0.0001")
>>> is_out_of_range(au_blank)
(True, True)
```

Control Analyses

The first control analysis corresponds to *Au*:

```
>>> au_control = controls[0]
```

For *Au* control, as per the reference definition used above, the expected result is 10 +/- 0.1% = 10 +/- 0.01

First, check for in-range values:

```
>>> au_control.setResult(10)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult(10.0)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("10")
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("10.0")
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult(9.995)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("9.995")
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult(10.005)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("10.005")
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult(9.99)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("9.99")
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult(10.01)
>>> is_out_of_range(au_control)
(False, False)
```

```
>>> au_control.setResult("10.01")
>>> is_out_of_range(au_control)
(False, False)
```

Now, check for out-of-range results:

```
>>> au_control.setResult(9.98)
>>> is_out_of_range(au_control)
(True, True)
```

```
>>> au_control.setResult("9.98")
>>> is_out_of_range(au_control)
(True, True)
```

```
>>> au_control.setResult(10.011)
>>> is_out_of_range(au_control)
(True, True)
```

```
>>> au_control.setResult("10.011")
>>> is_out_of_range(au_control)
(True, True)
```

And do the same with the control for *Cu* that expects -0.9 +/- 20%:

```
>>> cu_control = controls[1]
```

First, check for in-range values:

```
>>> cu_control.setResult(-0.9)
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult("-0.9")
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult(-1.08)
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult("-1.08")
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult(-1.07)
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult("-1.07")
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult(-0.72)
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult("-0.72")
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult(-0.73)
>>> is_out_of_range(cu_control)
(False, False)
```

```
>>> cu_control.setResult("-0.73")
>>> is_out_of_range(cu_control)
(False, False)
```

Now, check for out-of-range results:

```
>>> cu_control.setResult(0)
>>> is_out_of_range(cu_control)
(True, True)
```

```
>>> cu_control.setResult("0")
>>> is_out_of_range(cu_control)
(True, True)
```

```
>>> cu_control.setResult(-0.71)
>>> is_out_of_range(cu_control)
(True, True)
```

```
>>> cu_control.setResult("-0.71")
>>> is_out_of_range(cu_control)
(True, True)
```

```
>>> cu_control.setResult(-1.09)
>>> is_out_of_range(cu_control)
(True, True)
```

```
>>> cu_control.setResult("-1.09")
>>> is_out_of_range(cu_control)
(True, True)
```

4.3.5 Check if results are out of range when open interval is used

Set open interval for min and max from water specification

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'gt'
...     range['max_operator'] = 'lt'
>>> specification.setResultsRange(ranges)
```

We need to re-apply the Specification for the changes to take effect:

```
>>> ar.setSpecification(None)
>>> ar.setSpecification(specification)
```

First, get the analyses from slot 1 and sort them asc:

```
>>> analyses = worksheet.get_analyses_at(1)
>>> analyses.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

Set results for analysis *Au* (min: -5, max: 5, warn_min: -5.5, warn_max: 5.5):

```
>>> au_analysis = analyses[0]
>>> au_analysis.setResult(-5)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(5)
>>> is_out_of_range(au_analysis)
(True, False)
```

4.3.6 Check if results are out of range when left-open interval is used

Set left-open interval for min and max from water specification

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'geq'
...     range['max_operator'] = 'lt'
>>> specification.setResultsRange(ranges)
```

We need to re-apply the Specification for the changes to take effect:

```
>>> ar.setSpecification(None)
>>> ar.setSpecification(specification)
```

First, get the analyses from slot 1 and sort them asc:

```
>>> analyses = worksheet.get_analyses_at(1)
>>> analyses.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

Set results for analysis *Au* (min: -5, max: 5, warn_min: -5.5, warn_max: 5.5):

```
>>> au_analysis = analyses[0]
>>> au_analysis.setResult(-5)
>>> is_out_of_range(au_analysis)
(False, False)
```

```
>>> au_analysis.setResult(5)
>>> is_out_of_range(au_analysis)
(True, False)
```

4.3.7 Check if results are out of range when right-open interval is used

Set right-open interval for min and max from water specification

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'gt'
...     range['max_operator'] = 'leq'
>>> specification.setResultsRange(ranges)
```

We need to re-apply the Specification for the changes to take effect:

```
>>> ar.setSpecification(None)
>>> ar.setSpecification(specification)
```

First, get the analyses from slot 1 and sort them asc:

```
>>> analyses = worksheet.get_analyses_at(1)
>>> analyses.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
```

Set results for analysis *Au* (min: -5, max: 5, warn_min: -5.5, warn_max: 5.5):

```
>>> au_analysis = analyses[0]
>>> au_analysis.setResult(-5)
>>> is_out_of_range(au_analysis)
(True, False)
```

```
>>> au_analysis.setResult(5)
>>> is_out_of_range(au_analysis)
(False, False)
```

4.3.8 Check if formatted interval is rendered properly

Set closed interval for min and max from water specification

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'geq'
...     range['max_operator'] = 'leq'
>>> specification.setResultsRange(ranges)
```

Get the result range for *Au* (min: -5, max: 5)

```
>>> rr = specification.getResultsRange()
>>> res_range = filter(lambda item: item.get('keyword') == 'Au', rr)[0]
>>> get_formatted_interval(res_range)
'[-5;5]'
```

Try now with left-open interval

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'gt'
...     range['max_operator'] = 'leq'
>>> specification.setResultsRange(ranges)
```

Get the result range for *Au* (min: -5, max: 5)

```
>>> rr = specification.getResultsRange()
>>> res_range = filter(lambda item: item.get('keyword') == 'Au', rr)[0]
>>> get_formatted_interval(res_range)
'(-5;5]'
```

Try now with right-open interval

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'geq'
...     range['max_operator'] = 'lt'
>>> specification.setResultsRange(ranges)
```

Get the result range for *Au* (min: -5, max: 5)

```
>>> rr = specification.getResultsRange()
>>> res_range = filter(lambda item: item.get('keyword') == 'Au', rr)[0]
>>> get_formatted_interval(res_range)
'[-5;5)'
```

Try now with open interval

```
>>> ranges = specification.getResultsRange()
>>> for range in ranges:
...     range['min_operator'] = 'gt'
...     range['max_operator'] = 'lt'
>>> specification.setResultsRange(ranges)
```

Get the result range for *Au* (min: -5, max: 5)

```
>>> rr = specification.getResultsRange()
>>> res_range = filter(lambda item: item.get('keyword') == 'Au', rr)[0]
>>> get_formatted_interval(res_range)
'(-5;5)'
```

And if we set a 0 value as min or max?

```
>>> res_range['min'] = 0
>>> get_formatted_interval(res_range)
'(0;5)'
```

```
>>> res_range['max'] = 0
>>> res_range['min'] = -5
>>> get_formatted_interval(res_range)
'(-5;0)'
```

And now, set no value for min and/or max

```
>>> res_range['min'] = ''
>>> res_range['max'] = 5
>>> get_formatted_interval(res_range)
'<5'
```

```
>>> res_range['max'] = ''
>>> res_range['min'] = -5
>>> get_formatted_interval(res_range)
'>-5'
```

And change the operators

```
>>> res_range['min'] = ''
>>> res_range['max'] = 5
>>> res_range['max_operator'] = 'leq'
>>> get_formatted_interval(res_range)
'<=5'
```

```
>>> res_range['max'] = ''
>>> res_range['min'] = -5
>>> res_range['max_operator'] = 'lt'
>>> res_range['min_operator'] = 'geq'
>>> get_formatted_interval(res_range)
'>=-5'
```

4.4 API Analysis Service

The *api_analysisservice* modue provides single functions for single purposes especifically related with analyses services.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_AnalysisService
```

4.4.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.api.analysisservice import get_calculation_dependencies_for
>>> from bika.lims.api.analysisservice import get_calculation_dependants_for
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> calculations = setup.bika_calculations
>>> analysisservices = setup.bika_analysisservices
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.4.2 Calculation Dependencies

Calculations can reference analysis services by *Keyword* in their *Formula*.

The referenced Analysis Services of the calculation are then dependencies of the Analysis Service which has the Calculation assigned.

The dependencies of an Analysis Service can be retrieved by the API function *get_calculation_dependencies_for*.

Create some Analysis Services with unique Keywords:

```
>>> Ca = api.create(analysisservices, "AnalysisService", title="Calcium", Keyword="Ca
↪")
>>> Mg = api.create(analysisservices, "AnalysisService", title="Magnesium", Keyword=
↪"Mg")
>>> Cu = api.create(analysisservices, "AnalysisService", title="Copper", Keyword="Cu")
>>> Fe = api.create(analysisservices, "AnalysisService", title="Iron", Keyword="Fe")
>>> Au = api.create(analysisservices, "AnalysisService", title="Aurum", Keyword="Au")
>>> Test1 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 1", Keyword="Test1")
>>> Test2 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 2", Keyword="Test2")
```

None of these services has so far any calculation dependencies:

```
>>> any(map(get_calculation_dependencies_for, [Ca, Mg, Cu, Fe, Au, Test1, Test2]))
False
```

Create a calculation, which references the *Ca* and *Mg* services, and link the calculation to the *Test1* service:

```
>>> calc1 = api.create(calculations, "Calculation", title="Calculation 1")
>>> calc1.setFormula("[Ca] + [Mg]")
>>> Test1.setCalculation(calc1)
```

The *Test1* service depends now on *Ca* and *Mg*:

```
>>> deps = get_calculation_dependencies_for(Test1)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Ca', 'Mg']
```

Now we add *Fe* to the calculation:

```
>>> calc1.setFormula("[Ca] + [Mg] + [Fe]")
```

The *Test1* service depends now on *Fe* as well:

```
>>> deps = get_calculation_dependencies_for(Test1)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Ca', 'Fe', 'Mg']
```

Now we create a calculation which doubles the results of the calculated *Test1* service and assign it to the *Test2* service:

```
>>> calc2 = api.create(calculations, "Calculation", title="Calculation 2")
>>> calc2.setFormula("[Test1] * 2")
>>> Test2.setCalculation(calc2)
```

The *Test2* service depends now on the *Test1* service:

```
>>> deps = get_calculation_dependencies_for(Test2)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Test1']
```

4.4.3 Calculation Dependants

To get all Analysis Services which depend on a specific Analysis Service, the API provides the function *get_calculation_dependants_for*.

The Analysis Service *Test1* references *Ca*, *Mg* and *Fe* by its calculation:

```
>>> Test1.getCalculation().getFormula()
'[Ca] + [Mg] + [Fe]'
```

Therefore, the dependant service of *Ca*, *Mg* and *Fe* is *Test1*

```
>>> deps = get_calculation_dependants_for(Ca)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Test1']
```

```
>>> deps = get_calculation_dependants_for(Mg)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Test1']
```

```
>>> deps = get_calculation_dependants_for(Fe)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Test1']
```

The Analysis Service *Test2* doubles the calculated result from *Test1*:

```
>>> Test2.getCalculation().getFormula()
'[Test1] * 2'
```

Therefore, *Test2* is a dependant of *Test1*:

```
>>> deps = get_calculation_dependants_for(Test1)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
['Test2']
```

4.4.4 Checking edge cases

The assigned calculation of *Test2* doubles the value of *Test1*:

```
>>> Test2.getCalculation().getFormula()
'[Test1] * 2'
```

But what happens when the calculation references *Test2* as well?

```
>>> Test2.getCalculation().setFormula("[Test1] * [Test2]")
>>> Test2.getCalculation().getFormula()
'[Test1] * [Test2]'
```

Checking the dependants of *Test2* should not cause an infinite recursion:

```
>>> deps = get_calculation_dependants_for(Test2)
>>> sorted(map(lambda d: d.getKeyword(), deps.values()))
[]
```

4.5 API for sending emails

The mail API provides a simple interface to send emails in SENAITE.

NOTE: The API is called *mail* to avoid import conflicts with the Python *email* standard library.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_mail
```

4.5.1 Test Setup

Imports:

```
>>> import os
>>> from __future__ import print_function
```

```
>>> from bika.lims.api.mail import *
```

Variables:

```
>>> cur_dir = os.path.dirname(__file__)
>>> filename = "logo.png"
>>> filepath = os.path.join(cur_dir, filename)
```

4.5.2 Email Address

This function converts an email address and name pair to a string value suitable for an RFC 2822 *From*, *To* or *Cc* header:

```
>>> to_address = to_email_address("rb@ridingbytes.com", "Ramon Bartl")
```

```
>>> to_address
'Ramon Bartl <rb@ridingbytes.com>'
```

```
>>> to_email_address("rb@ridingbytes.com")
'rb@ridingbytes.com'
```

4.5.3 Email Subject

This function converts a string to a compliant RFC 2822 subject header:

```
>>> subject = u"Liberté"
>>> email_subject = to_email_subject(subject)
```

```
>>> email_subject
<email.header.Header instance at ...>
```

```
>>> print(email_subject)
=?utf-8?q?Libert=C3=83=C2=A9?=
```

4.5.4 Email Body Text

This function coverts a given text to a text/plain MIME document:

```
>>> text = "Check out SENAITE LIMS: $url"
>>> email_body = to_email_body_text(text, url="https://www.senaite.com")
```

```
>>> email_body
<email.mime.text.MIMEText instance at ...>
```

```
>>> print(email_body)
From ...
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
<BLANKLINE>
Check out SENAITE LIMS: https://www.senaite.com
```

4.5.5 Email Attachment

This function converts a filename with given filedata to a MIME attachment:

```
>>> attachment1 = to_email_attachment(file(filepath), filename=filename)
>>> attachment1
<email.mime.base.MIMEBase instance at ...>
```

```
>>> print(attachment1)
From ...
Content-Type: image/png
MIME-Version: 1.0
Content-Transfer-Encoding: base64
```

(continues on next page)

(continued from previous page)

```
Content-Disposition: attachment; filename=logo.png
<BLANKLINE>
iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAABGdBTUEAALGPC/xhBQAAACBjSFJN
...
5/sfV5M/kISv300AAAAASUVORK5CYII=
```

It is also possible to provide the full path to a file:

```
>>> attachment2 = to_email_attachment(filepath)
>>> attachment2
<email.mime.base.MIMEBase instance at ...>
```

```
>>> print(attachment2)
From ...
Content-Type: image/png
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=logo.png
<BLANKLINE>
iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAABGdBTUEAALGPC/xhBQAAACBjSFJN
...
5/sfV5M/kISv300AAAAASUVORK5CYII=
```

Providing an attachment works as well:

```
>>> attachment3 = to_email_attachment(attachment2)
>>> attachment3 == attachment2
True
```

4.5.6 Email Address Validation

This function checks if the given email address is valid:

```
>>> is_valid_email_address("rb@ridingbytes.com")
True
```

```
>>> is_valid_email_address(u"rb@ridingbytes.de")
True
```

```
>>> is_valid_email_address("rb@ridingbytes")
False
```

```
>>> is_valid_email_address("@ridingbyte.com")
False
```

```
>>> is_valid_email_address("rb")
False
```

```
>>> is_valid_email_address(None)
False
```

```
>>> is_valid_email_address(object())
False
```

4.5.7 Parse Email Address

This function parses an email address string into a (name, email) tuple:

```
>>> parse_email_address("Ramon Bartl <rb@ridingbytes.com>")
('Ramon Bartl', 'rb@ridingbytes.com')
```

```
>>> parse_email_address("<rb@ridingbytes.com>")
('', 'rb@ridingbytes.com')
```

```
>>> parse_email_address("rb@ridingbytes.com")
('', 'rb@ridingbytes.com')
```

4.5.8 Compose Email

This function composes a new MIME message:

```
>>> message = compose_email("from@senaite.com",
...                         ["to@senaite.com", "to2@senaite.com"],
...                         "Test Émail",
...                         "Check out the new SENAITE website: $url",
...                         attachments=[filepath],
...                         url="https://www.senaite.com")
```

```
>>> message
<email.mime.multipart.MIMEMultipart instance at ...>
```

```
>>> print(message)
From ...
Content-Type: multipart/mixed; boundary="..."
MIME-Version: 1.0
Subject: =?utf-8?q?Test_=C3=89mail?=
From: from@senaite.com
To: to@senaite.com, to2@senaite.com
<BLANKLINE>
This is a multi-part message in MIME format.
<BLANKLINE>
...
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
<BLANKLINE>
Check out the new SENAITE website: https://www.senaite.com
...
Content-Type: image/png
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=logo.png
<BLANKLINE>
```

(continues on next page)

(continued from previous page)

```
iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAABGdBTUEAALGPC/xhBQAAACBjSFJN
...
5/sfV5M/kISv300AAAAASUVORK5CYII=
...
<BLANKLINE>
```

4.6 API Security

The security API provides a simple interface to control access in SENAITE

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_security
```

4.6.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.api.security import *
>>> from bika.lims.permissions import FieldEditAnalysisHidden
>>> from bika.lims.permissions import FieldEditAnalysisResult
>>> from bika.lims.permissions import FieldEditAnalysisRemarks
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_sample(services):
...     values = {
...         "Client": client.UID(),
...         "Contact": contact.UID(),
...         "DateSampled": date_now,
...         "SampleType": sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     return create_analysisrequest(client, request, values, service_uids)
```

```
>>> def get_analysis(sample, id):
...     ans = sample.getAnalyses(getId=id, full_objects=True)
...     if len(ans) != 1:
...         return None
...     return ans[0]
```

4.6.2 Environment Setup

Setup the testing environment:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> setRoles(portal, TEST_USER_ID, ['LabManager', ])
>>> user = api.get_current_user()
```

4.6.3 LIMS Setup

Setup the Lab for testing:

```
>>> setup.setSelfVerificationEnabled(True)
>>> analysisservices = setup.bika_analysisservices
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH")
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="Water")
```

4.6.4 Content Setup

Create some Analysis Services with unique Keywords:

```
>>> Ca = api.create(analysisservices, "AnalysisService", title="Calcium", Keyword="Ca
↪")
>>> Mg = api.create(analysisservices, "AnalysisService", title="Magnesium", Keyword=
↪"Mg")
>>> Cu = api.create(analysisservices, "AnalysisService", title="Copper", Keyword="Cu")
>>> Fe = api.create(analysisservices, "AnalysisService", title="Iron", Keyword="Fe")
>>> Au = api.create(analysisservices, "AnalysisService", title="Aurum", Keyword="Au")
>>> Test1 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 1", Keyword="Test1")
>>> Test2 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 2", Keyword="Test2")
```

Create an new Sample:

```
>>> sample = new_sample([Cu, Fe, Au])
```

Get the contained *Cu* Analysis:

```
>>> cu = get_analysis(sample, Cu.getKeyword())
```

4.6.5 Get a security manager for the current thread

A security manager provides methods for checking access and managing executable context and policies:

```
>>> get_security_manager()
<AccessControl.ImplPython.SecurityManager instance at ...>
```

4.6.6 Get the possible permissions of an object

The possible permissions include the permissions on the object and the inherited permissions:

```
>>> possible_permissions = get_possible_permissions_for(cu)
>>> "Modify portal content" in possible_permissions
True
```

4.6.7 Get the mapped permissions of an object

While the possible permissions return *all* possible permissions of the object, only few of them are mapped to the object.

The function *get_mapped_permissions_for* returns only those permissions which have roles mapped on the given object or on objects within the acquisition chain.

```
>>> mapped_permissions = get_mapped_permissions_for(cu)
```

The mapped permissions are therefore a subset of the possible transitions:

```
>>> set(mapped_permissions).issubset(possible_permissions)
True
```

4.6.8 Get the granted permissions

This function returns the allowed permissions on an object for a user:

```
>>> allowed_permissions = get_allowed_permissions_for(cu)
```

The allowed permissions is a subset of the mapped permissions:

```
>>> set(allowed_permissions).issubset(mapped_permissions)
True
```

4.6.9 Get the non-granted permissions

This function returns the disallowed permissions on an object for a user:

```
>>> disallowed_permissions = get_disallowed_permissions_for(cu)
```

The disallowed permissions is a subset of the mapped permissions:

```
>>> set(disallowed_permissions).issubset(mapped_permissions)
True
```

It is mutual exclusive to the allowed permissions:

```
>>> set(disallowed_permissions).isdisjoint(allowed_permissions)
True
```

The allowed and disallowed permissions are exactly the mapped permissions:

```
>>> set(allowed_permissions + disallowed_permissions) == set(mapped_permissions)
True
```

4.6.10 Check if a user has a permission granted

This function checks if the user has a permission granted on an object:

```
>>> check_permission(get_allowed_permissions_for(cu)[0], cu)
True
```

```
>>> check_permission(get_disallowed_permissions_for(cu)[0], cu)
False
```

Non existing permissions are returned as False:

```
>>> check_permission("nonexisting_permission", cu)
False
```

4.6.11 Get the granted permissions of a role

This function returns the permissions that are granted to a role:

```
>>> get_permissions_for_role("Sampler", cu)
['senaite.core: Field: Edit Analysis Remarks', 'senaite.core: Field: Edit Analysis␣
↪Result']
```

4.6.12 Get the mapped roles of a permission

This function is the opposite of *get_permissions_for_role* and returns the roles for a given permission:

```
>>> get_roles_for_permission(FieldEditAnalysisResult, cu)
('Manager', 'Sampler')
```

4.6.13 Get the roles of a user

This function returns the global roles the user has:

```
>>> get_roles()
['Authenticated', 'LabManager']
```

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Sampler', ])
```

```
>>> get_roles()
['Authenticated', 'LabManager', 'Sampler']
```

The optional *user* parameter allows to get the roles of another user:

```
>>> get_roles("admin")
['Authenticated', 'Manager']
```

4.6.14 Get the local roles of a user

This function returns the local granted roles the user has for the given object:

```
>>> get_local_roles_for(cu)
['Owner']
```

The optional *user* parameter allows to get the local roles of another user:

```
>>> get_local_roles_for(cu, "admin")
[]
```

4.6.15 Granting local roles

This function allows to grant local roles on an object:

```
>>> grant_local_roles_for(cu, "Sampler")
['Owner', 'Sampler']
```

```
>>> grant_local_roles_for(cu, ["Analyst", "LabClerk"])
['Analyst', 'LabClerk', 'Owner', 'Sampler']
```

```
>>> get_local_roles_for(cu)
['Analyst', 'LabClerk', 'Owner', 'Sampler']
```

4.6.16 Revoking local roles

This function allows to revoke local roles on an object:

```
>>> revoke_local_roles_for(cu, "Sampler")
['Analyst', 'LabClerk', 'Owner']
```

```
>>> revoke_local_roles_for(cu, ["Analyst", "LabClerk"])
['Owner']
```

```
>>> get_local_roles_for(cu)
['Owner']
```

4.6.17 Getting all valid roles

This function lists all valid roles for an object:

```
>>> get_valid_roles_for(cu)
['Analyst', ...]
```

4.6.18 Granting a permission to a role

This function allows to grant a permission to one or more roles:

```
>>> get_permissions_for_role("Sampler", cu)
['senaite.core: Field: Edit Analysis Remarks', 'senaite.core: Field: Edit Analysis␣
↪Result']
```

```
>>> grant_permission_for(cu, FieldEditAnalysisHidden, "Sampler", acquire=0)
```

```
>>> get_permissions_for_role("Sampler", cu)
['senaite.core: Field: Edit Analysis Hidden', 'senaite.core: Field: Edit Analysis␣
↪Remarks', 'senaite.core: Field: Edit Analysis Result']
```

4.6.19 Revoking a permission from a role

This function allows to revoke a permission of one or more roles:

```
>>> revoke_permission_for(cu, FieldEditAnalysisHidden, "Sampler", acquire=0)
```

```
>>> get_permissions_for_role("Sampler", cu)
['senaite.core: Field: Edit Analysis Remarks', 'senaite.core: Field: Edit Analysis␣
↪Result']
```

4.6.20 Manage permissions

This function allows to set a permission explicitly to the given roles (drop other roles):

```
>>> grant_permission_for(cu, FieldEditAnalysisResult, ["Analyst", "LabClerk"])
```

```
>>> get_permissions_for_role("Analyst", cu)
['senaite.core: Field: Edit Analysis Result']
```

```
>>> get_permissions_for_role("LabClerk", cu)
['senaite.core: Field: Edit Analysis Result']
```

Now we use *manage_permission_for* to grant this permission *only* for Samplers:

```
>>> manage_permission_for(cu, FieldEditAnalysisResult, ["Sampler"])
```

The Sampler has now the permission granted:

```
>>> get_permissions_for_role("Sampler", cu)
['senaite.core: Field: Edit Analysis Remarks', 'senaite.core: Field: Edit Analysis␣
↪Result']
```

But the Analyst and LabClerk not anymore:

```
>>> get_permissions_for_role("Analyst", cu)
[]
```

```
>>> get_permissions_for_role("LabClerk", cu)
[]
```

4.7 API Snapshot

The snapshot API provides a simple interface to manage object snaphots.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_snapshot
```

4.7.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.api.snapshot import *
>>> from bika.lims.permissions import FieldEditAnalysisHidden
>>> from bika.lims.permissions import FieldEditAnalysisResult
>>> from bika.lims.permissions import FieldEditAnalysisRemarks
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_sample(services):
...     values = {
...         "Client": client.UID(),
...         "Contact": contact.UID(),
...         "DateSampled": date_now,
...         "SampleType": sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     return create_analysisrequest(client, request, values, service_uids)
```

```
>>> def get_analysis(sample, id):
...     ans = sample.getAnalyses(getId=id, full_objects=True)
...     if len(ans) != 1:
...         return None
...     return ans[0]
```

4.7.2 Environment Setup

Setup the testing environment:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> setRoles(portal, TEST_USER_ID, ['LabManager', ])
>>> user = api.get_current_user()
```

4.7.3 LIMS Setup

Setup the Lab for testing:

```
>>> setup.setSelfVerificationEnabled(True)
>>> analysisservices = setup.bika_analysisservices
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH")
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="Water")
```

4.7.4 Content Setup

Create some Analysis Services with unique Keywords:

```
>>> Ca = api.create(analysisservices, "AnalysisService", title="Calcium", Keyword="Ca
↪")
>>> Mg = api.create(analysisservices, "AnalysisService", title="Magnesium", Keyword=
↪"Mg")
>>> Cu = api.create(analysisservices, "AnalysisService", title="Copper", Keyword="Cu")
>>> Fe = api.create(analysisservices, "AnalysisService", title="Iron", Keyword="Fe")
>>> Au = api.create(analysisservices, "AnalysisService", title="Aurum", Keyword="Au")
>>> Test1 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 1", Keyword="Test1")
>>> Test2 = api.create(analysisservices, "AnalysisService", title="Calculated Test␣
↪Service 2", Keyword="Test2")
```

Create an new Sample:

```
>>> sample = new_sample([Cu, Fe, Au])
```

Get the contained *Cu* Analysis:

```
>>> cu = get_analysis(sample, Cu.getKeyword())
>>> fe = get_analysis(sample, Fe.getKeyword())
>>> au = get_analysis(sample, Au.getKeyword())
```

4.7.5 Check if an object supports snapshots

We can use the *support_snapshots* function to check if the object supports snapshots:

```
>>> supports_snapshots(sample)
True
```

```
>>> supports_snapshots(object())
False
```

4.7.6 Get the snapshot storage

The snapshot storage holds all the raw snapshots in JSON format:

```
>>> storage = get_storage(sample)
>>> storage
['{...}']
```

4.7.7 Get all snapshots

To get the data snapshots of an object, we can call *get_snapshots*:

```
>>> snapshots = get_snapshots(sample)
>>> snapshots
[{...}]
```

4.7.8 Check if an object has snapshots

To check if an object has snapshots, we can call *has_snapshots*:

```
>>> has_snapshots(sample)
True
```

```
>>> has_snapshots(cu)
True
```

```
>>> has_snapshots(fe)
True
```

```
>>> has_snapshots(au)
True
```

```
>>> has_snapshots(setup)
False
```

4.7.9 Get the number of snapshots

To check the number of snapshots (versions) an object has, we can call *get_snapshot_count*:

```
>>> get_snapshot_count(sample)
2
```

```
>>> get_snapshot_count(setup)
0
```

4.7.10 Get the version of an object

If an object has a snapshot, it is considered as version 0:

```
>>> get_version(cu)
0
```

If the object does not have any snapshots yet, this function returns -1:

```
>>> get_version(object())
-1
```

4.7.11 Get a snapshot by version

Snapshots can be retrieved by their index in the snapshot storage (version):

```
>>> get_snapshot_by_version(sample, 0)
{...}
```

Negative versions return *None*:

```
>>> get_snapshot_by_version(sample, -1)
```

Non existing versions return *None*:

```
>>> get_snapshot_by_version(sample, 9999)
```

4.7.12 Get the version of a snapshot

The index (version) of each snapshot can be retrieved:

```
>>> snap1 = get_snapshot_by_version(sample, 0)
>>> get_snapshot_version(sample, snap1)
0
```

```
>>> snap2 = get_snapshot_by_version(sample, 1)
>>> get_snapshot_version(sample, snap2)
1
```

4.7.13 Get the last snapshot taken

To get the latest snapshot, we can call *get_last_snapshot*:

```
>>> snap = get_last_snapshot(sample)
>>> get_snapshot_version(sample, snap)
1
```

4.7.14 Get the metadata of a snapshot

Each snapshot contains metadata which can be retrieved:

```
>>> metadata = get_snapshot_metadata(snap)
>>> metadata
{...}
```

The metadata holds the information about the performing user etc.:

```
>>> metadata.get("actor")
u'test_user_1_'
```

```
>>> metadata.get("roles")
[u'Authenticated', u'LabManager']
```

4.7.15 Take a new Snapshot

Snapshots can be taken programatically with the function *take_snapshot*:

```
>>> get_version(sample)
1
```

Now we take a new snapshot:

```
>>> snapshot = take_snapshot(sample)
```

The version should be increased:

```
>>> get_version(sample)
2
```

The new snapshot should be the most recent snapshot now:

```
>>> last_snapshot = get_last_snapshot(sample)
```

```
>>> last_snapshot == snapshot
True
```

4.7.16 Comparing Snapshots

The changes of two snapshots can be compared with *compare_snapshots*:

```
>>> snap0 = get_snapshot_by_version(sample, 2)
```

Add 2 more analyses (Mg and Ca):

```
>>> sample.edit(Analyses=[Cu, Fe, Au, Mg, Ca])
>>> new_snapshot = take_snapshot(sample)
>>> snap1 = get_snapshot_by_version(sample, 3)
```

Passing the *raw=True* keyword returns the raw field changes, e.g. in this case, the field *Analyses* is a *UIDReferenceField* which contained initially 3 values and after adding 2 analyses, 2 UID more references:

```
>>> diff_raw = compare_snapshots(snap0, snap1, raw=True)
>>> diff_raw
{u'Analyses': [([u'...', u'...', u'...'], [u'...', u'...', u'...', u'...', u'...'])]}
```

It is also possible to process the values to get a more human readable diff:

```
>>> diff = compare_snapshots(snap0, snap1, raw=False)
>>> diff
{u'Analyses': [('Aurum; Copper; Iron', 'Aurum; Calcium; Copper; Iron; Magnesium')]}
```

To directly compare the last two snapshots taken, we can call *compare_last_two_snapshots*.

First we edit the sample to get a new snapshot:

```
>>> sample.edit(CCEmails="rb@ridingbytes.com")
>>> snapshot = take_snapshot(sample)
```

```
>>> last_diff = compare_last_two_snapshots(sample, raw=False)
>>> last_diff
{u'CCEmails': [('Not set', 'rb@ridingbytes.com')]}
```

4.8 API User

The user API provides a simple interface to control users and groups in SENAITE

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t API_user
```

4.8.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.api.user import *
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

4.8.2 Environment Setup

Setup the testing environment:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager', ])
>>> user = api.get_current_user()
```

4.8.3 Get user

Get the user object (not the memberdata wrapped user):

```
>>> current_user = get_user()
>>> current_user
<PloneUser 'test-user'>
```

This function takes also an optional *user* argument:

```
>>> other_user = get_user("admin")
>>> other_user
<PropertiedUser 'admin'>
```

It can also take the user object:

```
>>> other_user = get_user(other_user)
>>> other_user
<PropertiedUser 'admin'>
```

Or a *MemberData* object:

```
>>> member = api.get_user(TEST_USER_ID)
>>> member
<Products.PlonePAS.tools.memberdata.MemberData object at ...>
```

```
>>> get_user(member)
<PloneUser 'test-user'>
```

It returns *None* if the user was not found:

```
>>> get_user("nonexistant") is None
True
```

4.8.4 Get user ID

The user ID can be retrieved by the same objects as the *get_user* function:

```
>>> current_user_id = get_user_id()
>>> current_user_id
'test_user_1_'
```

It takes also the optional *user* argument:

```
>>> get_user_id(TEST_USER_ID)
'test_user_1_'
```

It can also take the user object:

```
>>> current_user = get_user()
>>> get_user_id(current_user)
'test_user_1_'
```

If the user was not found, it returns *None*:

```
>>> get_user_id("nonexistant") is None
True
```

4.8.5 Get user groups

This function returns the groups the user belongs to:

```
>>> get_groups()
['AuthenticatedUsers']
```

It takes also the optional *user* argument:

```
>>> get_groups('admin')
['AuthenticatedUsers']
```

4.8.6 Get group

This function returns a group object:

```
>>> get_group('Analysts')
<GroupData at /plone/portal_groupdata/Analysts used for /plone/acl_users/source_
↪groups>
```

It returns *None* if the group was not found:

```
>>> get_group('noexistant') is None
True
```

If the group is *None*, all groups are returned:

```
>>> get_group(None) is None
True
```

4.8.7 Add group

This function adds users to group(s):

```
>>> add_group("Analysts")
['AuthenticatedUsers', 'Analysts']
```

It takes also an optinal *user* parameter to add another user to a group:

```
>>> add_group("LabManagers", "admin")
['AuthenticatedUsers', 'LabManagers']
```

Also adding a user to multiple groups are allowed:

```
>>> add_group(["Analyst", "Samplers", "Publishers"], "admin")
['Publishers', 'Samplers', 'LabManagers', 'AuthenticatedUsers']
```

4.8.8 Delete group

This function removes users from group(s):

```
>>> get_groups()
['AuthenticatedUsers', 'Analysts']
```

```
>>> del_group("Analysts")
['AuthenticatedUsers']
```

Also removing a user from multiple groups is allowed:

```
>>> get_groups("admin")
['Publishers', 'Samplers', 'LabManagers', 'AuthenticatedUsers']
```

```
>>> del_group(["Publishers", "Samplers", "LabManagers"], "admin")
['AuthenticatedUsers']
```

4.9 AR Analyses Field

This field manages Analyses for Analysis Requests.

It is capable to perform the following tasks:

- Create Analyses from Analysis Services
- Delete assigned Analyses
- Update Prices of assigned Analyses
- Update Specifications of assigned Analyses
- Update Interim Fields of assigned Analyses

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t ARAnalysesField
```

4.9.1 Test Setup

Imports:

```
>>> import transaction
>>> from operator import methodcaller
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
```

```
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def get_analyses_from(sample, services):
...     if not isinstance(services, (list, tuple)):
...         services = [services]
...     uids = map(api.get_uid, services)
...     analyses = sample.getAnalyses(full_objects=True)
...     return filter(lambda an: an.getServiceUID() in uids, analyses)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> calculations = setup.bika_calculations
>>> sampletypes = setup.bika_sampletypes
>>> samplepoints = setup.bika_samplepoints
>>> analysiscategories = setup.bika_analysiscategories
>>> analysisspecs = setup.bika_analysisspecs
>>> analysisservices = setup.bika_analysisservices
>>> labcontacts = setup.bika_labcontacts
>>> worksheets = setup.worksheets
>>> storagelocations = setup.bika_storagelocations
>>> samplingdeviations = setup.bika_samplingdeviations
>>> sampleconditions = setup.bika_sampleconditions
>>> portal_url = portal.absolute_url()
>>> setup_url = portal_url + "/bika_setup"
```

Test User:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.9.2 Prepare Test Environment

Create Client:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="Happy Hills", ClientID="HH")
>>> client
<Client at /plone/clients/client-1>
```

Create some Contact(s):

```
>>> contact1 = api.create(client, "Contact", Firstname="Client", Surname="One")
>>> contact1
<Contact at /plone/clients/client-1/contact-1>
```

```
>>> contact2 = api.create(client, "Contact", Firstname="Client", Surname="Two")
>>> contact2
<Contact at /plone/clients/client-1/contact-2>
```

Create a Sample Type:

```
>>> sampletype = api.create(sampletypes, "SampleType", Prefix="water", MinimumVolume=
↪"100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

Create a Sample Point:

```
>>> samplepoint = api.create(samplepoints, "SamplePoint", title="Lake Python")
>>> samplepoint
<SamplePoint at /plone/bika_setup/bika_samplepoints/samplepoint-1>
```

Create an Analysis Category:

```
>>> analysiscategory = api.create(analysiscategories, "AnalysisCategory", title="Water
↪")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
```

Create Analysis Service for PH (Keyword: *PH*):

```
>>> analysisservice1 = api.create(analysisservices, "AnalysisService", title="PH",␣
↪ShortTitle="ph", Category=analysiscategory, Keyword="PH", Price="10")
>>> analysisservice1
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
```

Create Analysis Service for Magnesium (Keyword: *MG*):

```
>>> analysisservice2 = api.create(analysisservices, "AnalysisService", title=
↪"Magnesium", ShortTitle="mg", Category=analysiscategory, Keyword="MG", Price="20")
>>> analysisservice2
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>
```

Create Analysis Service for Calcium (Keyword: *CA*):

```
>>> analysisservice3 = api.create(analysisservices, "AnalysisService", title="Calcium
↪", ShortTitle="ca", Category=analysiscategory, Keyword="CA", Price="30")
>>> analysisservice3
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-3>
```

Create Analysis Service for Total Hardness (Keyword: *THCaCO3*):

```
>>> analysisservice4 = api.create(analysisservices, "AnalysisService", title="Total␣
↪Hardness", ShortTitle="Tot. Hard", Category=analysiscategory, Keyword="THCaCO3",␣
↪Price="40")
>>> analysisservice4
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-4>
```

Create Analysis Service w/o calculation (Keyword: *NOCALC*):

```
>>> analysisservice5 = api.create(analysisservices, "AnalysisService", title="No␣
↪Calculation", ShortTitle="nocalc", Category=analysiscategory, Keyword="NoCalc",␣
↪Price="50")
>>> analysisservice5
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-5>
```

Create some Calculations with Formulas referencing existing AS keywords:

```
>>> calc1 = api.create(calculations, "Calculation", title="Round")
>>> calc1.setFormula("round(12345, 2)")
```

```
>>> calc2 = api.create(calculations, "Calculation", title="A in ppt")
>>> calc2.setFormula("[A] * 1000")
```

```
>>> calc3 = api.create(calculations, "Calculation", title="B in ppt")
>>> calc3.setFormula("[B] * 1000")
```

```
>>> calc4 = api.create(calculations, "Calculation", title="Total Hardness")
>>> calc4.setFormula("[CA] + [MG]")
```

Assign the calculations to the Analysis Services:

```
>>> analysisservice1.setCalculation(calc1)
>>> analysisservice2.setCalculation(calc2)
>>> analysisservice3.setCalculation(calc3)
>>> analysisservice4.setCalculation(calc4)
```

Create an Analysis Specification for *Water*:

```
>>> sampletype_uid = api.get_uid(sampletype)
```

```
>>> rr1 = {"keyword": "PH", "min": 5, "max": 7, "error": 10, "hidemin": "", "hidemax
↪": "", "rangecomment": "Lab PH Spec"}
>>> rr2 = {"keyword": "MG", "min": 5, "max": 7, "error": 10, "hidemin": "", "hidemax
↪": "", "rangecomment": "Lab MG Spec"}
>>> rr3 = {"keyword": "CA", "min": 5, "max": 7, "error": 10, "hidemin": "", "hidemax
↪": "", "rangecomment": "Lab CA Spec"}
>>> rr = [rr1, rr2, rr3]
```

```
>>> analysisspec1 = api.create(analysisspecs, "AnalysisSpec", title="Lab Water Spec",␣
↪SampleType=sampletype_uid, ResultsRange=rr)
```

Create an Analysis Request:

```
>>> values = {
...     'Client': client.UID(),
...     'Contact': contact1.UID(),
...     'CContact': contact2.UID(),
...     'SamplingDate': date_now,
...     'DateSampled': date_now,
...     'SampleType': sampletype.UID(),
...     'Priority': '1',
... }
```

```
>>> service_uids = [analysisservice1.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/water-0001>
```

4.9.3 ARAnalysesField

This field maintains *Analyses* within *AnalysesRequests*:

```
>>> field = ar.getField("Analyses")
>>> field.type
'analyses'
```

```
>>> from bika.lims.interfaces import IARAnalysesField
>>> IARAnalysesField.providedBy(field)
True
```

Getting Analyses

The *get* method returns a list of assined analyses brains:

```
>>> field.get(ar)
[<Products.ZCatalog.Catalog.mybrains object at ...>]
```

The full objects can be obtained by passing in *full_objects=True*:

```
>>> field.get(ar, full_objects=True)
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

The analysis *PH* is now contained in the AR:

```
>>> ar.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

Setting Analyses

The *set* method returns a list of new created analyses.

The field takes the following parameters:

- **items is a list that contains the items to be set:** The list can contain Analysis objects/brains, AnalysisService objects/brains and/or Analysis Service uids.
- **prices is a dictionary:** key = AnalysisService UID value = price
- **specs is a list of dictionaries:** key = AnalysisService UID value = dictionary: defined in ResultsRange field definition

Pass in all prior created Analysis Services:

```
>>> all_services = [analysisservice1, analysisservice2, analysisservice3]
>>> field.set(ar, all_services)
```

We expect to have now the *CA* and *MG* Analyses as well:

```
>>> sorted(ar.objectValues("Analysis"), key=methodcaller('getId'))
[<Analysis at /plone/clients/client-1/water-0001/CA>, <Analysis at /plone/clients/
↪client-1/water-0001/MG>, <Analysis at /plone/clients/client-1/water-0001/PH>]
```

Removing Analyses is done by omitting those from the *items* list:

```
>>> field.set(ar, [analysisservice1])
```

Now there should be again only one Analysis assigned:

```
>>> len(ar.objectValues("Analysis"))
1
```

We expect to have just the *PH* Analysis again:

```
>>> ar.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

The field can also handle UIDs of Analyses Services:

```
>>> service_uids = map(api.get_uid, all_services)
>>> field.set(ar, service_uids)
```

We expect again to have all the three Analyses:

```
>>> sorted(ar.objectValues("Analysis"), key=methodcaller("getId"))
[<Analysis at /plone/clients/client-1/water-0001/CA>, <Analysis at /plone/clients/
↪client-1/water-0001/MG>, <Analysis at /plone/clients/client-1/water-0001/PH>]
```

The field should also handle catalog brains:

```
>>> brains = api.search({"portal_type": "AnalysisService", "getKeyword": "CA"})
>>> brains
[<Products.ZCatalog.Catalog.mybrains object at 0x...>]
```

```
>>> brain = brains[0]
>>> api.get_title(brain)
'Calcium'
```

```
>>> field.set(ar, [brain])
```

We expect now to have just the *CA* analysis assigned:

```
>>> ar.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/water-0001/CA>]
```

Now let's try int mixed, one catalog brain and one object:

```
>>> field.set(ar, [analysisservice1, brain])
```

We expect now to have now *PH* and *CA*:

```
>>> sorted(ar.objectValues("Analysis"), key=methodcaller("getId"))
[<Analysis at /plone/clients/client-1/water-0001/CA>, <Analysis at /plone/clients/
↪client-1/water-0001/PH>]
```

Finally, we test it with an *Analysis* object:

```
>>> analysis1 = ar["PH"]
>>> field.set(ar, [analysis1])
```

```
>>> sorted(ar.objectValues("Analysis"), key=methodcaller("getId"))
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

Setting Analysis Specifications

Specifications are defined on the *ResultsRange* field of an Analysis Request. It is a dictionary with the following keys and values:

- keyword: The Keyword of the Analysis Service
- min: The minimum allowed value
- max: The maximum allowed value
- error: The error percentage
- hidemin: ?
- hidemax: ?
- rangecomment: ?

Each Analysis can request its own Specification (Result Range):

```
>>> field.set(ar, all_services)
```

```
>>> analysis1 = ar[analysisservice1.getKeyword()]
>>> analysis2 = ar[analysisservice2.getKeyword()]
>>> analysis3 = ar[analysisservice3.getKeyword()]
```

Now we will set the analyses with custom specifications through the ARAnalysesField. This should set the custom Specifications on the Analysis Request and have precedence over the lab specifications:

```
>>> spec_min = 5.5
>>> spec_max = 7.5
>>> error = 5
```

```
>>> arr1 = {"keyword": "PH", "min": 5.5, "max": 7.5, "error": 5, "hidemin": "",
↪"hidemax": "", "rangecomment": "My PH Spec"}
>>> arr2 = {"keyword": "MG", "min": 5.5, "max": 7.5, "error": 5, "hidemin": "",
↪"hidemax": "", "rangecomment": "My MG Spec"}
>>> arr3 = {"keyword": "CA", "min": 5.5, "max": 7.5, "error": 5, "hidemin": "",
↪"hidemax": "", "rangecomment": "My CA Spec"}
>>> arr = [arr1, arr2, arr3]
```

```
>>> all_analyses = [analysis1, analysis2, analysis3]
>>> field.set(ar, all_analyses, specs=arr)
```

```
>>> myspec1 = analysis1.getResultsRange()
>>> myspec1.get("rangecomment")
'My PH Spec'
```

```
>>> myspec2 = analysis2.getResultsRange()
>>> myspec2.get("rangecomment")
'My MG Spec'
```

```
>>> myspec3 = analysis3.getResultsRange()
>>> myspec3.get("rangecomment")
'My CA Spec'
```

Result Ranges are set to analyses level, but not present in the AR:

```
>>> sorted(map(lambda r: r.get("rangecomment"), ar.getResultsRange()))
[]
```

Now we simulate the form input data of the ARs "Manage Analysis" form, so that the User only selected the *PH* service and gave some custom specifications for this Analysis.

The specifications get applied if the keyword matches:

```
>>> ph_specs = {"keyword": analysis1.getKeyword(), "min": 5.2, "max": 7.9, "error": 3}
>>> field.set(ar, [analysis1], specs=[ph_specs])
```

We expect to have now just one Analysis set:

```
>>> analyses = field.get(ar, full_objects=True)
>>> analyses
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

And the specification should be according to the values we have set

```
>>> ph = analyses[0]
>>> phspec = ph.getResultsRange()
```

```
>>> phspec.get("min")
5.2
```

```
>>> phspec.get("max")
7.9
```

```
>>> phspec.get("error")
3
```

Setting Analyses Prices

Prices are primarily defined on Analyses Services:

```
>>> analysisservice1.getPrice()
'10.00'
```

```
>>> analysisservice2.getPrice()
'20.00'
```

```
>>> analysisservice3.getPrice()
'30.00'
```

Created Analyses inherit that price:

```
>>> field.set(ar, all_services)
```

```
>>> analysis1 = ar[analysisservice1.getKeyword()]
>>> analysis2 = ar[analysisservice2.getKeyword()]
>>> analysis3 = ar[analysisservice3.getKeyword()]
```

```
>>> analysis1.getPrice()
'10.00'
```

```
>>> analysis2.getPrice()
'20.00'
```

```
>>> analysis3.getPrice()
'30.00'
```

The *setter* also allows to set custom prices for the Analyses:

```
>>> prices = {
...     analysisservice1.UID(): "100",
...     analysisservice2.UID(): "200",
...     analysisservice3.UID(): "300",
... }
```

Now we set the field with all analyses services and new prices:

```
>>> field.set(ar, all_services, prices=prices)
```

The Analyses have now the new prices:

```
>>> analysis1.getPrice()
'100.00'
```

```
>>> analysis2.getPrice()
'200.00'
```

```
>>> analysis3.getPrice()
'300.00'
```

The Services should retain the old prices:

```
>>> analysisservice1.getPrice()
'10.00'
```

```
>>> analysisservice2.getPrice()
'20.00'
```

```
>>> analysisservice3.getPrice()
'30.00'
```

Calculations and Interim Fields

When an Analysis is assigned to an AR, it inherits its Calculation and Interim Fields.

Create some interim fields:

```
>>> interim1 = {"keyword": "A", "title": "Interim A", "value": 1, "hidden": False,
↪"type": "int", "unit": "x"}
>>> interim2 = {"keyword": "B", "title": "Interim B", "value": 2, "hidden": False,
↪"type": "int", "unit": "x"}
```

(continues on next page)

(continued from previous page)

```
>>> interim3 = {"keyword": "C", "title": "Interim C", "value": 3, "hidden": False,
↪"type": "int", "unit": "x"}
>>> interim4 = {"keyword": "D", "title": "Interim D", "value": 4, "hidden": False,
↪"type": "int", "unit": "x"}
```

Append interim field *A* to the *Total Hardness* Calculation:

```
>>> calc4.setInterimFields([interim1])
>>> map(lambda x: x["keyword"], calc4.getInterimFields())
['A']
```

Append interim field *B* to the *Total Hardness* Analysis Service:

```
>>> analysisservice4.setInterimFields([interim2])
>>> map(lambda x: x["keyword"], analysisservice4.getInterimFields())
['B', 'A']
```

Now we assign the *Total Hardness* Analysis Service:

```
>>> field.set(ar, [analysisservice4])
>>> new_analyses = get_analyses_from(ar, analysisservice4)
>>> analysis = new_analyses[0]
>>> analysis
<Analysis at /plone/clients/client-1/water-0001/THCaCO3>
```

The created Analysis has the same Calculation attached, as the Analysis Service:

```
>>> analysis_calc = analysis.getCalculation()
>>> analysis_calc
<Calculation at /plone/bika_setup/bika_calculations/calculation-4>
```

And therefore, also the same Interim Fields as the Calculation:

```
>>> map(lambda x: x["keyword"], analysis_calc.getInterimFields())
['A']
```

The Analysis also inherits the Interim Fields of the Analysis Service:

```
>>> map(lambda x: x["keyword"], analysis.getInterimFields())
['B', 'A']
```

But what happens if the Interim Fields of either the Analysis Service or of the Calculation change and the AR is updated with the same Analysis Service?

Change the Interim Field of the Calculation to *C*:

```
>>> calc4.setInterimFields([interim3])
>>> map(lambda x: x["keyword"], calc4.getInterimFields())
['C']
```

Change the Interim Fields of the Analysis Service to *D*:

```
>>> analysisservice4.setInterimFields([interim4])
```

The Analysis Service returns the interim fields from the Calculation too:

```
>>> map(lambda x: x["keyword"], analysisservice4.getInterimFields())
['D', 'C']
```

Update the AR with the new Analysis Service:

```
>>> field.set(ar, [analysisservice4])
```

The Analysis should be still there:

```
>>> analysis = ar[analysisservice4.getKeyword()]
>>> analysis
<Analysis at /plone/clients/client-1/water-0001/THCaCO3>
```

The calculation should be still there:

```
>>> analysis_calc = analysis.getCalculation()
>>> analysis_calc
<Calculation at /plone/bika_setup/bika_calculations/calculation-4>
```

And therefore, also the same Interim Fields as the Calculation:

```
>>> map(lambda x: x["keyword"], analysis_calc.getInterimFields())
['C']
```

The existing Analysis retains the initial Interim Fields of the Analysis Service, together with the interim from the associated Calculation:

```
>>> map(lambda x: x["keyword"], analysis.getInterimFields())
['B', 'A']
```

Worksheets

If the an Analysis is assigned to a worksheet, it should be detached before it is removed from an Analysis Request.

Assign the *PH* Analysis:

```
>>> field.set(ar, [analysisservice1])
>>> new_analyses = ar.getAnalyses(full_objects=True)
```

Create a new Worksheet and assign the Analysis to it:

```
>>> ws = api.create(worksheets, "Worksheet", "WS")
>>> analysis = new_analyses[0]
>>> ws.addAnalysis(analysis)
```

The analysis is not associated to the Worksheet because the AR is not received:

```
>>> analysis.getWorksheet() is None
True
>>> ws.getAnalyses()
[]
>>> success = do_action_for(ar, "receive")
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Try to assign again the Analysis to the Worksheet:

```
>>> ws.addAnalysis(analysis)
```

The analysis is associated to the Worksheet:

```
>>> analysis.getWorksheet().UID() == ws.UID()
True
```

The worksheet contains now the Analysis:

```
>>> ws.getAnalyses()
[<Analysis at /plone/clients/client-1/water-0001/PH>]
```

Removing the analysis from the AR also unassignes it from the worksheet:

```
>>> field.set(ar, [analysisservice2])
```

```
>>> ws.getAnalyses()
[]
```

Dependencies

The Analysis Service *Total Hardness* uses the *Total Hardness* Calculation:

```
>>> analysisservice4.getCalculation()
<Calculation at /plone/bika_setup/bika_calculations/calculation-4>
```

The Calculation is dependent on the *CA* and *MG* Services through its Formula:

```
>>> analysisservice4.getCalculation().getFormula()
'[CA] + [MG]'
```

Get the dependent services:

```
>>> sorted(analysisservice4.getServiceDependencies(), key=methodcaller('getId'))
[<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>,
 ↪<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-3>]
```

We expect that dependent services get automatically set:

```
>>> field.set(ar, [analysisservice4])
```

```
>>> sorted(ar.objectValues("Analysis"), key=methodcaller('getId'))
[<Analysis at /plone/clients/client-1/water-0001/CA>, <Analysis at /plone/clients/
 ↪client-1/water-0001/MG>, <Analysis at /plone/clients/client-1/water-0001/THCaCO3>]
```

4.9.4 Attachments

Attachments can be assigned to the Analysis Request or to individual Analyses.

If an attachment was assigned to a specific analysis, it must be deleted if the Analysis was removed, see https://github.com/senaite/senaite.core/issues/1025.

Hoever, for invalidated/retested ARs the attachments are linked to the original AR/Analyses as well as to the retested AR/Analyses. Therefore, it must be retained when it is still referenced.

Create a new AR and assign the *PH* analysis:

```
>>> service_uids = [analysisservice1.UID()]
>>> ar2 = create_analysisrequest(client, request, values, service_uids)
>>> ar2
<AnalysisRequest at /plone/clients/client-1/water-0002>
```

Get the analysis:

```
>>> an1 = ar2[analysisservice1.getKeyword()]
>>> an1
<Analysis at /plone/clients/client-1/water-0002/PH>
```

It should have *no* attachments assigned:

```
>>> an1.getAttachment()
[]
```

We create a new attachment in the client and assign it to this specific analysis:

```
>>> att1 = api.create(ar2.getClient(), "Attachment", title="PH.png")
>>> an1.setAttachment(att1)
>>> an1.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-1>]
```

Now we remove the *PH* analysis. Since it is prohibited by the field to remove all analyses from an AR, we will set here some other analyses instead:

```
>>> field.set(ar2, [analysisservice2, analysisservice3])
```

The attachment should be deleted from the client folder as well:

```
>>> att1.getId() in ar2.getClient().objectIds()
False
```

Re-adding the *PH* analysis should start with no attachments:

```
>>> field.set(ar2, [analysisservice1, analysisservice2, analysisservice3])
>>> an1 = ar2[analysisservice1.getKeyword()]
>>> an1.getAttachment()
[]
```

This should work as well when multiple attachments are assigned.

```
>>> field.set(ar2, [analysisservice1, analysisservice2])
```

```
>>> an1 = ar2[analysisservice1.getKeyword()]
>>> an2 = ar2[analysisservice2.getKeyword()]
```

```
>>> att2 = api.create(ar2.getClient(), "Attachment", title="test2.png")
>>> att3 = api.create(ar2.getClient(), "Attachment", title="test3.png")
>>> att4 = api.create(ar2.getClient(), "Attachment", title="test4.png")
```

```
>>> att5 = api.create(ar2.getClient(), "Attachment", title="test5.png")
>>> att6 = api.create(ar2.getClient(), "Attachment", title="test6.png")
>>> att7 = api.create(ar2.getClient(), "Attachment", title="test7.png")
```

Assign the first half of the attachments to the *PH* analysis:

```
>>> an1.setAttachment([att2, att3, att4])
>>> an1.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-2>, <Attachment at /plone/clients/
↪client-1/attachment-3>, <Attachment at /plone/clients/client-1/attachment-4>]
```

Assign the second half of the attachments to the *Magnesium* analysis:

```
>>> an2.setAttachment([att5, att6, att7])
>>> an2.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-5>, <Attachment at /plone/clients/
↪client-1/attachment-6>, <Attachment at /plone/clients/client-1/attachment-7>]
```

Removing the *PH* analysis should also remove all the assigned attachments:

```
>>> field.set(ar2, [analysisservice2])
```

```
>>> att2.getId() in ar2.getClient().objectIds()
False
```

```
>>> att3.getId() in ar2.getClient().objectIds()
False
```

```
>>> att4.getId() in ar2.getClient().objectIds()
False
```

The attachments of *Magnesium* should be still there:

```
>>> att5.getId() in ar2.getClient().objectIds()
True
```

```
>>> att6.getId() in ar2.getClient().objectIds()
True
```

```
>>> att7.getId() in ar2.getClient().objectIds()
True
```

Attachments linked to multiple ARs/ANs

When an AR is invalidated, a copy of it get created for retesting. This copy holds also the Attachments as references.

Create a new AR for that and assign a service w/o caclucation:

```
>>> service_uids = [analysisservice5.UID()]
>>> ar3 = create_analysisrequest(client, request, values, service_uids)
>>> ar3
<AnalysisRequest at /plone/clients/client-1/water-0003>
```

Receive the AR:

```
>>> transitioned = do_action_for(ar3, "receive")
>>> transitioned[0]
True
```

```
>>> ar3.portal_workflow.getInfoFor(ar3, "review_state")
'sample_received'
```

Assign an attachment to the AR:

```
>>> att_ar = api.create(ar3.getClient(), "Attachment", title="ar.png")
>>> ar3.setAttachment(att_ar)
>>> ar3.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-8>]
```

Assign an attachment to the Analysis:

```
>>> an = ar3[analysisservice5.getKeyword()]
>>> att_an = api.create(ar3.getClient(), "Attachment", title="an.png")
>>> an.setAttachment(att_an)
>>> an.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-9>]
```

Set the results of the Analysis and submit and verify them directly. Therefore, self-verification must be allowed in the setup:

```
>>> setup.setSelfVerificationEnabled(True)
```

```
>>> for analysis in ar3.getAnalyses(full_objects=True):
...     analysis.setResult("12")
...     transitioned = do_action_for(analysis, "submit")
...     transitioned = do_action_for(analysis, "verify")
```

Finally we can publish the AR:

```
>>> transitioned = do_action_for(ar3, "publish")
```

And invalidate it directly:

```
>>> transitioned = do_action_for(ar3, "invalidate")
```

A new AR is automatically created for retesting:

```
>>> ar_retest = ar3.getRetest()
>>> ar_retest
<AnalysisRequest at /plone/clients/client-1/water-0003-R01>
```

```
>>> an_retest = ar3.getRetest()[analysisservice5.getKeyword()]
>>> an_retest
<Analysis at /plone/clients/client-1/water-0003-R01/NoCalc>
```

However, this retest AR **references the same Attachments** as the original AR:

```
>>> ar_retest.getAttachment() == ar3.getAttachment()
True
```

```
>>> att_ar.getLinkedRequests()
[<AnalysisRequest at /plone/clients/client-1/water-0003-R01>, <AnalysisRequest at /
↪plone/clients/client-1/water-0003>]
```

```
>>> att_ar.getLinkedAnalyses()
[]
```

And all contained Analyses of the retest keep references to the same Attachments:

```
>>> an_retest.getAttachment() == an.getAttachment()
True
```

```
>>> att_an.getLinkedRequests()
[]
```

```
>>> att_an.getLinkedAnalyses()
[<Analysis at /plone/clients/client-1/water-0003/NoCalc>, <Analysis at /plone/clients/
↪client-1/water-0003-R01/NoCalc>]
```

This means that removing that attachment from the retest should **not** delete the attachment from the original AR:

```
>>> field.set(ar_retest, [analysisservice1])
>>> an.getAttachment()
[<Attachment at /plone/clients/client-1/attachment-9>]
```

```
>>> att_an.getId() in ar3.getClient().objectIds()
True
```

And the attachment is now only linked to the attachment of the original analysis:

```
>>> att_an.getLinkedAnalyses()
[<Analysis at /plone/clients/client-1/water-0003/NoCalc>]
```

4.10 AR Analyses Field when using Partitions

The setter of the ARAnalysesField takes descendants (partitions) and ancestors from the current instance into account to prevent inconsistencies: In a Sample lineage analyses from a node are always masked by same analyses in leaves. This can drive to inconsistencies and therefore, there is the need to keep the tree without duplicates.

Running this test from the buildout directory:

bin/test test_textual_doctests -t ARAnalysesFieldWithPartitions

4.10.1 Test Setup

Needed imports:

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils.analysisrequest import create_partition
>>> from bika.lims.workflow import doActionFor as do_action_for
```

(continues on next page)

(continued from previous page)

```
>>> from zope.interface import alsoProvides
>>> from zope.interface import noLongerProvides
```

Functional Helpers:

```
>>> def new_sample(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime().strftime("%Y-%m-%d"),
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def get_analysis_from(sample, service):
...     service_uid = api.get_uid(service)
...     for analysis in sample.getAnalyses(full_objects=True):
...         if analysis.getServiceUID() == service_uid:
...             return analysis
...     return None
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = api.get_setup()
```

Create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> Mg = api.create(setup.bika_analysisservices, "AnalysisService", title="Magnesium",
↪ Keyword="Mg", Price="20", Category=category.UID())
```

4.10.2 Creation of a Sample with a Partition

Create a Sample and receive:

```
>>> sample = new_sample([Cu, Fe])
```

Create a Partition containing of the Sample, containing the analysis *Cu*:

```
>>> cu = get_analysis_from(sample, Cu)
>>> partition = create_partition(sample, request, [cu])
```

The analysis 'Cu' lives in the partition:

```
>>> cu = get_analysis_from(partition, Cu)
>>> api.get_parent(cu) == partition
True
```

Although is also returned by the primary:

```
>>> cu = get_analysis_from(sample, Cu)
>>> api.get_parent(cu) == partition
True
>>> api.get_parent(cu) == sample
False
```

4.10.3 Analyses retrieval

Get the ARAnalysesField to play with:

```
>>> field = sample.getField("Analyses")
```

get_from_instance

When asked for *Fe* when the primary is given, it returns the analysis, cause it lives in the primary:

```
>>> fe = field.get_from_instance(sample, Fe)[0]
>>> fe.getServiceUID() == api.get_uid(Fe)
True
```

But when asked for *Cu* when the primary is given, it returns empty, cause it lives in the partition:

```
>>> field.get_from_instance(sample, Cu)
[]
```

While it returns the analysis when the partition is used:

```
>>> cu = field.get_from_instance(partition, Cu)[0]
>>> cu.getServiceUID() == api.get_uid(Cu)
True
```

But when asking the partition for *Fe* it returns empty, cause it lives in the ancestor:

```
>>> field.get_from_instance(partition, Fe)
[]
```

get_from_ancestor

When asked for *Fe* to primary, it returns empty because there is no ancestor containing *Fe*:

```
>>> field.get_from_ancestor(sample, Fe)
[]
```

But when asked for *Fe* to the partition, it returns the analysis, cause it it lives in an ancestor from the partition:

```
>>> fe = field.get_from_ancestor(partition, Fe)[0]
>>> fe.getServiceUID() == api.get_uid(Fe)
True
```

If I ask for *Cu*, that lives in the partition, it will return empty for both:

```
>>> field.get_from_ancestor(sample, Cu)
[]
```

```
>>> field.get_from_ancestor(partition, Cu)
[]
```

get_from_descendant

When asked for *Fe* to primary, it returns None because there is no descendant containing *Fe*:

```
>>> field.get_from_descendant(sample, Fe)
[]
```

And same with partition:

```
>>> field.get_from_descendant(partition, Fe)
[]
```

When asked for *Cu* to primary, it returns the analysis, because it lives in a descendant (partition):

```
>>> field.get_from_descendant(sample, Cu)
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>]
```

But returns None if I ask to the partition:

```
>>> field.get_from_descendant(partition, Cu)
[]
```

get_analyses_from_descendants

It returns the analyses contained by the descendants:

```
>>> field.get_analyses_from_descendants(sample)
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>]
```

```
>>> field.get_analyses_from_descendants(partition)
[]
```

4.10.4 Resolution of analyses from the Sample lineage

resolve_analyses

Resolves the analysis from the sample lineage if exists:

```
>>> field.resolve_analyses(sample, Fe)
[<Analysis at /plone/clients/client-1/W-0001/Fe>]
```

```
>>> field.resolve_analyses(sample, Cu)
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>]
```

```
>>> field.resolve_analyses(sample, Au)
[]
```

But when we use the partition and the analysis is found in an ancestor, it moves the analysis into the partition:

```
>>> field.resolve_analyses(partition, Fe)
[<Analysis at /plone/clients/client-1/W-0001-P01/Fe>]
```

```
>>> sample.objectValues("Analysis")
[]
```

```
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>]
```

4.10.5 Addition of analyses

add_analysis

If we try to add now an analysis that already exists, either in the partition or in the primary, the analysis won't be added:

```
>>> field.add_analysis(sample, Fe)
>>> sample.objectValues("Analysis")
[]
```

```
>>> field.add_analysis(partition, Fe)
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>]
```

If we add a new analysis, this will be added in the sample we are working with:

```
>>> field.add_analysis(sample, Au)
>>> sample.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001/Au>]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>]
```

Apply the changes:

```
>>> transaction.commit()
```

If I try to add an analysis that exists in an ancestor, the analysis gets moved while the function returns None:

```
>>> field.add_analysis(partition, Au)
>>> sample.objectValues("Analysis")
[]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

4.10.6 Set analyses

If we try to set same analyses as before to the root sample, nothing happens because the analyses are already there:

```
>>> field.set(sample, [Cu, Fe, Au])
```

The analyses still belong to the partition though:

```
>>> sample.objectValues("Analysis")
[]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

Same result if I set the analyses to the partition:

```
>>> field.set(partition, [Cu, Fe, Au])
>>> sample.objectValues("Analysis")
[]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

If I add a new analysis in the list, the analysis is successfully added:

```
>>> field.set(sample, [Cu, Fe, Au, Mg])
>>> sample.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001/Mg>]
```

And the partition keeps its own analyses:

```
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

Apply the changes:

```
>>> transaction.commit()
```

If I set the same analyses to the partition, the *Mg* analysis is moved into the partition:

```
>>> field.set(partition, [Cu, Fe, Au, Mg])
>>> sample.objectValues("Analysis")
[]
```

(continues on next page)

(continued from previous page)

```
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>,
↪<Analysis at /plone/clients/client-1/W-0001-P01/Mg>]
```

To remove *Mg* analysis, pass the list without *Mg*:

```
>>> field.set(sample, [Cu, Fe, Au])
```

The analysis *Mg* has been removed, although it belonged to the partition:

```
>>> sample.objectValues("Analysis")
[]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

But if I add a new analysis to the primary and I try to remove it from the partition, nothing will happen:

```
>>> field.set(sample, [Cu, Fe, Au, Mg])
```

```
>>> field.set(partition, [Cu, Fe, Au])
```

```
>>> sample.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001/Mg>]
>>> partition.objectValues("Analysis")
[<Analysis at /plone/clients/client-1/W-0001-P01/Cu>, <Analysis at /plone/clients/
↪client-1/W-0001-P01/Fe>, <Analysis at /plone/clients/client-1/W-0001-P01/Au>]
```

4.11 Abbott's m2000 Real Time import interface

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t AbbottM2000rtImportInterface
```

4.11.1 Test Setup

Needed imports:

~~ code:

```
>>> import codecs
>>> import os
>>> import transaction
>>> from DateTime import DateTime
>>> from Products.CMFCore.utils import getToolByName
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from senaite.core.exportimport import instruments
>>> from senaite.core.exportimport.instruments.abbott.m2000rt.m2000rt import␣
↪Abbottm2000rtTSVParser
```

(continues on next page)

(continued from previous page)

```
>>> from senaite.core.exportimport.instruments.abbott.m2000rt.m2000rt import␣
↪Abbottm2000rtImporter
>>> from bika.lims.browser.resultsimport.resultsimport import ConvertToUploadFile
```

Functional helpers:

~~ code:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

~~ code:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_instruments = bika_setup.bika_instruments
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_calculations = bika_setup.bika_calculations
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager:

~~ code:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.11.2 Availability of instrument interface

Check that the instrument interface is available:

~~ code:

```
>>> exims = []
>>> for exim_id in instruments.__all__:
...     exims.append(exim_id)
>>> 'abbott.m2000rt.m2000rt' in exims
True
```

4.11.3 Assigning the Import Interface to an Instrument

Create an *Instrument* and assign to it the tested Import Interface:

~~ code:

```
>>> instrument = api.create(bika_instruments, "Instrument", title="Instrument-1")
>>> instrument
<Instrument at /plone/bika_setup/bika_instruments/instrument-1>
>>> instrument.setImportDataInterface(['abbott.m2000rt.m2000rt'])
>>> instrument.getImportDataInterface()
['abbott.m2000rt.m2000rt']
```

4.11.4 Import test

Required steps: Create and receive Analysis Request for import test

An *AnalysisRequest* can only be created inside a *Client*, and it also requires a *Contact* and a *SampleType*:

~~ code:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="NARALABS", ClientID="NLABS")
>>> client
<Client at /plone/clients/client-1>
>>> contact = api.create(client, "Contact", Firstname="Juan", Surname="Gallostra")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="H2O",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

Create an *AnalysisCategory* (which categorizes different *AnalysisServices*), and add to it an *AnalysisService*. This service matches the service specified in the file from which the import will be performed:

~~ code:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
>>> analysisservice = api.create(bika_analysisservices,
...                              "AnalysisService",
...                              title="HIV06ml",
...                              ShortTitle="hiv06",
...                              Category=analysiscategory,
...                              Keyword="HIV06ml")
>>> analysisservice
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>

>>> total_calc = api.create(bika_calculations, 'Calculation', title='TotalCalc')
>>> total_calc.setFormula('[HIV06ml] * 100')
>>> analysisservice2 = api.create(bika_analysisservices,
...                              "AnalysisService",
...                              title="Test Total Results",
...                              ShortTitle="TestTotalResults",
...                              Category=analysiscategory,
...                              Keyword="TTR")
>>> analysisservice2.setUseDefaultCalculation(False)
>>> analysisservice2.setCalculation(total_calc)
```

(continues on next page)

(continued from previous page)

```
>>> analysisservice2
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>
```

Set some interim fields present in the results test file intoto the created AnalysisService, so not on the second server:

~~ code:

```
>>> service_interim_fields = [{'keyword': 'ASRExpDate',
...                            'title': 'ASRExpDate',
...                            'unit': '',
...                            'default': ''},
...                           {'keyword': 'ASRLotNumber',
...                            'title': 'ASRLotNumber',
...                            'unit': '',
...                            'default': ''},
...                           {'keyword': 'AssayCalibrationTime',
...                            'title': 'AssayCalibrationTime',
...                            'unit': '',
...                            'default': ''},
...                           {'keyword': 'FinalResult',
...                            'title': 'FinalResult',
...                            'unit': '',
...                            'default': ''},
...                           {'keyword': 'Location',
...                            'title': 'Location',
...                            'unit': '',
...                            'default': ''},
...                           ]
>>> analysisservice.setInterimFields(service_interim_fields)
>>> analysisservice.getInterimFields()
[{'default': '', 'unit': '', 'keyword': 'ASRExpDate', 'title': 'ASRExpDate'},
 {'default': '', 'unit': '', 'keyword': 'ASRLotNumber', 'title': 'ASRLotNumber'},
 {'default': '', 'unit': '', 'keyword': 'AssayCalibrationTime', 'title':
↪'AssayCalibrationTime'},
 {'default': '', 'unit': '', 'keyword': 'FinalResult', 'title': 'FinalResult'},
 {'default': '', 'unit': '', 'keyword': 'Location', 'title': 'Location'}]
```

Create an *AnalysisRequest* with this *AnalysisService* and receive it:

~~ code:

```
>>> values = {
...            'Client': client.UID(),
...            'Contact': contact.UID(),
...            'SamplingDate': date_now,
...            'DateSampled': date_now,
...            'SampleType': sampletype.UID()
...           }
>>> service_uids = [analysisservice.UID(), analysisservice2.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/H2O-0001>
>>> ar.getReceivedBy()
''
>>> wf = getToolByName(ar, 'portal_workflow')
>>> wf.doActionFor(ar, 'receive')
>>> ar.getReceivedBy()
'test_user_1_'
```

Import test

Load results test file and import the results:

~~ code:

```
>>> dir_path = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..', 'files
↪'))
>>> temp_file = codecs.open(dir_path + '/AbbottM2000.log.123',
...                         encoding='utf-8-sig')
>>> test_file = ConvertToUploadFile(temp_file)
>>> abbott_parser = Abbottm2000rtTSVParser(test_file)
>>> importer = Abbottm2000rtImporter(parser=abbott_parser,
...                                  context=portal,
...                                  allowed_ar_states=['sample_received',
↪'attachment_due', 'to_be_verified'],
...                                  allowed_analysis_states=None,
...                                  override=[True, True])
>>> importer.process()
```

Check from the importer logs that the file from where the results have been imported is indeed the specified file:

~~ code:

```
>>> '/AbbottM2000.log.123' in importer.logs[0]
True
```

Check the rest of the importer logs to verify that the values were correctly imported:

~~ code:

```
>>> importer.logs[1:]
['End of file reached successfully: 24 objects, 1 analyses, 24 results', 'Allowed␣
↪Sample states: sample_received, attachment_due, to_be_verified', 'Allowed analysis␣
↪states: unassigned, assigned, to_be_verified', "H2O-0001 result for
↪'HIV06ml:ASRExpDate': '20141211'", "H2O-0001 result for 'HIV06ml:ASRLotNumber':
↪'0123456'", "H2O-0001 result for 'HIV06ml:AssayCalibrationTime': '20150423 16:37:05'
↪", "H2O-0001 result for 'HIV06ml:FinalResult': '18'", "H2O-0001 result for
↪'HIV06ml:Location': 'A12'", "H2O-0001: calculated result for 'TTR': '1800.0'", "H2O-
↪0001: [u'Analysis HIV06ml'] imported sucessfully", 'Import finished successfully: 1␣
↪Samples and 1 results updated']
```

And finally check if indeed the analysis has the imported results:

~~ code:

```
>>> analyses = ar.getAnalyses()
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title == 'HIV06ml
↪'][0]
>>> an.getResult()
'18'
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title == 'Test␣
↪Total Results'][0]
>>> an.getResult()
'1800.0'
```

4.12 Action Handler Pool

The *ActionHandlerPool* is a singleton instance to increase performance by postponing reindexing operations for objects.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t ActionHandlerPool
```

4.13 Test Setup

Needed Imports:

```
>>> from bika.lims.workflow import ActionHandlerPool
```

4.14 Testing

Getting an instance of the action handler pool:

```
>>> pool = ActionHandlerPool.get_instance()
```

```
>>> pool
<ActionHandlerPool for UIDs:[]>
```

When a piece of code is utilizing the utility function *doActionFor*, the pool is utilized to increase the performance by

- avoiding the same transition to be multiple times
- postponing the reindexing to the end of the process

For this to work, each calling function needs to call *queue_pool()* to postpone (eventual) multiple reindex operation:

```
>>> pool.queue_pool()
```

This will *increase* the internal *num_calls* counter:

```
>>> pool.num_calls
1
```

If all operations are done by the calling code, it has to call *resume()*, which will decrease the counter by 1:

```
>>> pool.resume()
```

This will *decrease* the internal *num_calls* counter:

```
>>> pool.num_calls
0
```

Multiple calls to *resume()* should not lead to a negative counter:

```
>>> for i in range(10):
...     pool.resume()
```

```
>>> pool.num_calls
0
```

Because the *ActionHandlerPool* is a singleton, we must ensure that it is thread safe. This means that concurrent access to this counter must be protected.

To simulate this, we will need to simulate concurrent calls to *queue_pool()*, which will add some lag in between the reading and writing operation.

```
>>> import random
>>> import threading
>>> import time
```

```
>>> threads = []
```

```
>>> def simulate_queue_pool(tid):
...     pool.queue_pool()
...     time.sleep(random.random())
```

```
>>> for num in range(100):
...     t = threading.Thread(target=simulate_queue_pool, args=(num, ))
...     threads.append(t)
...     t.start()
```

```
>>> for t in threads:
...     t.join()
```

4.15 Alphanumber

Tests the Alphanumber object, useful for alphanumeric IDs generation

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t Alphanumber
```

4.15.1 Test Setup

Needed Imports:

```
>>> import re
>>> from bika.lims import api
>>> from bika.lims.alphanumber import to_decimal
>>> from bika.lims.alphanumber import Alphanumber
```

Create and test basic alphanumeric functions:

```
>>> alpha = Alphanumber(0)
>>> int(alpha)
0
>>> str(alpha)
'AAA000'
>>> repr(alpha)
```

(continues on next page)

(continued from previous page)

```
'AAA000'
>>> format(alpha, "2a2d")
'AA00'
>>> alpha.format("5a4d")
'AAAAA0000'
>>> "{alpha:2a4d}".format(alpha=alpha)
'AA0000'
>>> alpha1 = alpha + 1
>>> int(alpha1)
1
>>> str(alpha1)
'AAA001'
>>> repr(alpha1)
'AAA001'
>>> format(alpha1, "2a2d")
'AA01'
>>> alpha1.format("5a4d")
'AAAAA0001'
>>> "{alpha:2a4d}".format(alpha=alpha1)
'AA0001'
>>> alpha2 = Alphanumber(2674, num_digits=2)
>>> int(alpha2)
2674
>>> str(alpha2)
'ABB01'
```

Addition of an integer:

```
>>> alpha3 = alpha2 + 1
>>> int(alpha3)
2675
>>> str(alpha3)
'ABB02'
>>> to_decimal(str(alpha3))
2675
```

Addition of another Alphanumber object:

```
>>> alpha3 = alpha2 + alpha1
>>> int(alpha3)
2675
>>> str(alpha3)
'ABB02'
>>> alpha3 = alpha2 + alpha2
>>> int(alpha3)
5348
>>> str(alpha3)
'ACC02'
>>> to_decimal(str(alpha3))
5348
```

Subtraction of an integer:

```
>>> alpha3 = alpha2 - 1
>>> int(alpha3)
2673
>>> str(alpha3)
```

(continues on next page)

(continued from previous page)

```
'ABA99'
>>> to_decimal(str(alpha3))
2673
```

Subtraction of another Alphanumber object:

```
>>> alpha3 = alpha2 - alpha1
>>> int(alpha3)
2673
>>> str(alpha3)
'ABA99'
>>> alpha3 = alpha2 - alpha2
>>> int(alpha3)
0
>>> str(alpha3)
'AAA00'
>>> to_decimal(str(alpha3))
0
```

We can also create the instance with a string representing an alpha number:

```
>>> alpha = Alphanumber("ABB23", num_chars=3, num_digits=2)
>>> str(alpha)
'ABB23'
>>> int(alpha)
2696
>>> to_decimal(str(alpha))
2696
```

We can even change the number of digits to default (3 digits) and the result will be formatted accordingly:

```
>>> alpha = Alphanumber("ABB23")
>>> str(alpha)
'AAC698'
>>> int(alpha)
2696
```

Or we can do the same, but using another Alphanumber instance as argument:

```
>>> alpha = Alphanumber(alpha, num_chars=2)
>>> str(alpha)
'AC698'
>>> int(alpha)
2696
```

We can also use our own alphabet:

```
>>> alpha = Alphanumber(alpha, alphabet="yu")
>>> str(alpha)
'yuy698'
>>> int(alpha)
2696
>>> to_decimal(str(alpha), alphabet="yu")
2696
```

And we can add or subtract regardless of alphabet, number of digits and number of characters:

```
>>> alpha1 = Alphanumber("ABB23")
>>> int(alpha1)
2696
>>> alpha2 = Alphanumber("yu753", alphabet="yu")
>>> int(alpha2)
1752
>>> alpha3 = alpha1 + alpha2
>>> int(alpha3)
4448
>>> str(alpha3)
'AAE452'
```

Formatted value must change when a different number of digits is used:

```
>>> str(alpha3)
'AAE452'
>>> format(alpha3, "2a3d")
'AE452'
>>> format(alpha3, "5a3d")
'AAAAE452'
>>> format(alpha3, "3a2d")
'ABS92'
```

We can also compare two Alphanumbers:

```
>>> alpha3 > alpha2
True
```

```
>>> alpha1 > alpha3
False
```

```
>>> alpha4 = Alphanumber(4448)
>>> alpha3 == alpha4
True
```

Or get the max and the min:

```
>>> alphas = [alpha1, alpha3, alpha2]
>>> alpha_max = max(alphas)
>>> int(alpha_max)
4448
```

```
>>> alpha_min = min(alphas)
>>> int(alpha_min)
1752
```

We can also convert to int directly:

```
>>> int(alpha4)
4448
```

Or use the lims api:

```
>>> api.to_int(alpha4)
4448
```

4.16 Analysis Profile

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t AnalysisProfile
```

Needed Imports:

```
>>> import re
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.content.analysisrequest import AnalysisRequest
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils import tmpID
>>> from bika.lims.interfaces import ISubmitted
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getCurrentState
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def get_services(sample):
...    analyses = sample.getAnalyses(full_objects=True)
...    services = map(lambda an: an.getAnalysisService(), analyses)
...    return services
```

```
>>> def receive_sample(sample):
...     do_action_for(sample, "receive")
```

```
>>> def submit_analyses(sample):
...     for analysis in sample.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def verify_analyses(sample):
...     for analysis in sample.getAnalyses(full_objects=True):
...         if ISubmitted.providedBy(analysis):
...             do_action_for(analysis, "verify")
```

```
>>> def retract_analyses(sample):
...     for analysis in sample.getAnalyses(full_objects=True):
...         if ISubmitted.providedBy(analysis):
...             do_action_for(analysis, "retract")
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> supplier = api.create(setup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> Zn = api.create(setup.bika_analysisservices, "AnalysisService", title="Zink",␣
↪Keyword="Zn", Price="20", Category=category.UID())
>>> service_uids1 = [Cu.UID(), Fe.UID(), Au.UID()]
>>> service_uids2 = [Zn.UID()]
>>> service_uids3 = [Cu.UID(), Fe.UID(), Au.UID(), Zn.UID()]
>>> profile1 = api.create(setup.bika_analysisprofiles, "AnalysisProfile", title=
↪"Profile", Service=service_uids1)
>>> profile2 = api.create(setup.bika_analysisprofiles, "AnalysisProfile", title=
↪"Profile", Service=service_uids2)
>>> profile3 = api.create(setup.bika_analysisprofiles, "AnalysisProfile", title=
↪"Profile", Service=service_uids3)
```

4.16.1 Assign Profile(s)

Assigning Analysis Profiles adds the Analyses of the profile to the sample.

```
>>> setup.setSelfVerificationEnabled(True)
```

```
>>> values = {
...     'Client': client.UID(),
...     'Contact': contact.UID(),
...     'DateSampled': date_now,
...     'SampleType': sampletype.UID()}
```

Create some Analysis Requests:

```
>>> ar1 = create_analysisrequest(client, request, values, [Au.UID()])
>>> ar2 = create_analysisrequest(client, request, values, [Fe.UID()])
>>> ar3 = create_analysisrequest(client, request, values, [Cu.UID()])
```

Apply the profile object. Note the custom *setProfiles* (plural) setter:

```
>>> ar1.setProfiles(profile1)
```

All analyses from the profile should be added to the sample:

```
>>> services = get_services(ar1)
>>> set(map(api.get_uid, services)).issuperset(service_uids1)
True
```

The profile is applied to the sample:

```
>>> profile1 in ar1.getProfiles()
True
```

Apply the profile UID:

```
>>> ar2.setProfiles(profile2.UID())
```

All analyses from the profile should be added to the sample:

```
>>> services = get_services(ar2)
>>> set(map(api.get_uid, services)).issuperset(service_uids2)
True
```

The profile is applied to the sample:

```
>>> profile2 in ar2.getProfiles()
True
```

Apply multiple profiles:

```
>>> ar3.setProfiles([profile1, profile2, profile3.UID()])
```

All analyses from the profiles should be added to the sample:

```
>>> services = get_services(ar3)
>>> set(map(api.get_uid, services)).issuperset(service_uids1 + service_uids2 +␣
↪service_uids3)
True
```

4.16.2 Remove Profile(s)

Removing an analyis Sample retains the assigned analyses:

```
>>> analyses = ar1.getAnalyses(full_objects=True)
>>> ar1.setProfiles([])
>>> ar1.getProfiles()
[]
```

```
>>> set(ar1.getAnalyses(full_objects=True)) == set(analyses)
True
```

4.16.3 Assigning Profiles in "to_be_verified" status

```
>>> ar4 = create_analysisrequest(client, request, values, [Au.UID()])
```

```
>>> receive_sample(ar4)
>>> submit_analyses(ar4)
```

```
>>> api.get_workflow_status_of(ar4)
'to_be_verified'
```

```
>>> ar4.getProfiles()
[]
```

Setting the profile works up to this state:

```
>>> ar4.setProfiles(profile1.UID())
>>> api.get_workflow_status_of(ar4)
'sample_received'
```

```
>>> services = get_services(ar3)
>>> set(map(api.get_uid, services)).issuperset(service_uids1 + [Au.UID()])
True
```

4.17 Analysis Request invalidate

Running this test from the buildout directory:

bin/test test_textual_doctests -t AnalysisRequestInvalidate

4.17.1 Test Setup

Needed Imports:

```
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
```

```
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> browser = self.getBrowser()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_labcontacts = bika_setup.bika_labcontacts
>>> bika_storagelocations = bika_setup.bika_storagelocations
>>> bika_samplingdeviations = bika_setup.bika_samplingdeviations
>>> bika_sampleconditions = bika_setup.bika_sampleconditions
>>> portal_url = portal.absolute_url()
>>> bika_setup_url = portal_url + "/bika_setup"
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager', 'LabManager',])
```

4.17.2 Create Analysis Requests (AR)

An *AnalysisRequest* can only be created inside a *Client*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="NARALABS", ClientID="JG")
>>> client
<Client at /plone/clients/client-1>
```

To create a new AR, a *Contact* is needed:

```
>>> contact = api.create(client, "Contact", Firstname="Juan", Surname="Gallostra")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
```

A *SampleType* defines how long the sample can be retained, the minimum volume needed, if it is hazardous or not, the point where the sample was taken etc.:

```
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="water",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

A *SamplePoint* defines the location, where a *Sample* was taken:

```
>>> samplepoint = api.create(bika_samplepoints, "SamplePoint", title="Lake of␣
↪Constance")
>>> samplepoint
<SamplePoint at /plone/bika_setup/bika_samplepoints/samplepoint-1>
```

An *AnalysisCategory* categorizes different *AnalysisServices*:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
```

An *AnalysisService* defines a analysis service offered by the laboratory:

```
>>> analysisservice = api.create(bika_analysisservices, "AnalysisService", title="PH",
↪ ShortTitle="ph", Category=analysiscategory, Keyword="PH")
>>> analysisservice
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
```

Finally, the *AnalysisRequest* can be created:

```
>>> values = {
...            'Client': client.UID(),
...            'Contact': contact.UID(),
...            'SamplingDate': date_now,
...            'DateSampled': date_now,
...            'SampleType': sampletype.UID(),
...            'Priority': '1',
...           }
```

```
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/water-0001>
```

Also, make sure that the Analysis Request only has one analysis. You will see why later:

```
>>> len(ar.getAnalyses())
1
```

4.17.3 Submit Analyses results for the current Analysis Request

First transition the Analysis Request to received:

```
>>> transitioned = do_action_for(ar, 'receive')
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Set the results of the Analysis and transition them for verification:

```
>>> for analysis in ar.getAnalyses(full_objects=True):
...     analysis.setResult('12')
...     transitioned = do_action_for(analysis, 'submit')
>>> transitioned[0]
True
```

Check that both the Analysis Request and its analyses have been transitioned to 'to_be_verified':

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

(continues on next page)

(continued from previous page)

```
>>> not_to_be_verified = 0
>>> for analysis in ar.getAnalyses(full_objects=True):
...     if api.get_workflow_status_of(analysis) != 'to_be_verified':
...         not_to_be_verified += 1
>>> not_to_be_verified
0
```

4.17.4 Verify Analyses results for the current Analysis Request

Same user cannot verify by default:

```
>>> ar.bika_setup.setSelfVerificationEnabled(True)
```

Select all analyses from the Analysis Request and verify them:

```
>>> for analysis in ar.getAnalyses(full_objects=True):
...     transitioned = do_action_for(analysis, 'verify')
>>> transitioned[0]
True
```

Check that both the Analysis Request analyses have been transitioned to *verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
>>> not_verified = 0
>>> for analysis in ar.getAnalyses(full_objects=True):
...     if api.get_workflow_status_of(analysis) != 'verified':
...         not_verified += 1
>>> not_verified
0
```

4.17.5 Invalidate the Analysis Request

When an Analysis Request is invalidated two things should happen:

1- The Analysis Request is transitioned to 'invalid'. Analyses remain in *verified* state.

2- A new Analysis Request (retest) is created automatically, with same analyses as the invalidated, but in *sample_received* state.

Invalidate the Analysis Request:

```
>>> transitioned = do_action_for(ar, 'invalidate')
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'invalid'
>>> ar.isInvalid()
True
```

Verify a new Analysis Request (retest) has been created, with same analyses as the invalidated:

```
>>> retest = ar.getRetest()
>>> retest
<AnalysisRequest at /plone/clients/client-1/water-0001-R01>
```

```
>>> retest.getInvalidated()
<AnalysisRequest at /plone/clients/client-1/water-0001>
```

```
>>> api.get_workflow_status_of(retest)
'sample_received'
```

```
>>> retest_ans = map(lambda an: an.getKeyword(), retest.getAnalyses(full_
↪objects=True))
>>> invalid_ans = map(lambda an: an.getKeyword(), ar.getAnalyses(full_objects=True))
>>> len(set(retest_ans)-set(invalid_ans))
0
```

4.17.6 Invalidate the retest

We can even invalidate the retest generated previously. As a result, a new retest will be created.

First, submit all analyses from the retest:

```
>>> for analysis in retest.getAnalyses(full_objects=True):
...     analysis.setResult(12)
...     transitioned = do_action_for(analysis, 'submit')
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(retest)
'to_be_verified'
```

Now, verify all analyses from the retest:

```
>>> for analysis in retest.getAnalyses(full_objects=True):
...     transitioned = do_action_for(analysis, 'verify')
>>> transitioned[0]
True
```

```
>>> not_verified = 0
>>> for analysis in retest.getAnalyses(full_objects=True):
...     if api.get_workflow_status_of(analysis) != 'verified':
...         not_verified += 1
>>> not_verified
0
```

```
>>> api.get_workflow_status_of(retest)
'verified'
```

Invalidate the Retest:

```
>>> transitioned = do_action_for(retest, 'invalidate')
>>> transitioned[0]
True
>>> api.get_workflow_status_of(retest)
'invalid'
>>> retest.isInvalid()
True
```

Verify a new Analysis Request (retest 2) has been created, with same analyses as the invalidated (retest):

```
>>> retest2 = retest.getRetest()
>>> retest2
<AnalysisRequest at /plone/clients/client-1/water-0001-R02>
```

```
>>> retest2.getInvalidated()
<AnalysisRequest at /plone/clients/client-1/water-0001-R01>
```

```
>>> retest2.getInvalidated().getInvalidated()
<AnalysisRequest at /plone/clients/client-1/water-0001>
```

```
>>> api.get_workflow_status_of(retest2)
'sample_received'
```

```
>>> not_registered = 0
>>> for analysis in retest2.getAnalyses(full_objects=True):
...     if api.get_workflow_status_of(analysis) != 'unassigned':
...         registered += 1
>>> not_registered
0
```

```
>>> retest_ans = map(lambda an: an.getKeyword(), retest2.getAnalyses(full_
↪objects=True))
>>> invalid_ans = map(lambda an: an.getKeyword(), retest.getAnalyses(full_
↪objects=True))
>>> len(set(retest_ans)-set(invalid_ans))
0
```

4.18 Analysis Request retract

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t AnalysisRequestRetract
```

4.18.1 Test Setup

Needed Imports:

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone import api as ploneapi

>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

(continues on next page)

(continued from previous page)

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_labcontacts = bika_setup.bika_labcontacts
>>> bika_storagelocations = bika_setup.bika_storagelocations
>>> bika_samplingdeviations = bika_setup.bika_samplingdeviations
>>> bika_sampleconditions = bika_setup.bika_sampleconditions
>>> portal_url = portal.absolute_url()
>>> bika_setup_url = portal_url + "/bika_setup"
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.18.2 Create Analysis Requests (AR)

An *AnalysisRequest* can only be created inside a *Client*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="NARALABS", ClientID="JG")
>>> client
<Client at /plone/clients/client-1>
```

To create a new AR, a *Contact* is needed:

```
>>> contact = api.create(client, "Contact", Firstname="Juan", Surname="Gallostra")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
```

A *SampleType* defines how long the sample can be retained, the minimum volume needed, if it is hazardous or not, the point where the sample was taken etc.:

```
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="water",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

A *SamplePoint* defines the location, where a *Sample* was taken:

```
>>> samplepoint = api.create(bika_samplepoints, "SamplePoint", title="Lake of␣
↪Constance")
>>> samplepoint
<SamplePoint at /plone/bika_setup/bika_samplepoints/samplepoint-1>
```

An *AnalysisCategory* categorizes different *AnalysisServices*:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
```

An *AnalysisService* defines a analysis service offered by the laboratory:

```
>>> analysisservice = api.create(bika_analysisservices, "AnalysisService", title="PH",
↪ ShortTitle="ph", Category=analysiscategory, Keyword="PH")
>>> analysisservice
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
```

Finally, the *AnalysisRequest* can be created:

```
>>> values = {
...            'Client': client.UID(),
...            'Contact': contact.UID(),
...            'SamplingDate': date_now,
...            'DateSampled': date_now,
...            'SampleType': sampletype.UID(),
...            'Priority': '1',
...           }

>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/water-0001>
```

Also, make sure that the Analysis Request only has one analysis. You will see why later:

```
>>> len(ar.getAnalyses())
1
```

4.18.3 Submit Analyses results for the current Analysis Request

First transition the Analysis Request to received:

```
>>> transitioned = do_action_for(ar, 'receive')
>>> transitioned[0]
True
>>> ar.portal_workflow.getInfoFor(ar, 'review_state')
'sample_received'
```

Set the results of the Analysis and transition them for verification:

```
>>> for analysis in ar.getAnalyses(full_objects=True):
...     analysis.setResult('12')
...     transitioned = do_action_for(analysis, 'submit')
```

(continues on next page)

(continued from previous page)

```
>>> transitioned[0]
True
```

Check that both the Analysis Request and its analyses have been transitioned to 'to_be_verified':

```
>>> ar.portal_workflow.getInfoFor(ar, 'review_state')
'to_be_verified'
>>> not_to_be_verified = 0
>>> for analysis in ar.getAnalyses(full_objects=True):
...     if analysis.portal_workflow.getInfoFor(analysis, 'review_state') != 'to_be_
↪verified':
...         not_to_be_verified += 1
>>> not_to_be_verified
0
```

4.18.4 Retract the Analysis Request

When an Analysis Request is retracted two things should happen:

1- The Analysis Request is transitioned to 'sample_received'. Since the results have been retracted its review state goes back to just before the submission of results.

2- Its current analyses are transitioned to 'retracted' and a duplicate of each analysis is created (so that results can be introduced again) with review state 'sample_received'.

Retract the Analysis Request:

```
>>> transitioned = do_action_for(ar, 'retract')
>>> transitioned[0]
True
>>> ar.portal_workflow.getInfoFor(ar, 'review_state')
'sample_received'
```

Verify that its analyses have also been retracted and that a new analysis has been created with review status 'unassigned'. Since we previously checked that the AR had only one analyses the count for both 'retracted' and 'unassigned' analyses should be one:

```
>>> registered = 0
>>> retracted = 0
>>> for analysis in ar.getAnalyses(full_objects=True):
...     if analysis.portal_workflow.getInfoFor(analysis, 'review_state') == 'retracted
↪':
...         retracted += 1
...     if analysis.portal_workflow.getInfoFor(analysis, 'review_state') !=
↪'unassigned':
...         registered += 1
>>> registered
1
>>> retracted
1
```

4.19 Analysis Requests

Analysis Requests in Bika LIMS describe an Analysis Order from a Client to the Laboratory. Each Analysis Request manages a Sample, which holds the data of the physical Sample from the Client. The Sample is currently not handled by its own in Bika LIMS. So the managing Analysis Request is the primary interface from the User (Client) to the Sample.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t AnalysisRequests
```

4.19.1 Test Setup

Needed Imports:

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone import api as ploneapi

>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.api import do_transition_for
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)

>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_labcontacts = bika_setup.bika_labcontacts
>>> bika_storagelocations = bika_setup.bika_storagelocations
>>> bika_samplingdeviations = bika_setup.bika_samplingdeviations
>>> bika_sampleconditions = bika_setup.bika_sampleconditions
>>> portal_url = portal.absolute_url()
>>> bika_setup_url = portal_url + "/bika_setup"
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.19.2 Analysis Requests (AR)

An *AnalysisRequest* can only be created inside a *Client*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="RIDING BYTES", ClientID="RB")
>>> client
<Client at /plone/clients/client-1>
```

To create a new AR, a *Contact* is needed:

```
>>> contact = api.create(client, "Contact", Firstname="Ramon", Surname="Bartl")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
```

A *SampleType* defines how long the sample can be retained, the minimum volume needed, if it is hazardous or not, the point where the sample was taken etc.:

```
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="water",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

A *SamplePoint* defines the location, where a *Sample* was taken:

```
>>> samplepoint = api.create(bika_samplepoints, "SamplePoint", title="Lake of␣
↪Constance")
>>> samplepoint
<SamplePoint at /plone/bika_setup/bika_samplepoints/samplepoint-1>
```

An *AnalysisCategory* categorizes different *AnalysisServices*:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
```

An *AnalysisService* defines a analysis service offered by the laboratory:

```
>>> analysisservice = api.create(bika_analysisservices, "AnalysisService", title="PH",
↪ ShortTitle="ph", Category=analysiscategory, Keyword="PH")
>>> analysisservice
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
```

Finally, the *AnalysisRequest* can be created:

```
>>> values = {
...           'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': date_now,
...           'DateSampled': date_now,
```

(continues on next page)

(continued from previous page)

```
...             'SampleType': sampletype.UID(),
...             'Priority': '1',
...            }

>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/water-0001>
>>> ar.getPriority()
'1'
>>> ar.getPriorityText()
u'Highest'
```

4.19.3 DateReceived field should be editable in Received state

For this we need an AR with more than one Analysis:

```
>>> from bika.lims.adapters.widgetvisibility import␣
↪DateReceivedFieldVisibility
>>> from bika.lims.workflow import doActionFor
```

```
>>> as2 = api.create(bika_analysisservices, 'AnalysisService', title=
↪'Another Type Of Analysis', ShortTitle='Another',␣
↪Category=analysiscategory, Keyword='AN')
>>> ar1 = create_analysisrequest(client, request, values, service_uids +␣
↪[as2.UID()])
```

In states earlier than *sample_received* the DateReceived field is uneditable:

```
>>> field = ar1.getField("DateReceived")
>>> field.checkPermission("edit", ar1) and True or False
False
```

In the *sample_received* state however, it is possible to modify the field. In this case the SampleDateReceived adapter also simply passes the schema default unmolested.

```
>>> p = api.do_transition_for(ar1, 'receive')
>>> field = ar1.getField("DateReceived")
>>> field.checkPermission("edit", ar1) and True or False
True
```

After any analysis has been submitted, the field is no longer editable. The adapter sets the widget.visible to 'invisible'.

```
>>> an = ar1.getAnalyses(full_objects=True)[0]
>>> an.setResult('1')
>>> p = doActionFor(an, 'submit')
>>> DateReceivedFieldVisibility(ar1)(ar1, 'edit', ar1.schema['DateReceived'],
↪ 'default')
'invisible'
```

4.20 Analysis Service - Activations and Inactivations

The inactivation and activation of Analysis Services relies on *senaite_deactivable_type_workflow*. To prevent inconsistencies that could have undesired effects, an Analysis Service can only be deactivated if it does not have active dependents (this is, other services that depends on the Analysis Service to calculate their results).

Following the same reasoning, an Analysis Service can only be activated if does not have any calculation assigned or if does, the calculation is active, as well as its dependencies (this is, other services the Analysis Service depends on to calculate its result) are active .

4.21 Test Setup

Running this test from the buildout directory:

bin/test -t AnalysisServiceInactivation

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.workflow import doActionFor
>>> from bika.lims.workflow import getAllowedTransitions
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> bika_analysiscategories = bikasetup.bika_analysiscategories
>>> bika_analysisservices = bikasetup.bika_analysisservices
>>> bika_calculations = bikasetup.bika_calculations
>>> bika_suppliers = bikasetup.bika_suppliers
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bika_analysiscategories, "AnalysisCategory", title="Metals",
↪ Department=department)
>>> supplier = api.create(bika_suppliers, "Supplier", Name="Naralabs")
>>> Ca = api.create(bika_analysisservices, "AnalysisService", title="Calcium",␣
↪Keyword="Ca", Price="15", Category=category.UID())
```

(continues on next page)

(continued from previous page)

```
>>> Mg = api.create(bika_analysisservices, "AnalysisService", title="Magnesium",␣
↪Keyword="Mg", Price="10", Category=category.UID())
>>> Au = api.create(bika_analysisservices, "AnalysisService", title="Gold", Keyword=
↪"Au", Price="20", Category=category.UID())
```

4.21.1 Deactivation of Analysis Service

All services can be deactivated:

```
>>> getAllowedTransitions(Ca)
['deactivate']
>>> getAllowedTransitions(Mg)
['deactivate']
>>> getAllowedTransitions(Au)
['deactivate']
```

But if we create a new Analysis Service with a calculation that depends on them:

```
>>> calc = api.create(bika_calculations, "Calculation", title="Total Hardness")
>>> calc.setFormula("[Ca] + [Mg]")
>>> hardness = api.create(bika_analysisservices, "AnalysisService", title="Total␣
↪Hardness", Keyword="TotalHardness")
>>> hardness.setCalculation(calc)
```

Then, only *Au* can be deactivated, cause *harndess* is active and depends on *Ca* and *Mg*:

```
>>> getAllowedTransitions(Ca)
[]
>>> getAllowedTransitions(Mg)
[]
>>> getAllowedTransitions(Au)
['deactivate']
>>> getAllowedTransitions(hardness)
['deactivate']
```

If we deactivate *Hardness*:

```
>>> performed = doActionFor(hardness, 'deactivate')
>>> api.is_active(hardness)
False
```

```
>>> getAllowedTransitions(hardness)
['activate']
```

Then we will be able to deactivate both *Ca* and *Mg*:

```
>>> getAllowedTransitions(Ca)
['deactivate']
>>> getAllowedTransitions(Mg)
['deactivate']
```

4.21.2 Activation of Analysis Service

Deactivate the Analysis Service *Ca*:

```
>>> performed = doActionFor(Ca, 'deactivate')
>>> api.is_active(Ca)
False
```

```
>>> getAllowedTransitions(Ca)
['activate']
```

And now, we cannot activate *Hardness*, cause one of its dependencies (*Ca*) is not active:

```
>>> api.is_active(hardness)
False
>>> getAllowedTransitions(hardness)
[]
```

But if we activate *Ca* again:

```
>>> performed = doActionFor(Ca, 'activate')
>>> api.is_active(Ca)
True
```

Hardness can be activated again:

```
>>> getAllowedTransitions(hardness)
['activate']
```

```
>>> performed = doActionFor(hardness, 'activate')
>>> api.is_active(hardness)
True
```

4.22 Analysis Turnaround Time

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t AnalysisTurnaroundTime
```

4.22.1 Test Setup

Needed Imports:

```
>>> import re
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.api.analysis import get_formatted_interval
>>> from bika.lims.api.analysis import is_out_of_range
>>> from bika.lims.content.analysisrequest import AnalysisRequest
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils import tmpID
>>> from bika.lims.workflow import doActionFor
>>> from bika.lims.workflow import getCurrentState
>>> from bika.lims.workflow import getAllowedTransitions
>>> from bika.lims.workflow import getReviewHistory
>>> from DateTime import DateTime
```

(continues on next page)

(continued from previous page)

```
>>> from datetime import timedelta
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
>>> from Products.ATContentTypes.utils import DT2dt, dt2DT
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def change_receive_date(ar, days):
...     prev_date = ar.getDateReceived()
...     ar.Schema().getField('DateReceived').set(ar, prev_date + days)
...     for analysis in ar.getAnalyses(full_objects=True):
...         an_created = analysis.created()
...         analysis.getField('creation_date').set(analysis, an_created + days)
```

```
>>> def compute_due_date(analysis):
...     start = DT2dt(analysis.getStartProcessDate())
...     tat = api.to_minutes(**analysis.getMaxTimeAllowed())
...     due_date = start + timedelta(minutes=tat)
...     return dt2DT(due_date)
```

```
>>> def compute_duration(date_from, date_to):
...     return (date_to - date_from) * 24 * 60
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), DuplicateVariation="0.5")
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID(), DuplicateVariation="0.5")
```

(continues on next page)

(continued from previous page)

```
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID(), DuplicateVariation="0.5")
>>> Mg = api.create(bikasetup.bika_analysisservices, "AnalysisService", title=
↪"Magnesium", Keyword="Mg", Price="20", Category=category.UID(), DuplicateVariation=
↪"0.5")
>>> service_uids = [api.get_uid(an) for an in [Cu, Fe, Au, Mg]]
>>> sampletype_uid = api.get_uid(sampletype)
```

Set different Turnaround Times for every single Analysis Service:

```
>>> Au.setMaxTimeAllowed(dict(days=2, hours=8, minutes=30))
>>> maxtime = Au.getMaxTimeAllowed()
>>> [maxtime.get("days"), maxtime.get("hours"), maxtime.get("minutes")]
[2, 8, 30]
```

```
>>> Cu.setMaxTimeAllowed(dict(days=1, hours=4, minutes=0))
>>> maxtime = Cu.getMaxTimeAllowed()
>>> [maxtime.get("days"), maxtime.get("hours"), maxtime.get("minutes")]
[1, 4, 0]
```

```
>>> Fe.setMaxTimeAllowed(dict(days=3, hours=0, minutes=0))
>>> maxtime = Fe.getMaxTimeAllowed()
>>> [maxtime.get("days"), maxtime.get("hours"), maxtime.get("minutes")]
[3, 0, 0]
```

And leave Magnesium (Mg) without any Turnaround Time set, so it will use the default Turnaround time set in setup:

```
>>> maxtime = bikasetup.getDefaultTurnaroundTime()
>>> [maxtime.get("days"), maxtime.get("hours"), maxtime.get("minutes")]
[5, 0, 0]
```

```
>>> maxtime = Mg.getMaxTimeAllowed()
>>> [maxtime.get("days"), maxtime.get("hours"), maxtime.get("minutes")]
[5, 0, 0]
```

Create an Analysis Request:

```
>>> values = {
...     'Client': api.get_uid(client),
...     'Contact': api.get_uid(contact),
...     'DateSampled': date_now,
...     'SampleType': sampletype_uid,
...     'Priority': '1',
... }
```

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
```

Get the Analyses for further use:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> analyses = sorted(analyses, key=lambda an: an.getKeyword())
>>> map(lambda an: an.getKeyword(), analyses)
['Au', 'Cu', 'Fe', 'Mg']
>>> analyses_dict = {an.getKeyword(): an for an in analyses}
```

4.22.2 Test basic functions related with TAT

Since we haven't received the Analysis Request yet, the duration (time in minutes taken for analyses must be zero):

```
>>> map(lambda an: an.getStartProcessDate(), analyses)
[None, None, None, None]
```

```
>>> map(lambda an: an.getDuration(), analyses)
[0, 0, 0, 0]
```

So Due Date returns empty:

```
>>> map(lambda an: an.getDueDate(), analyses)
[None, None, None, None]
```

And none of the analyses are late:

```
>>> map(lambda an: an.isLateAnalysis(), analyses)
[False, False, False, False]
```

And Earliness (in minutes) matches with the TAT assigned to each analysis:

```
>>> map(lambda an: api.to_minutes(**an.getMaxTimeAllowed()), analyses)
[3390, 1680, 4320, 7200]
>>> map(lambda an: an.getEarliness(), analyses)
[3390, 1680, 4320, 7200]
```

Receive the Analysis Request:

```
>>> success = doActionFor(ar, 'receive')
```

The process date now for analyses is the received date:

```
>>> start_process = map(lambda an: an.getStartProcessDate(), analyses)
>>> received = map(lambda an: an.getDateReceived(), analyses)
>>> received == start_process
True
```

Also, the Analysis Request is not late because none of its analyses is late:

```
>>> ar.getLate()
False
```

4.22.3 Test TAT with analyses received 2d ago

We manually force a receive date 2d before so we can test:

```
>>> new_received = map(lambda rec: rec-2, received)
>>> change_receive_date(ar, -2)
>>> received = map(lambda an: an.getDateReceived(), analyses)
>>> start_process = map(lambda an: an.getStartProcessDate(), analyses)
>>> new_received == received == start_process
True
```

Analyses Au and Fe are not late, but Cu is late:

```
>>> map(lambda an: an.isLateAnalysis(), analyses)
[False, True, False, False]
```

Check Due Dates:

```
>>> expected_due_dates = map(lambda an: compute_due_date(an), analyses)
>>> due_dates = map(lambda an: an.getDueDate(), analyses)
>>> due_dates == expected_due_dates
True
```

And duration:

```
>>> expected = map(lambda an: int(compute_duration(an.getStartProcessDate(),␣
↪DateTime())), analyses)
>>> durations = map(lambda an: int(an.getDuration()), analyses)
>>> expected == durations
True
```

Earliness in minutes. Note the value for Cu is negative (is late), and the value for Mg is 0 (no Turnaround Time) set:

```
>>> map(lambda an: int(round(an.getEarliness())), analyses)
[510, -1200, 1440, 4320]
```

Lateness in minutes. Note that all values are negative except for Cu:

```
>>> map(lambda an: int(round(an.getLateness())), analyses)
[-510, 1200, -1440, -4320]
```

Because one of the analyses (Cu) is late, the Analysis Request is late too:

```
>>> ar.getLate()
True
```

4.23 Analysis publication guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t BatchClientAssignment

4.23.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from zope.lifecycleevent import modified
```

Variables and basic objects for the test:

```
>>> portal = self.portal
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
```

(continues on next page)

(continued from previous page)

4.23.2 Batch creation and Client assignment

Create a new Batch:

```
>>> batches = portal.batches
>>> batch = api.create(batches, "Batch", title="Test batch")
>>> batch.aq_parent
<BatchFolder at /plone/batches>
```

The batches folder contains the batch, while Client folder remains empty:

```
>>> len(batches.objectValues("Batch"))
1
>>> len(client.objectValues("Batch"))
0
```

Assign a client to the Batch and the latter is automatically moved inside Client's folder:

```
>>> batch.setClient(client)
>>> modified(batch)
>>> len(batches.objectValues("Batch"))
0
>>> len(client.objectValues("Batch"))
1
```

If the client is assigned on creation, same behavior as before:

```
>>> batch = api.create(portal.batches, "Batch", Client=client)
>>> len(batches.objectValues("Batch"))
0
>>> len(client.objectValues("Batch"))
2
```

4.24 Calculations

Bika LIMS can dynamically calculate a value based on the results of several Analyses with a formula.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t Calculations
```

4.24.1 Test Setup

Needed Imports:

```
>>> import transaction
>>> from operator import methodcaller
>>> from plone import api as ploneapi

>>> from bika.lims import api
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_calculations = bika_setup.bika_calculations
>>> bika_analysisservices = bika_setup.bika_analysisservices
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.24.2 Calculation

Calculations are created in the *bika_setup/bika_calculations* folder. They offer a *Formula* field, where keywords from Analyses can be used to calculate a result.

Each *AnalysisService* contains a *Keyword* field, which can be referenced in a formula:

```
>>> as1 = api.create(bika_analysisservices, "AnalysisService", title="Calcium")
>>> as1.setKeyword("Ca")
>>> as1.reindexObject()

>>> as2 = api.create(bika_analysisservices, "AnalysisService", title="Magnesium")
>>> as2.setKeyword("Mg")
>>> as2.reindexObject()
```

Create one *Calculation*:

```
>>> calc = api.create(bika_calculations, "Calculation", title="Total Hardness")
```

The *Formula* field references the Keywords from Analysis Services:

```
>>> calc.setFormula("[Ca] + [Mg]")
>>> calc.getFormula()
'[Ca] + [Mg]'

>>> calc.getMinifiedFormula()
'[Ca] + [Mg]'
```

The *Calculation* depends now on the two Analysis Services:

```
>>> sorted(calc.getCalculationDependencies(flat=True), key=methodcaller('getId'))
[<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>,
 ↪<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>]
```

It is also possible to find out if an *AnalysisService* depends on the calculation:

```
>>> as1.setCalculation(calc)
>>> calc.getCalculationDependants()
[<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>]
```

Or to find out which services have selected a particular calculation as their primary Calculation field's value:

```
>>> from bika.lims.browser.fields.uidreferencefield import get_backreferences
>>> get_backreferences(calc, 'AnalysisServiceCalculation')
['...']
```

The *Formula* can be tested with dummy values in the *TestParameters* field:

```
>>> form_value = [{"keyword": "Ca", "value": 5.6}, {"keyword": "Mg", "value": 3.3},]
>>> calc.setTestParameters(form_value)
>>> calc.setTestResult(form_value)
>>> calc.getTestResult()
'8.9'
```

Within a *Calculation* it is also possible to use a Python function to calculate a result. The user can add a Python *module* as a dotted name and a member function in the *PythonImports* field:

```
>>> calc.setPythonImports([{'module': 'math', 'function': 'floor'}])
>>> calc.setFormula("floor([Ca] + [Mg])")
>>> calc.getFormula()
'floor([Ca] + [Mg])'

>>> calc.setTestResult(form_value)
>>> calc.getTestResult()
'8.0'
```

A *Calculation* can therefore dynamically get a module and a member:

```
>>> calc._getModuleMember('math', 'ceil')
<built-in function ceil>
```

4.25 Cobas Integra 400+ import interface

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t CobasIntegra400plusImportInterface
```

4.25.1 Test Setup

Needed imports:

```
>>> import os
>>> import transaction
>>> from Products.CMFCore.utils import getToolByName
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from DateTime import DateTime
```

(continues on next page)

(continued from previous page)

```
>>> import codecs
>>> from senaite.core.exportimport import instruments
>>> from senaite.core.exportimport.instruments.cobasintegra.model_400_plus.model_400_
↪plus import CobasIntegra400plus2Importer
>>> from senaite.core.exportimport.instruments.cobasintegra.model_400_plus.model_400_
↪plus import CobasIntegra400plus2CSVParser
>>> from bika.lims.browser.resultsimport.resultsimport import ConvertToUploadFile
```

Functional helpers:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_instruments = bika_setup.bika_instruments
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.25.2 Availability of instrument interface

Check that the instrument interface is available::

```
>>> exims = []
>>> for exim_id in instruments.__all__:
...     exims.append(exim_id)
>>> 'cobasintegra.model_400_plus.model_400_plus' in exims
True
```

4.25.3 Assigning the Import Interface to an Instrument

Create an *Instrument* and assign to it the tested Import Interface:

```
>>> instrument = api.create(bika_instruments, "Instrument", title="Instrument-1")
>>> instrument
<Instrument at /plone/bika_setup/bika_instruments/instrument-1>
>>> instrument.setImportDataInterface(['cobasintegra.model_400_plus.model_400_plus'])
>>> instrument.getImportDataInterface()
['cobasintegra.model_400_plus.model_400_plus']
```

4.25.4 Import test

Required steps: Create and receive Analysis Request for import test

An *AnalysisRequest* can only be created inside a *Client*, and it also requires a *Contact* and a *SampleType*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="BHPLAB", ClientID="BLAB")
>>> client
<Client at /plone/clients/client-1>
>>> contact = api.create(client, "Contact", Firstname="Moffat", Surname="More")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="H2O",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

Create an *AnalysisCategory* (which categorizes different *AnalysisServices*), and add to it some of the *AnalysisServices* that are found in the results file:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
>>> analysisservice_1 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="WBC",
...                                ShortTitle="wbc",
...                                Category=analysiscategory,
...                                Keyword="WBC")
>>> analysisservice_1
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
>>> analysisservice_2 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="RBC",
...                                ShortTitle="rbc",
...                                Category=analysiscategory,
...                                Keyword="RBC")
>>> analysisservice_2
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>
>>> analysisservice_3 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="HGB",
...                                ShortTitle="hgb",
...                                Category=analysiscategory,
...                                Keyword="HGB")
>>> analysisservice_3
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-3>
>>> analysisservice_4 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="HCT",
...                                ShortTitle="hct",
...                                Category=analysiscategory,
...                                Keyword="HCT")
>>> analysisservice_4
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-4>
```

(continues on next page)

(continued from previous page)

```
>>> analysisservices = [analysisservice_1, analysisservice_2, analysisservice_3,␣
↪analysisservice_4]
```

Create an *AnalysisRequest* with this *AnalysisService* and receive it:

```
>>> values = {
...            'Client': client.UID(),
...            'Contact': contact.UID(),
...            'SamplingDate': date_now,
...            'DateSampled': date_now,
...            'SampleType': sampletype.UID()
...          }
>>> service_uids = [analysisservice.UID() for analysisservice in analysisservices]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/H2O-0001>
>>> ar.getReceivedBy()
''
>>> wf = getToolByName(ar, 'portal_workflow')
>>> wf.doActionFor(ar, 'receive')
>>> ar.getReceivedBy()
'test_user_1_'
```

Import test

Load results test file and import the results:

```
>>> dir_path = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..', 'files
↪'))
>>> temp_file = codecs.open(dir_path + '/cobasintegra.csv',
...                         encoding='utf-8-sig')
>>> test_file = ConvertToUploadFile(temp_file)
>>> cobasintegra_parser = CobasIntegra400plus2CSVParser(test_file)
>>> importer = CobasIntegra400plus2Importer(parser=cobasintegra_parser,
...                                         context=portal,
...                                         allowed_ar_states=['sample_received', 'attachment_due
↪', 'to_be_verified'],
...                                         allowed_analysis_states=None,
...                                         override=[True, True])
>>> importer.process()
```

Check from the importer logs that the file from where the results have been imported is indeed the specified file:

```
>>> 'cobasintegra.csv' in importer.logs[0]
True
```

Check the rest of the importer logs to verify that the values were correctly imported:

```
>>> importer.logs[1:]
['End of file reached successfully: 25 objects, 8 analyses, 112 results'...
```

4.26 Clients, Contacts and linked Users

Clients are the customers of the lab. A client represents another company, which has one or more natural persons as contacts.

Each contact can be linked to a Plone system user. The linking process adds the linked user to the "Clients" group, which has the "Customer" role.

Furthermore, the user gets the local "Owner" role for the owning client object.

Running this test from the buildout directory:

    bin/test -t ContactUser

4.27 Test Setup

```
>>> import transaction
>>> from plone import api as ploneapi
>>> from zope.lifecycleevent import modified
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

```
>>> portal = self.portal
>>> portal_url = portal.absolute_url()
>>> bika_setup = portal.bika_setup
>>> bika_setup_url = portal_url + "/bika_setup"
>>> browser = self.getBrowser()
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Manager', 'Owner'])
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def login(user=TEST_USER_ID, password=TEST_USER_PASSWORD):
...     browser.open(portal_url + "/login_form")
...     browser.getControl(name='__ac_name').value = user
...     browser.getControl(name='__ac_password').value = password
...     browser.getControl(name='buttons.login').click()
...     assert("__ac_password" not in browser.contents)
...     return ploneapi.user.get_current()
```

```
>>> def logout():
...     browser.open(portal_url + "/logout")
...     assert("You are now logged out" in browser.contents)
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

```
>>> def create(container, portal_type, title=None):
...     # Creates a content in a container and manually calls processForm
...     title = title is None and "Test {}".format(portal_type) or title
...     _ = container.invokeFactory(portal_type, id="tmpID", title=title)
...     obj = container.get(_)
...     obj.processForm()
...     modified(obj)  # notify explicitly for the test
...     transaction.commit()  # somehow the created method did not appear until I␣
↪added this
...     return obj
```

```
>>> def get_workflows_for(context):
...     # Returns a tuple of assigned workflows for the given context
...     workflow = ploneapi.portal.get_tool("portal_workflow")
...     return workflow.getChainFor(context)
```

```
>>> def get_workflow_status_of(context):
...     # Returns the workflow status of the given context
...     return ploneapi.content.get_state(context)
```

4.28 Client

A *client* lives in the */clients* folder:

```
>>> clients = portal.clients
>>> client1 = create(clients, "Client", title="Client-1")
>>> client2 = create(clients, "Client", title="Client-2")
```

4.29 Contact

A *contact* lives inside a *client*:

```
>>> contact1 = create(client1, "Contact", "Contact-1")
>>> contact2 = create(client2, "Contact", "Contact-2")
```

4.30 User

A *user* is able to login to the system.

Create a new user for the contact:

```
>>> user1 = ploneapi.user.create(email="contact-1@example.com", username="user-1",␣
↪password=TEST_USER_PASSWORD, properties=dict(fullname="Test User 1"))
>>> user2 = ploneapi.user.create(email="contact-2@example.com", username="user-2",␣
↪password=TEST_USER_PASSWORD, properties=dict(fullname="Test User 2"))
>>> transaction.commit()
```

4.30.1 Client Browser Test

Login with the first user:

```
>>> user = login(user1.id)
```

The user is not allowed to access any clients folder:

```
>>> browser.open(client1.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

Linking the user to a client contact grants access to this client:

```
>>> contact1.setUser(user1)
True
>>> transaction.commit()
```

Linking a user adds this user to the *Clients* group:

```
>>> clients_group = ploneapi.group.get("Clients")
>>> user1.getId() in clients_group.getAllGroupMemberIds()
True
```

This gives the user the global *Client* role:

```
>>> sorted(ploneapi.user.get_roles(user=user1))
['Authenticated', 'Client', 'Member']
```

It also grants local *Owner* role on the client object:

```
>>> sorted(user1.getRolesInContext(client1))
['Authenticated', 'Member', 'Owner']
```

The user is able to modify the *client* object properties:

```
>>> browser.open(client1.absolute_url() + "/base_edit")
>>> "edit_form" in browser.contents
True
```

As well as the *contact* object properties:

```
>>> browser.open(contact1.absolute_url() + "/base_edit")
>>> "edit_form" in browser.contents
True
```

But the user can not access other clients:

```
>>> browser.open(client2.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

Or modify other clients:

```
>>> browser.open(client2.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Unlink the user revokes all access to the client:

```
>>> contact1.unlinkUser()
True
>>> transaction.commit()
```

The user has no local owner role anymore on the client:

```
>>> sorted(user1.getRolesInContext(client1))
['Authenticated', 'Member']

>>> browser.open(client1.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

4.31 LabContact users

All non-client lab users should be created as Lab Contacts in site-setup:

~~ code

```
>>> labcontact = create(portal.bika_setup.bika_labcontacts, "LabContact")
```

And a new user for the labcontact:

~~ code

```
>>> user3 = ploneapi.user.create(email="labmanager@example.com", username="labmanager1
↪", password="secret", properties=dict(fullname="Lab Manager 1"))
```

Link the user to the labcontact:

~~ code

```
>>> labcontact.setUser(user3)
True
```

Linking a user to a LabContact does not give any client group membership:

```
>>> 'Client' in sorted(ploneapi.user.get_roles(user=user3)) and "Labcontact should␣
↪not have the Client role!" or False
False
```

4.31.1 Login Details View

The login details view manages to link/unlink users to contacts.

Get the *login_details* view for the first contact:

```
>>> login_details_view = contact1.restrictedTraverse("login_details")
```

The form expects a searchstring coming from the request. We fake it here:

```
>>> login_details_view.searchstring = ""
```

Search for linkable users:

```
>>> linkable_users = login_details_view.linkable_users()
>>> linkable_user_ids = map(lambda x: x.get("id"), linkable_users)
```

Both users should be now in the search results:

```
>>> user1.getId() in linkable_user_ids
True
```

```
>>> user2.id in linkable_user_ids
True
```

Users with higher roles should not be listed:

```
>>> setRoles(portal, "user-2", ['Member', 'Client', 'LabClerk'])
```

```
>>> linkable_users = login_details_view.linkable_users()
>>> linkable_user_ids = map(lambda x: x.get("id"), linkable_users)
```

```
>>> user2.id in linkable_user_ids
False
```

This contact is not linked to a user:

```
>>> contact1.hasUser()
False
```

Now we link a user over the view:

```
>>> login_details_view._link_user(user1.id)

>>> contact1.hasUser()
True
```

The search should now omit this user from the search, so that it can not be linked anymore:

```
>>> linkable_users = login_details_view.linkable_users()
>>> linkable_user_ids = map(lambda x: x.get("id"), linkable_users)
>>> user1.id in linkable_user_ids
False
```

4.32 Duplicate results range

The valid result range for a duplicate analysis is calculated by applying a duplicate variation percentage to the result from the original analysis. If the analysis has result options enabled or string results enabled, results from both duplicate and original analysis must match 100%.

Running this test from the buildout directory:

bin/test test_textual_doctests -t DuplicateResultsRange

4.32.1 Test Setup

Needed imports:

```
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from bika.lims import api
>>> from bika.lims.api.analysis import is_out_of_range
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def new_sample(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime().strftime("%Y-%m-%d"),
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def new_worksheet(analyses):
...     analyses = []
...     for num in range(num_analyses):
...         sample = new_sample(analyses)
...         analyses.extend(sample.getAnalyses(full_objects=True))
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     worksheet.addAnalyses(analyses)
...     return worksheet
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = api.get_setup()
```

Create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
```

(continues on next page)

(continued from previous page)

```
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID())
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.32.2 Duplicate of an analysis with numeric result

Set the duplicate variation in percentage for *Cu*:

```
>>> Cu.setDuplicateVariation("10")
>>> Cu.getDuplicateVariation()
'10.00'
```

Create a Sample and receive:

```
>>> sample = new_sample([Cu])
```

Create a worksheet and assign the analyses:

```
>>> analyses = sample.getAnalyses(full_objects=True)
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalyses(analyses)
```

Add a duplicate for analysis *Cu*:

```
>>> worksheet.addDuplicateAnalyses(1)
[<DuplicateAnalysis at /plone/worksheets/WS-001/...
```

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.getAnalysis()
<Analysis at /plone/clients/client-1/W-0001/Cu>
```

```
>>> duplicate.getResultsRange()
{}
```

Set a result of 50 for the original analysis *Cu*:

```
>>> cu = analyses[0]
>>> cu.setResult(50)
>>> duplicate.getAnalysis().getResult()
'50'
```

```
>>> result_range = duplicate.getResultsRange()
>>> (result_range.min, result_range.max)
('45.0', '55.0')
```

We can set a result for the duplicate within the range:

```
>>> duplicate.setResult(47)
>>> is_out_of_range(duplicate)
(False, False)
```

Or an out-of-range result:

```
>>> duplicate.setResult(42)
>>> is_out_of_range(duplicate)
(True, True)
```

We can do same exercise, but the other way round. We can submit the result for the duplicate first:

```
>>> sample = new_sample([Cu])
>>> cu = sample.getAnalyses(full_objects=True)[0]
>>> worksheet.addAnalyses([cu])
```

We add a duplicate for new analysis, that is located at slot number 3:

```
>>> worksheet.addDuplicateAnalyses(src_slot=3)
[<DuplicateAnalysis at /plone/worksheets/WS-001/...
```

```
>>> duplicate = worksheet.getDuplicateAnalyses()
>>> duplicate = filter(lambda dup: dup.getAnalysis() == cu, duplicate)[0]
>>> duplicate.getAnalysis()
<Analysis at /plone/clients/client-1/W-0002/Cu>
```

```
>>> duplicate.getResultsRange()
{}
```

We set the result for the duplicate first, but it does not have a valid result range because the original analysis has no result yet:

```
>>> duplicate.setResult(58)
>>> duplicate.getResultsRange()
{}
```

```
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> cu.setResult(50)
>>> result_range = duplicate.getResultsRange()
>>> (result_range.min, result_range.max)
('45.0', '55.0')
```

```
>>> is_out_of_range(duplicate)
(True, True)
```

4.32.3 Duplicate of an analysis with result options

Let's add some results options to service *Fe*:

```
>>> results_options = [
...     {"ResultValue": "1", "ResultText": "Number 1"},
...     {"ResultValue": "2", "ResultText": "Number 2"},
...     {"ResultValue": "3", "ResultText": "Number 3"}]
>>> Fe.setResultOptions(results_options)
>>> Fe.getResultOptions()
[{'ResultValue': '1', 'ResultText': 'Number 1'}, {'ResultValue': '2', 'ResultText':
↪'Number 2'}, {'ResultValue': '3', 'ResultText': 'Number 3'}]
```

Create a Sample and receive:

```
>>> sample = new_sample([Fe])
```

Create a worksheet and assign the analyses:

```
>>> analyses = sample.getAnalyses(full_objects=True)
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalyses(analyses)
```

Add a duplicate for analysis *Fe*:

```
>>> worksheet.addDuplicateAnalyses(1)
[<DuplicateAnalysis at /plone/worksheets/WS-002/...
```

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> fe = duplicate.getAnalysis()
>>> fe
<Analysis at /plone/clients/client-1/W-0003/Fe>
```

```
>>> duplicate.getResultsRange()
{}
```

Set a result for original analysis:

```
>>> fe.setResult(2)
>>> fe.getResult()
'2'
>>> fe.getFormattedResult()
'Number 2'
```

The result range for duplicate does not longer consider duplicate variation, rather expects an exact result:

```
>>> duplicate.getResultsRange()
{}
```

```
>>> duplicate.setResult(1)
>>> duplicate.getResult()
'1'
>>> duplicate.getFormattedResult()
'Number 1'
>>> duplicate.getResultsRange()
{}
>>> is_out_of_range(duplicate)
(True, True)
```

```
>>> duplicate.setResult(2)
>>> duplicate.getResultsRange()
{}
>>> is_out_of_range(duplicate)
(False, False)
```

```
>>> duplicate.setResult(3)
>>> duplicate.getResultsRange()
{}
>>> is_out_of_range(duplicate)
(True, True)
```

4.32.4 Duplicate of an analysis with string results enabled

Let's add make the analysis *Au* to accept string results:

```
>>> Au.setStringResult(True)
```

Create a Sample and receive:

```
>>> sample = new_sample([Au])
```

Create a worksheet and assign the analyses:

```
>>> analyses = sample.getAnalyses(full_objects=True)
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalyses(analyses)
```

Add a duplicate for analysis *Au*:

```
>>> worksheet.addDuplicateAnalyses(1)
[<DuplicateAnalysis at /plone/worksheets/WS-003/...
```

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> au = duplicate.getAnalysis()
>>> au
<Analysis at /plone/clients/client-1/W-0004/Au>
```

```
>>> duplicate.getStringResult()
True
```

```
>>> duplicate.getResultsRange()
{}
```

Submit a string result for original analysis:

```
>>> au.setResult("Positive")
>>> au.getResult()
'Positive'
```

```
>>> au.getFormattedResult()
'Positive'
```

The result range for duplicate does not longer consider duplicate variation, rather expects an exact result:

```
>>> duplicate.getResultsRange()
{}
```

```
>>> duplicate.setResult("Negative")
>>> duplicate.getResult()
'Negative'
>>> duplicate.getFormattedResult()
'Negative'
>>> duplicate.getResultsRange()
{}
>>> is_out_of_range(duplicate)
(True, True)
```

```
>>> duplicate.setResult("Positive")
>>> duplicate.getResultsRange()
{}
>>> is_out_of_range(duplicate)
(False, False)
```

But when we submit a numeric result for an analysis with string result enabled, the system will behave as if it was indeed, a numeric result:

```
>>> Au.setDuplicateVariation("10")
>>> Au.getDuplicateVariation()
'10.00'
```

```
>>> Au.getStringResult()
True
```

```
>>> sample = new_sample([Au])
>>> au = sample.getAnalyses(full_objects=True)[0]
>>> worksheet.addAnalyses([au])
```

We add a duplicate for new analysis, that is located at slot number 3:

```
>>> worksheet.addDuplicateAnalyses(src_slot=3)
[<DuplicateAnalysis at /plone/worksheets/WS-003/...
```

```
>>> duplicate = worksheet.getDuplicateAnalyses()
>>> duplicate = filter(lambda dup: dup.getAnalysis() == au, duplicate)[0]
>>> duplicate.getAnalysis()
<Analysis at /plone/clients/client-1/W-0005/Au>
```

```
>>> duplicate.getStringResult()
True
```

```
>>> duplicate.getResultsRange()
{}
```

And we set a numeric result:

```
>>> au.setResult(50)
>>> results_range = duplicate.getResultsRange()
>>> (results_range.min, results_range.max)
('45.0', '55.0')
```

4.33 Dynamic Analysis Specifications

A *Dynamic Analysis Specification* can be assigned to *Analysis Specifications*.

When retrieving the result ranges (specification) for an Analysis, a lookup is done on the *Dynamic Analysis Specification*.

4.33.1 Example

Given is an Excel with the following minimal set of columns:

——- ——— — — Keyword Method min max ——- ——— — — Ca Method A 1 2 Ca Method B 3 4 Mg Method A 5 6 Mg Method B 7 8 ——- ——— — —

This Excel is uploaded to an *Dynamic Analysis Specification* object, which is linked to an Analysis Specification for the Sample Type "Water".

A new "Water" Sample is created with an containing *H2O* analysis to be tested with *Method-2*. The results range selected will be *[3;4]*.

Running this test from the buildout directory:

bin/test test_textual_doctests -t DynamicAnalysisSpec.rst

4.33.2 Test Setup

Needed imports:

```
>>> from DateTime import DateTime
>>> from StringIO import StringIO
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from openpyxl import Workbook
>>> from openpyxl.writer.excel import save_virtual_workbook
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> from plone.namedfile.file import NamedBlobFile
>>> import csv
```

Some Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = api.get_setup()
```

Functional Helpers:

```
>>> def new_sample(services, specification=None, results_ranges=None):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime().strftime("%Y-%m-%d"),
...         'SampleType': sampletype.UID(),
...         'Analyses': map(api.get_uid, services),
...         'Specification': specification or None }
...
...     ar = create_analysisrequest(client, request, values)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

Privileges:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.33.3 Creating a Dynamic Analysis Specification

Dynamic Analysis Specifications are actually only small wrappers around an Excel file, where result ranges are defined per row.

Let's create first a small helper function that generates an Excel for us:

```
>>> def to_excel(data):
...     workbook = Workbook()
...     first_sheet = workbook.get_active_sheet()
...     reader = csv.reader(StringIO(data))
...     for row in reader:
...         first_sheet.append(row)
...     return NamedBlobFile(save_virtual_workbook(workbook))
```

Then we create the data according to the example given above:

```
>>> data = """Keyword,Method,min,max
... Ca,Method A,1,2
... Ca,Method B,3,4
... Mg,Method A,5,6
... Mg,Method B,7,8"""
```

Now we can create a Dynamic Analysis Specification Object:

```
>>> ds = api.create(setup.dynamic_analysisspecs, "DynamicAnalysisSpec")
>>> ds.specs_file = to_excel(data)
```

We can get now directly the parsed header:

```
>>> header = ds.get_header()
>>> header
[u'Keyword', u'Method', u'min', u'max']
```

And the result ranges:

```
>>> rr = ds.get_specs()
>>> map(lambda r: [r.get(k) for k in header], rr)
[[u'Ca', u'Method A', 1, 2], [u'Ca', u'Method B', 3, 4], [u'Mg', u'Method A', 5, 6],␣
↪[u'Mg', u'Method B', 7, 8]]
```

We can also get the specs by Keyword:

```
>>> mg_rr = ds.get_by_keyword()["Mg"]
>>> map(lambda r: [r.get(k) for k in header], mg_rr)
[[u'Mg', u'Method A', 5, 6], [u'Mg', u'Method B', 7, 8]]
```

4.33.4 Hooking in a Dynamic Analysis Specification

Dynamic Analysis Specifications can only be assigned to a default Analysis Specification.

First we build some basic setup structure:

```
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
```

(continues on next page)

(continued from previous page)

```
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
```

```
>>> method_a = api.create(portal.methods, "Method", title="Method A")
>>> method_b = api.create(portal.methods, "Method", title="Method B")
```

```
>>> Ca = api.create(setup.bika_analysisservices, "AnalysisService", title="Calcium",␣
↪Keyword="Ca", Category=category, Method=method_a)
>>> Mg = api.create(setup.bika_analysisservices, "AnalysisService", title="Magnesium",
↪ Keyword="Mg", Category=category, Method=method_a)
```

Then we create a default Analysis Specification:

```
>>> rr1 = {"keyword": "Ca", "min": 10, "max": 20, "warn_min": 9, "warn_max": 21}
>>> rr2 = {"keyword": "Mg", "min": 10, "max": 20, "warn_min": 9, "warn_max": 21}
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="H2O")
>>> specification = api.create(setup.bika_analysisspecs, "AnalysisSpec", title="Lab␣
↪Water Spec", SampleType=sampletype.UID(), ResultsRange=[rr1, rr2])
```

And create a new sample with the given Analyses and the Specification:

```
>>> services = [Ca, Mg]
>>> sample = new_sample(services, specification=specification)
>>> ca, mg = sample["Ca"], sample["Mg"]
```

The specification is according to the values we have set before:

```
>>> ca_spec = ca.getResultsRange()
>>> ca_spec["min"], ca_spec["max"]
(10, 20)
```

```
>>> mg_spec = mg.getResultsRange()
>>> mg_spec["min"], mg_spec["max"]
(10, 20)
```

Now we hook in our Dynamic Analysis Specification to the standard Specification:

```
>>> specification.setDynamicAnalysisSpec(ds)
```

The specification need to get unset/set again, so that the dynamic values get looked up:

```
>>> sample.setSpecification(None)
>>> sample.setSpecification(specification)
```

The specification of the *Ca* Analysis with the Method *Method A*:

```
>>> ca_spec = ca.getResultsRange()
>>> ca_spec["min"], ca_spec["max"]
(1, 2)
```

Now let's change the *Ca* Analysis Method to *Method B*:

```
>>> ca.setMethod(method_b)
```

Unset and set the specification again:

```
>>> sample.setSpecification(None)
>>> sample.setSpecification(specification)
```

And get the results range again:

```
>>> ca_spec = ca.getResultsRange()
>>> ca_spec["min"], ca_spec["max"]
(3, 4)
```

The same now with the *Mg* Analysis in one run:

```
>>> mg_spec = mg.getResultsRange()
>>> mg_spec["min"], mg_spec["max"]
(5, 6)
```

```
>>> mg.setMethod(method_b)
```

Unset and set the specification again:

```
>>> sample.setSpecification(None)
>>> sample.setSpecification(specification)
```

```
>>> mg_spec = mg.getResultsRange()
>>> mg_spec["min"], mg_spec["max"]
(7, 8)
```

4.34 History Aware Reference Field

This field behaves almost the same like the standard AT ReferenceField, but stores the version of the reference object on *set* and keeps that version.

Currently, only analyses uses that field to store the exact version of their calculation. This ensures that later changes in, e.g. the formula, does not affect already created analyses.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t HistoryAwareReferenceField
```

4.34.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.api.security import *
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
```

(continues on next page)

(continued from previous page)

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_sample(services):
...     values = {
...         "Client": client.UID(),
...         "Contact": contact.UID(),
...         "DateSampled": date_now,
...         "SampleType": sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     return create_analysisrequest(client, request, values, service_uids)
```

```
>>> def get_analysis(sample, id):
...     ans = sample.getAnalyses(getId=id, full_objects=True)
...     if len(ans) != 1:
...         return None
...     return ans[0]
```

4.34.2 Environment Setup

Setup the testing environment:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> setRoles(portal, TEST_USER_ID, ['LabManager', ])
>>> user = api.get_current_user()
```

4.34.3 LIMS Setup

Setup the Lab for testing:

```
>>> setup.setSelfVerificationEnabled(True)
>>> analysisservices = setup.bika_analysisservices
>>> calculations = setup.bika_calculations
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH")
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="Water")
```

4.34.4 Content Setup

Create some Analysis Services with unique Keywords:

```
>>> Ca = api.create(analysisservices, "AnalysisService", title="Calcium", Keyword="Ca
↪")
>>> Mg = api.create(analysisservices, "AnalysisService", title="Magnesium", Keyword=
↪"Mg")
>>> TH = api.create(analysisservices, "AnalysisService", title="Total Hardness",␣
↪Keyword="TH")
```

Create a calculation for Total Hardness:

```
>>> calc = api.create(calculations, "Calculation", title="Total Hardness")
```

The *Formula* field references the keywords from Analysis Services:

```
>>> calc.setFormula("[Ca] + [Mg]")
>>> calc.processForm()

>>> calc.getFormula()
'[Ca] + [Mg]'

>>> calc.getMinifiedFormula()
'[Ca] + [Mg]'
```

Set the calculation to the *TH* analysis service:

```
>>> TH.setCalculation(calc)
```

Create an new Sample:

```
>>> sample = new_sample([Ca, Mg, TH])
```

Get the *TH* analysis:

```
>>> th = get_analysis(sample, "TH")
```

The calculation of the analysis should be unchanged:

```
>>> th_calc = th.getCalculation()
>>> th_calc.getFormula()
'[Ca] + [Mg]'
```

Now we change the calculation formula:

```
>>> calc.setFormula("2 * ([Ca] + [Mg])")
>>> calc.getFormula()
'2 * ([Ca] + [Mg])'
>>> calc.processForm()
```

The calculation of the analysis should be unchanged:

```
>>> th_calc = th.getCalculation()
>>> th_calc.getFormula()
'[Ca] + [Mg]'
```

4.35 ID Server

The ID Server in SENAITE LIMS provides IDs for content items base of the given format specification. The format string is constructed in the same way as a python format() method based predefined variables per content type. The only variable available to all type is 'seq'. Currently, seq can be constructed either using number generator or a counter of existing items. For generated IDs, one can specifypoint at which the format string will be split to create the generator key. For counter IDs, one must specify context and the type of counter which is either the number of backreferences or the number of contained objects.

Configuration Settings: * format:

- a python format string constructed from predefined variables like client, sampleType.
- special variable 'seq' must be positioned last in the format string
- sequence type: [generated|counter]
- context: if type counter, provides context the counting function
- counter type: [backreference|contained]
- counter reference: a parameter to the counting function
- prefix: default prefix if none provided in format string
- split length: the number of parts to be included in the prefix

ToDo: * validation of format strings

Running this test from the buildout directory:

```
bin/test -t IDServer
```

4.35.1 Test Setup

Needed Imports:

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
>>> from zope.component import getUtility
```

```
>>> from bika.lims import alphanumber as alpha
>>> from bika.lims import api
>>> from bika.lims import idserver
>>> from bika.lims.interfaces import INumberGenerator
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> year = date_now.split('-')[0][2:]
>>> sample_date = DateTime(2017, 1, 31)
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> bika_sampletypes = setup.bika_sampletypes
>>> bika_samplepoints = setup.bika_samplepoints
>>> bika_analysiscategories = setup.bika_analysiscategories
>>> bika_analysisservices = setup.bika_analysisservices
>>> bika_labcontacts = setup.bika_labcontacts
>>> bika_storagelocations = setup.bika_storagelocations
>>> bika_samplingdeviations = setup.bika_samplingdeviations
>>> bika_sampleconditions = setup.bika_sampleconditions
>>> portal_url = portal.absolute_url()
>>> setup_url = portal_url + "/bika_setup"
>>> browser = self.getBrowser()
>>> current_user = ploneapi.user.get_current()
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.35.2 Analysis Requests (AR)

An *AnalysisRequest* can only be created inside a *Client*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="RIDING BYTES", ClientID="RB")
>>> client
<...client-1>
```

To create a new AR, a *Contact* is needed:

```
>>> contact = api.create(client, "Contact", Firstname="Ramon", Surname="Bartl")
>>> contact
<...contact-1>
```

A *SampleType* defines how long the sample can be retained, the minimum volume needed, if it is hazardous or not, the point where the sample was taken etc.:

```
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="water")
>>> sampletype
<...sampletype-1>
```

A *SamplePoint* defines the location, where a *Sample* was taken:

```
>>> samplepoint = api.create(bika_samplepoints, "SamplePoint", title="Lake of␣
↪Constance")
>>> samplepoint
<...samplepoint-1>
```

An *AnalysisCategory* categorizes different *AnalysisServices*:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<...analysiscategory-1>
```

An *AnalysisService* defines a analysis service offered by the laboratory:

```
>>> analysisservice = api.create(bika_analysisservices, "AnalysisService",
...     title="PH", Category=analysiscategory, Keyword="PH")
>>> analysisservice
<...analysisservice-1>
```

4.35.3 ID generation

IDs can contain *alphanumeric* or *numeric* numbers, depending on the provided ID Server configuration.

Set up *ID Server* configuration:

```
>>> values = [
...             {'form': '{sampleType}-{year}-{alpha:2a3d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 1},
...             {'form': 'BA-{year}-{seq:04d}',
...              'portal_type': 'Batch',
...              'prefix': 'batch',
...              'sequence_type': 'generated',
...              'split_length': 1,
...              'value': ''},
...          ]
```

```
>>> setup.setIDFormatting(values)
```

An *AnalysisRequest* can be created:

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': sample_date,
...           'DateSampled': sample_date,
...           'SampleType': sampletype.UID(),
...          }
```

```
>>> ploneapi.user.grant_roles(user=current_user,roles = ['Sampler', 'LabClerk'])
>>> transaction.commit()
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId() == "water-{}-AA001".format(year)
True
```

Create a second *AnalysisRequest*:

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': sample_date,
...           'DateSampled': sample_date,
...           'SampleType': sampletype.UID(),
...           }
```

```
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId() == "water-{}-AA002".format(year)
True
```

Create a *Batch*:

```
>>> batches = self.portal.batches
>>> batch = api.create(batches, "Batch", ClientID="RB")
>>> batch.getId() == "BA-{}-0001".format(year)
True
```

Change ID formats and create new *AnalysisRequest*:

```
>>> values = [
...             {'form': '{clientId}-{dateSampled:%Y%m%d}-{sampleType}-{seq:04d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 1},
...             {'form': 'BA-{year}-{seq:04d}',
...              'portal_type': 'Batch',
...              'prefix': 'batch',
...              'sequence_type': 'generated',
...              'split_length': 1,
...              'value': ''},
...           ]
```

```
>>> setup.setIDFormatting(values)
```

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': sample_date,
...           'DateSampled': sample_date,
...           'SampleType': sampletype.UID(),
...           }
```

```
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'RB-20170131-water-0001'
```

Re-seed and create a new *Batch*:

```
>>> from zope.component import getUtility
>>> from bika.lims.numbergenerator import INumberGenerator
>>> ng = getUtility(INumberGenerator)
>>> seed = ng.set_number("batch-BA", 10)
```

```
>>> batch = api.create(batches, "Batch", ClientID="RB")
>>> batch.getId() == "BA-{}-0011".format(year)
True
```

Change ID formats and use alphanumeric ids:

```
>>> sampletype2 = api.create(bika_sampletypes, "SampleType", Prefix="WB")
>>> sampletype2
<...sampletype-2>
```

```
>>> values = [
...             {'form': '{sampleType}-{alpha:3a1d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 1},
...           ]
```

```
>>> setup.setIDFormatting(values)
>>> values = {'SampleType': sampletype2.UID(),}
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'WB-AAA1'
```

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'WB-AAA2'
```

Now generate 8 more ARs to force the alpha segment to change:

```
>>> for num in range(8):
...     ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'WB-AAB1'
```

And try now without separators:

```
>>> values = [
...             {'form': '{sampleType}{alpha:3a1d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 1},
...           ]
```

```
>>> setup.setIDFormatting(values)
>>> values = {'SampleType': sampletype2.UID(),}
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
```

The system continues after the previous ID, even if no separator is used:

```
>>> ar.getId()
'WBAAB2'
```

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'WBAAB3'
```

Now generate 8 more ARs to force the alpha segment to change

```
>>> for num in range(8):
...     ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'WBAAC2'
```

TODO: Test the case when numbers are exhausted in a sequence!

4.35.4 IDs with Suffix

In SENAITE < 1.3.0 it was differentiated between an *Analysis Request* and a *Sample*. The *Analysis Request* acted as a "holder" of a *Sample* and the ID used to be the same as the holding *Sample* but with the suffix *-R01*.

This suffix was incremented, e.g. *-R01* to *-R02*, when a retest was requested, while keeping the ID of the previous part constant.

With SENAITE 1.3.0 there is no differentiation anymore between Analysis Request and Sample. However, some labs might still want to follow the old ID scheme with the suffix and incrementation of retests to keep their analysis reports in a sane state.

Therefore, the ID Server also supports Suffixes and the logic to generated the next suffix number for retests:

```
>>> values = [
...             {'form': '{sampleType}-{year}-{seq:04d}-R01',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 2},
...             {'form': '{parent_base_id}-R{test_count:02d}',
...              'portal_type': 'AnalysisRequestRetest',
...              'prefix': 'analysisrequestretest',
...              'sequence_type': '',
...              'split_length': 1},
...           ]
```

```
>>> setup.setIDFormatting(values)
```

Allow self-verification of results:

```
>>> setup.setSelfVerificationEnabled(True)
```

Create a new *AnalysisRequest*:

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': sample_date,
...           'DateSampled': sample_date,
...           'SampleType': sampletype.UID(),
...           }
```

```
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId() == "water-{}-0001-R01".format(year)
True
```

Receive the Sample:

```
>>> do_action_for(ar, "receive")[0]
True
```

Submit and verify results:

```
>>> an = ar.getAnalyses(full_objects=True)[0]
>>> an.setResult(5)
```

```
>>> do_action_for(an, "submit")[0]
True
```

```
>>> do_action_for(an, "verify")[0]
True
```

The AR should benow in the state *verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
```

We can invalidate it now:

```
>>> do_action_for(ar, "invalidate")[0]
True
```

Now a retest was created with the same ID as the invalidated AR, but an incremented suffix:

```
>>> retest = ar.getRetest()
>>> retest.getId() == "water-{}-0001-R02".format(year)
True
```

Submit and verify results of the retest:

```
>>> an = retest.getAnalyses(full_objects=True)[0]
>>> an.setResult(5)
```

```
>>> do_action_for(an, "submit")[0]
True
```

```
>>> do_action_for(an, "verify")[0]
True
```

The Retest should benow in the state *verified*:

```
>>> api.get_workflow_status_of(retest)
'verified'
```

We can invalidate it now:

```
>>> do_action_for(retest, "invalidate")[0]
True
```

Now a retest of the retest was created with the same ID as the invalidated AR, but an incremented suffix:

```
>>> retest = retest.getRetest()
>>> retest.getId() == "water-{}-0001-R03".format(year)
True
```

4.35.5 ID Slicing

The ID slicing machinery that comes with ID Server takes into consideration both wildcards (e.g "{SampleType}") and separators (by default "-"):

```
>>> id_format = "AR-{sampleType}-{parentId}{alpha:3a2d}"
```

If default separator "-" is used, the segments generated are: *["AR", "{sampleType}", "{parentId}", "{alpha:3a2d}"]*

```
>>> idserver.slice(id_format, separator="-", start=0, end=3)
'AR-{sampleType}-{parentId}'
```

```
>>> idserver.slice(id_format, separator="-", start=1, end=2)
'{sampleType}-{parentId}'
```

If no separator is used, note the segments generated are like follows: *["AR-", "{sampleType}", "-", "{parentId}", "{alpha:3a2d}"]*

```
>>> idserver.slice(id_format, separator="", start=0, end=3)
'AR-{sampleType}-'
```

```
>>> idserver.slice(id_format, separator="", start=1, end=2)
'{sampleType}-'
```

And if we use a separator other than "-", we have the same result as before:

```
>>> idserver.slice(id_format, separator=".", start=0, end=3)
'AR-{sampleType}-'
```

```
>>> idserver.slice(id_format, separator=".", start=1, end=2)
'{sampleType}-'
```

Unless we define an ID format in accordance:

```
>>> id_format = "AR.{sampleType}.{parentId}{alpha:3a2d}"
```

So we get the same results as the beginning:

```
>>> idserver.slice(id_format, separator=".", start=0, end=3)
'AR.{sampleType}.{parentId}'
```

```
>>> idserver.slice(id_format, separator=".", start=1, end=2)
'{sampleType}.{parentId}'
```

If we define an ID format without separator, the result will always be the same regardless of setting a separator as a parm or not:

```
>>> id_format = "AR{sampleType}{parentId}{alpha:3a2d}"
>>> idserver.slice(id_format, separator="-", start=0, end=3)
'AR{sampleType}{parentId}'
```

```
>>> idserver.slice(id_format, separator="", start=0, end=3)
'AR{sampleType}{parentId}'
```

```
>>> idserver.slice(id_format, separator="-", start=1, end=2)
'{sampleType}{parentId}'
```

Try now with a simpler and quite common ID:

```
>>> id_format = "WS-{seq:04d}"
>>> idserver.slice(id_format, separator="-", start=0, end=1)
'WS'
```

```
>>> id_format = "WS{seq:04d}"
>>> idserver.slice(id_format, separator="-", start=0, end=1)
'WS'
```

```
>>> idserver.slice(id_format, separator="", start=0, end=1)
'WS'
```

4.35.6 Number generator storage behavior for IDs with/without separators

Number generator machinery keeps track of the last IDs generated to:

1. Make the creation of new IDs faster. The system does not need to find out the last ID number generated for a given portal type by walking through all objects each time an object is created.
2. Allow to manually reseed the numbering through ng interface. Sometimes, the lab wants an ID to start from a specific number, set manually.

These last-generated IDs are stored in annotation storage.

Set up *ID Server* configuration with an hyphen separated format and create an Analysis Request:

```
>>> id_formatting = [
...             {'form': 'NG-{sampleType}-{alpha:2a3d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 2},
...          ]
```

```
>>> setup.setIDFormatting(id_formatting)
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': sample_date,
...           'DateSampled': sample_date,
...           'SampleType': sampletype.UID(),
...          }
>>> service_uids = [analysisservice.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
```

(continues on next page)

(continued from previous page)

```
>>> ar.getId()
'NG-water-AA001'
```

Check the ID was correctly seeded in storage:

```
>>> number_generator = getUtility(INumberGenerator)
>>> last_number = number_generator.get("analysisrequest-NG-water")
>>> alpha.to_decimal('AA001') == last_number
True
```

Create a new Analysis Request with same format and check again:

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'NG-water-AA002'
>>> number_generator = getUtility(INumberGenerator)
>>> last_number = number_generator.get("analysisrequest-NG-water")
>>> alpha.to_decimal('AA002') == last_number
True
```

Do the same, but with an ID formatting without separators:

```
>>> id_formatting = [
...             {'form': 'NG{sampleType}{alpha:2a3d}',
...              'portal_type': 'AnalysisRequest',
...              'prefix': 'analysisrequest',
...              'sequence_type': 'generated',
...              'split_length': 2},
...          ]
```

```
>>> setup.setIDFormatting(id_formatting)
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'NGwaterAA001'
```

Check if the ID was correctly seeded in storage:

```
>>> number_generator = getUtility(INumberGenerator)
>>> last_number = number_generator.get("analysisrequest-NGwater")
>>> alpha.to_decimal('AA001') == last_number
True
```

Create a new Analysis Request with same format and check again:

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar.getId()
'NGwaterAA002'
>>> number_generator = getUtility(INumberGenerator)
>>> last_number = number_generator.get("analysisrequest-NGwater")
>>> alpha.to_decimal('AA002') == last_number
True
```

4.36 Instrument Calibration, Certification and Validation

Instruments represent the physical gadgets of the lab.

Each instrument needs calibration from time to time, which can be done inhouse or externally.

If an instrument is calibrated, an instrument certification is issued. Certifications are only valid within a specified date range.

Instruments can also be validated by the lab personell for a given time.

Only valid instruments, which are not currently calibrated or validated are available in the system and can be used to fetch results for analysis.

Running this test from the buildout directory:

```
bin/test -t InstrumentCalibrationCertificationAndValidation
```

4.37 Test Setup

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
>>> from zope.lifecycleevent import modified
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

```
>>> portal = self.portal
>>> portal_url = portal.absolute_url()
>>> bika_setup = portal.bika_setup
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Manager', 'Owner'])
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

```
>>> def create(container, portal_type, title=None):
...     # Creates a content in a container and manually calls processForm
...     title = title is None and "Test {}".format(portal_type) or title
...     _ = container.invokeFactory(portal_type, id="tmpID", title=title)
...     obj = container.get(_)
...     obj.processForm()
...     modified(obj)  # notify explicitly for the test
...     transaction.commit()  # somehow the created method did not appear until I␣
↪added this
...     return obj
```

```
>>> def get_workflows_for(context):
...     # Returns a tuple of assigned workflows for the given context
...     workflow = ploneapi.portal.get_tool("portal_workflow")
...     return workflow.getChainFor(context)
```

```
>>> def get_workflow_status_of(context):
...     # Returns the workflow status of the given context
...     return ploneapi.content.get_state(context)
```

4.38 Instruments

All instruments live in the */bika_setup/bika_instruments* folder:

```
>>> instruments = bika_setup.bika_instruments
>>> instrument1 = create(instruments, "Instrument", title="Instrument-1")
>>> instrument2 = create(instruments, "Instrument", title="Instrument-2")
>>> instrument3 = create(instruments, "Instrument", title="Instrument-3")
```

Instruments provide the *IInstrument* interface:

```
>>> from bika.lims.interfaces import IInstrument
>>> IInstrument.providedBy(instrument1)
True
```

4.39 Calibrations

Instrument calibrations live inside an instrument:

```
>>> calibration1 = create(instrument1, "InstrumentCalibration", title="Calibration-1")
>>> calibration2 = create(instrument2, "InstrumentCalibration", title="Calibration-2")
```

Calibrations provide the *IInstrumentCalibration* interface:

```
>>> from bika.lims.interfaces import IInstrumentCalibration
>>> IInstrumentCalibration.providedBy(calibration1)
True
```

Calibrations can be in progress or not, depending on the entered dates:

```
>>> calibration1.isCalibrationInProgress()
False
```

The *DownFrom* field specifies the start date of the calibration:

```
>>> calibration1.setDownFrom(DateTime())
```

The calibration shouldn't be in progress with only this field set:

```
>>> calibration1.isCalibrationInProgress()
False
```

The *DownTo* field specifies the end date of the calibration:

```
>>> calibration1.setDownTo(DateTime() + 7) # In calibration for 7 days
```

With this valid date range, the calibration is in progress:

```
>>> calibration1.isCalibrationInProgress()
True
```

The instrument will return in 7 days:

```
>>> calibration1.getRemainingDaysInCalibration()
7
```

Only valid date ranges switch the calibration to "in progress":

```
>>> calibration2.setDownFrom(DateTime() + 7)
>>> calibration2.setDownTo(DateTime())

>>> calibration2.isCalibrationInProgress()
False

>>> calibration2.getRemainingDaysInCalibration()
0
```

The instrument knows if a calibration is in progress:

```
>>> instrument1.isCalibrationInProgress()
True

>>> instrument2.isCalibrationInProgress()
False
```

Since multiple calibrations might be in place, the instrument needs to know about the calibration which takes the longest time:

```
>>> calibration3 = create(instrument1, "InstrumentCalibration", title="Calibration-3")
>>> calibration3.setDownFrom(DateTime())
>>> calibration3.setDownTo(DateTime() + 365)

>>> instrument1.getLatestValidCalibration()
<InstrumentCalibration at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentcalibration-3>
```

Only calibrations which are currently in progress are returned. So if it would start tomorrow, it should not be returned:

```
>>> calibration3.setDownFrom(DateTime() + 1)
>>> calibration3.isCalibrationInProgress()
False
>>> instrument1.getLatestValidCalibration()
<InstrumentCalibration at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentcalibration-1>
```

If all calibrations are dated in the future, it should return none:

```
>>> calibration1.setDownFrom(DateTime() + 1)
>>> calibration1.isCalibrationInProgress()
False
>>> instrument1.getLatestValidCalibration()
```

Instruments w/o any calibration should return no valid calibrations:

```
>>> instrument3.getLatestValidCalibration()
```

4.40 Calibration Certificates

Certification live inside an instrument:

```
>>> certification1 = create(instrument1, "InstrumentCertification", title=
↪"Certification-1")
>>> certification2 = create(instrument2, "InstrumentCertification", title=
↪"Certification-2")
```

Certifications provide the *IInstrumentCertification* interface:

```
>>> from bika.lims.interfaces import IInstrumentCertification
>>> IInstrumentCertification.providedBy(certification1)
True
```

Certifications can be in valid or not, depending on the entered dates:

```
>>> certification1.isValid()
False
```

The *ValidFrom* field specifies the start date of the certification:

```
>>> certification1.setValidFrom(DateTime())
```

The certification shouldn't be valid with only this field set:

```
>>> certification1.isValid()
False
```

The *ValidTo* field specifies the expiration date of the certification:

```
>>> certification1.setValidTo(DateTime() + 7)  # one week until expiration
```

With this valid date range, the certification is in valid:

```
>>> certification1.isValid()
True
```

For exactly 7 days:

```
>>> certification1.getDaysToExpire()
7
```

Or one week:

```
>>> certification1.getWeeksAndDaysToExpire()
(1, 0)
```

Only valid date ranges switch the certification to "valid":

```
>>> certification2.setValidFrom(DateTime() + 7)
>>> certification2.setValidTo(DateTime())

>>> certification2.isValid()
False

>>> certification2.getDaysToExpire()
0

>>> certification2.getWeeksAndDaysToExpire()
(0, 0)
```

The instrument knows if a certification is valid/out of date:

```
>>> instrument1.isOutOfDate()
False

>>> instrument2.isOutOfDate()
True
```

Since multiple certifications might be in place, the instrument needs to know about the certification with the longest validity:

```
>>> certification3 = create(instrument1, "InstrumentCertification", title=
↪"Certification-3")
>>> certification3.setValidFrom(DateTime())
>>> certification3.setValidTo(DateTime() + 365)  # one year until expiration

>>> instrument1.getLatestValidCertification()
<InstrumentCertification at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentcertification-3>
```

Only certifications which are valid are returned. So if the validation would start tomorrow, it should not be returned:

```
>>> certification3.setValidFrom(DateTime() + 1)
>>> certification3.isValid()
False
>>> instrument1.getLatestValidCertification()
<InstrumentCertification at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentcertification-1>
```

If all certifications are dated in the future, it shouldn't be returned:

```
>>> certification1.setValidFrom(DateTime() + 1)
>>> certification1.setValidTo(DateTime() + 7)
>>> instrument1.getLatestValidCertification()
```

It should also marked as invalid:

```
>>> certification1.isValid()
False
```

But the days to expire are calculated until the *ValidTo* date from today. Thus, the full 7 days are returned:

```
>>> certification1.getDaysToExpire()
7
```

Instruments w/o any certifications should also return no valid certifications:

```
>>> instrument3.getLatestValidCertification()
```

4.41 Certification Expiration Intervals

Besides the *ValidFrom* and *ValidTo* date range, users might also specify an *ExpirationInterval*, which calculates the expiration date automatically on save.

Removing the *ValidTo* field makes the certificate invalid:

```
>>> certification1.setValidFrom(DateTime())
>>> certification1.setValidTo(None)

>>> certification1.isValid()
False
```

Setting an interval of 1 year (365 days):

```
>>> certification1.setExpirationInterval(365)
```

The interval takes now precedence over the *ValidTo* date, but only if the custom *setValidTo* setter is called. This setter is always called when using the *edit* form in Plone:

```
>>> certification1.setValidTo(None)
>>> certification1.isValid()
True

>>> certification1.getDaysToExpire()
365
```

4.42 Validation

Validations live inside an instrument:

```
>>> validation1 = create(instrument1, "InstrumentValidation", title="Validation-1")
>>> validation2 = create(instrument2, "InstrumentValidation", title="Validation-2")
```

Validations provide the *IInstrumentValidation* interface:

```
>>> from bika.lims.interfaces import IInstrumentValidation
>>> IInstrumentValidation.providedBy(validation1)
True
```

Validations can be in progress or not, depending on the entered dates:

```
>>> validation1.isValidationInProgress()
False
```

The *DownFrom* field specifies the start date of the validation:

```
>>> validation1.setDownFrom(DateTime())
```

The validation shouldn't be in progress with only this field set:

```
>>> validation1.isValidationInProgress()
False
```

The *DownTo* field specifies the end date of the validation:

```
>>> validation1.setDownTo(DateTime() + 7)  # Down for 7 days
```

With this valid date range, the calibration is in progress:

```
>>> validation1.isValidationInProgress()
True
```

The instrument will be available after 7 days:

```
>>> validation1.getRemainingDaysInValidation()
7
```

Since multiple validations might be in place, the instrument needs to know about the validation which takes the longest time:

```
>>> validation3 = create(instrument1, "InstrumentValidation", title="Validation-3")
>>> validation3.setDownFrom(DateTime())
>>> validation3.setDownTo(DateTime() + 365)

>>> instrument1.getLatestValidValidation()
<InstrumentValidation at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentvalidation-3>
```

Only validations which are currently in progress are returned. So if it would start tomorrow, it should not be returned:

```
>>> validation3.setDownFrom(DateTime() + 1)
>>> validation3.isValidationInProgress()
False
>>> instrument1.getLatestValidValidation()
<InstrumentValidation at /plone/bika_setup/bika_instruments/instrument-1/
↪instrumentvalidation-1>
```

If all validations are dated in the future, it should return none:

```
>>> validation1.setDownFrom(DateTime() + 1)
>>> validation1.isValidationInProgress()
False
>>> instrument1.getLatestValidValidation()
```

Instruments w/o any validation should return no valid validations:

```
>>> instrument3.getLatestValidValidation()
```

4.43 Instruments import interface

We are going to test all instruments import interfaces on this one doctest 1. These files can only be added on *tests/files/instruments/* 2. The filenames(files to be imported) have to have the same name as their

> import data interface i.e *exportimport/instruments/generic/two_dimension.py* would match with *tests/files/instruments/generic.two_dimension.csv* and *exportimport/instruments/varian/vistapro/icp.py*

would match with *tests/files/instruments/varian.vistapro.icp.csv* The reason for the above filenaming is so that we can do *interface = varian.vistapro.icp exec('from senaite.core.exportimport.instruments.{} import Import'.format(inteface))* LINE:225

3. All the files would have the same SampleID/AR-ID *H2O-0001*
4. Same analyses and same results because they will be testing against the same AR *Ca* = 0.0 *Mg* = 2.0
5. To set DefaultResult to float *0.0* use *get_result* example can be found at *exportimport/instruments/varian/vistapro/icp.py*

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t InstrumentsImportInterface
```

4.43.1 Test Setup

Needed imports:

```
>>> import os
>>> import transaction
>>> import cStringIO
>>> from Products.CMFCore.utils import getToolByName
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from DateTime import DateTime

>>> import codecs
>>> from senaite.core.exportimport import instruments
>>> from senaite.core.exportimport.instruments.abbott.m2000rt.m2000rt \
...     import Abbottm2000rtTSVParser, Abbottm2000rtImporter
>>> from bika.lims.browser.resultsimport.resultsimport import ConvertToUploadFile
>>> from zope.publisher.browser import FileUpload, TestRequest
```

Functional helpers:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)

>>> class TestFile(object):
...     def __init__(self, file, filename='dummy.txt'):
...         self.file = file
...         self.headers = {}
...         self.filename = filename
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_instruments = bika_setup.bika_instruments
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_calculations = bika_setup.bika_calculations
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.43.2 Import test

Required steps: Create and receive Analysis Request for import test

An *AnalysisRequest* can only be created inside a *Client*, and it also requires a *Contact* and a *SampleType*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="NARALABS", ClientID="NLABS")
>>> client
<Client at /plone/clients/client-1>
>>> contact = api.create(client, "Contact", Firstname="Juan", Surname="Gallostra")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="H2O",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

Create an *AnalysisCategory* (which categorizes different *AnalysisServices*), and add to it an *AnalysisService*. This service matches the service specified in the file from which the import will be performed:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
>>> analysisservice1 = api.create(bika_analysisservices,
...                               "AnalysisService",
...                               title="HIV06ml",
...                               ShortTitle="hiv06",
...                               Category=analysiscategory,
...                               Keyword="HIV06ml")
>>> analysisservice1
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>

>>> analysisservice2 = api.create(bika_analysisservices,
...                          'AnalysisService',
...                          title='Magnesium',
...                          ShortTitle='Mg',
...                          Category=analysiscategory,
...                          Keyword="Mg")
>>> analysisservice2
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>
>>> analysisservice3 = api.create(bika_analysisservices,
...                           'AnalysisService',
...                           title='Calcium',
...                           ShortTitle='Ca',
...                           Category=analysiscategory,
...                           Keyword="Ca")
>>> analysisservice3
```

(continues on next page)

(continued from previous page)

```
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-3>

>>> total_calc = api.create(bika_calculations, 'Calculation', title='TotalMagCal')
>>> total_calc.setFormula('[Mg] + [Ca]')
>>> analysisservice4 = api.create(bika_analysisservices, 'AnalysisService', title=
↪'THCaCO3', Keyword="THCaCO3")
>>> analysisservice4.setUseDefaultCalculation(False)
>>> analysisservice4.setCalculation(total_calc)
>>> analysisservice4
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-4>

>>> interim_calc = api.create(bika_calculations, 'Calculation', title='Test-Total-Pest
↪')
>>> pest1 = {'keyword': 'pest1', 'title': 'Pesticide 1', 'value': 0, 'type': 'int',
↪'hidden': False, 'unit': ''}
>>> pest2 = {'keyword': 'pest2', 'title': 'Pesticide 2', 'value': 0, 'type': 'int',
↪'hidden': False, 'unit': ''}
>>> pest3 = {'keyword': 'pest3', 'title': 'Pesticide 3', 'value': 0, 'type': 'int',
↪'hidden': False, 'unit': ''}
>>> interims = [pest1, pest2, pest3]
>>> interim_calc.setInterimFields(interims)
>>> self.assertEqual(interim_calc.getInterimFields(), interims)
>>> interim_calc.setFormula('((([pest1] > 0.0) or ([pest2] > .05) or ([pest3] > 10.0)␣
↪) and "PASS" or "FAIL" )')
>>> analysisservice5 = api.create(bika_analysisservices, 'AnalysisService', title=
↪'Total Terpenes', Keyword="TotalTerpenes")
>>> analysisservice5.setUseDefaultCalculation(False)
>>> analysisservice5.setCalculation(interim_calc)
>>> analysisservice5.setInterimFields(interims)
>>> analysisservice5
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-5>
```

Create an *AnalysisRequest* with this *AnalysisService* and receive it:

```
>>> values = {
...           'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SamplingDate': date_now,
...           'DateSampled': date_now,
...           'SampleType': sampletype.UID()
...          }
>>> service_uids = [analysisservice1.UID(),
...                 analysisservice2.UID(),
...                 analysisservice3.UID(),
...                 analysisservice4.UID(),
...                 analysisservice5.UID()
...                ]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/H2O-0001>
>>> ar.getReceivedBy()
''
>>> wf = getToolByName(ar, 'portal_workflow')
>>> wf.doActionFor(ar, 'receive')
>>> ar.getReceivedBy()
'test_user_1_'
```

4.43.3 Instruments files path

Where testing files live:

```
>>> files_path = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..',
↪'files/instruments'))
>>> instruments_path = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '../.
↪.', 'exportimport/instruments'))
>>> files = os.listdir(files_path)
>>> interfaces = []
>>> importer_filename = [] #List of tuples [(importer,filename),(importer, filename)]
>>> for fl in files:
...     inst_interface = os.path.splitext(fl)[0]
...     inst_path = '.'.join([inst_interface.replace('.', '/'), 'py'])
...     if os.path.isfile(os.path.join(instruments_path, inst_path)):
...         interfaces.append(inst_interface)
...         importer_filename.append((inst_interface, fl))
...     else:
...         inst_path = '.'.join([fl.replace('.', '/'), 'py'])
...         if os.path.isfile(os.path.join(instruments_path, inst_path)):
...             interfaces.append(fl)
...             importer_filename.append((fl, fl))
...         else:
...             self.fail('File {} found does match any import interface'.format(fl))
```

4.43.4 Availability of instrument interface

Check that the instrument interface is available:

```
>>> exims = []
>>> for exim_id in instruments.__all__:
...     exims.append(exim_id)
>>> [f for f in interfaces if f not in exims]
[]
```

4.43.5 Assigning the Import Interface to an Instrument

Create an *Instrument* and assign to it the tested Import Interface:

```
>>> for inter in interfaces:
...     title = inter.split('.')[0].title()
...     instrument = api.create(bika_instruments, "Instrument", title=title)
...     instrument.setImportDataInterface([inter])
...     if instrument.getImportDataInterface() != [inter]:
...         self.fail('Instrument Import Data Interface did not get set')

>>> for inter in importer_filename:
...     exec('from senaite.core.exportimport.instruments.{} import Import'.
↪format(inter[0]))
...     filename = os.path.join(files_path, inter[1])
...     data = open(filename, 'r').read()
...     import_file = FileUpload(TestFile(cStringIO.StringIO(data), inter[1]))
...     request = TestRequest(form=dict(
...                                 submitted=True,
```

(continues on next page)

(continued from previous page)

```
...                                 artoapply='received_tobeverified',
...                                 results_override='override',
...                                 instrument_results_file=import_file,
...                                 sample='requestid',
...                                 instrument=''))
...     context = self.portal
...     results = Import(context, request)
...     test_results = eval(results)
...     #TODO: Test for interim fields on other files aswell
...     analyses = ar.getAnalyses(full_objects=True)
...     if 'Parsing file generic.two_dimension.csv' in test_results['log']:
...         # Testing also for interim fields, only for `generic.two_dimension`␣
↪interface
...         # TODO: Test for - H2O-0001: calculated result for 'THCaCO3': '2.0'
...         if 'Import finished successfully: 1 Samples and 3 results updated' not in␣
↪test_results['log']:
...             self.fail("Results Update failed")
...         if "H2O-0001 result for 'TotalTerpenes:pest1': '1'" not in test_results[
↪'log']:
...             self.fail("pest1 did not get updated")
...         if "H2O-0001 result for 'TotalTerpenes:pest2': '1'" not in test_results[
↪'log']:
...             self.fail("pest2 did not get updated")
...         if "H2O-0001 result for 'TotalTerpenes:pest3': '1'" not in test_results[
↪'log']:
...             self.fail("pest3 did not get updated")
...         for an in analyses:
...             if an.getKeyword() == 'TotalTerpenes':
...                 if an.getResult() != 'PASS':
...                     msg = "{}:Result did not get updated".format(an.getKeyword())
...                     self.fail(msg)
...
...     elif 'Import finished successfully: 1 Samples and 2 results updated' not in␣
↪test_results['log']:
...         self.fail("Results Update failed")
...
...     for an in analyses:
...         if an.getKeyword() ==  'Ca':
...             if an.getResult() != '0.0':
...                 msg = "{}:Result did not get updated".format(an.getKeyword())
...                 self.fail(msg)
...         if an.getKeyword() ==  'Mg':
...             if an.getResult() != '2.0':
...                 msg = "{}:Result did not get updated".format(an.getKeyword())
...                 self.fail(msg)
...         if an.getKeyword() ==  'THCaCO3':
...             if an.getResult() != '2.0':
...                 msg = "{}:Result did not get updated".format(an.getKeyword())
...                 self.fail(msg)
...
...     if 'Import' in globals():
...         del Import
```

4.44 Internal Use of Samples and Analyses

Running this test from the buildout directory:

bin/test test_textual_doctests -t InternalUse

4.44.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.interfaces import IInternalUse
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils.analysisrequest import create_partition
>>> from bika.lims.subscribers.analysisrequest import gather_roles_for_permission
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from Products.CMFCore import permissions
>>> from zope.lifecycleevent import modified
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_sample(services, internal_use=False):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID(),
...         'InternalUse': internal_use,}
...     service_uids = map(api.get_uid, services)
...     return create_analysisrequest(client, request, values, service_uids)
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.44.2 Set a Sample for internal use

Create a Sample for non internal use:

```
>>> sample = new_sample([Cu, Fe, Au])
>>> transitioned = do_action_for(sample, "receive")
>>> sample.getInternalUse()
False
>>> IInternalUse.providedBy(sample)
False
>>> internals = map(IInternalUse.providedBy, sample.getAnalyses(full_objects=True))
>>> any(internals)
False
```

Client contact does have access to this Sample:

```
>>> "Owner" in gather_roles_for_permission(permissions.View, sample)
True
>>> "Owner" in gather_roles_for_permission(permissions.ListFolderContents, sample)
True
>>> "Owner" in gather_roles_for_permission(permissions.AccessContentsInformation,␣
↪sample)
True
```

Set the sample for internal use:

```
>>> sample.setInternalUse(True)
>>> modified(sample)
>>> sample.getInternalUse()
```

(continues on next page)

(continued from previous page)

```
True
>>> IInternalUse.providedBy(sample)
True
>>> internals = map(IInternalUse.providedBy, sample.getAnalyses(full_objects=True))
>>> all(internals)
True
```

Client contact does not have access to this Sample anymore:

```
>>> "Owner" in gather_roles_for_permission(permissions.View, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.ListFolderContents, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.AccessContentsInformation,␣
↪sample)
False
```

Even if we submit results and sample is transitioned thereafter:

```
>>> for analysis in sample.getAnalyses(full_objects=True):
...     analysis.setResult(12)
...     success = do_action_for(analysis, "submit")
>>> api.get_workflow_status_of(sample)
'to_be_verified'
```

```
>>> sample.getInternalUse()
True
>>> IInternalUse.providedBy(sample)
True
>>> internals = map(IInternalUse.providedBy, sample.getAnalyses(full_objects=True))
>>> all(internals)
True
>>> "Owner" in gather_roles_for_permission(permissions.View, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.ListFolderContents, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.AccessContentsInformation,␣
↪sample)
False
```

4.44.3 Creation of a Sample for internal use

Create a Sample for internal use:

```
>>> sample = new_sample([Cu, Fe, Au], internal_use=True)
>>> transitioned = do_action_for(sample, "receive")
>>> modified(sample)
>>> sample.getInternalUse()
True
>>> IInternalUse.providedBy(sample)
True
>>> internals = map(IInternalUse.providedBy, sample.getAnalyses(full_objects=True))
>>> all(internals)
True
```

Client contact does not have access to this Sample:

```
>>> "Owner" in gather_roles_for_permission(permissions.View, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.ListFolderContents, sample)
False
>>> "Owner" in gather_roles_for_permission(permissions.AccessContentsInformation,␣
↪sample)
False
```

4.44.4 Creation of a Partition for internal use

Create a Sample for non internal use:

```
>>> sample = new_sample([Cu, Fe, Au])
>>> transitioned = do_action_for(sample, "receive")
```

Create two partitions, the first for internal use:

```
>>> analyses = sample.getAnalyses(full_objects=True)
>>> part1 = create_partition(sample, request, analyses[2:], internal_use=True)
>>> part2 = create_partition(sample, request, analyses[:2], internal_use=False)
>>> IInternalUse.providedBy(part1)
True
>>> IInternalUse.providedBy(part2)
False
>>> IInternalUse.providedBy(sample)
False
```

Submit results for partition 2 (non-internal-use):

```
>>> for analysis in part2.getAnalyses(full_objects=True):
...     analysis.setResult(12)
...     success = do_action_for(analysis, "submit")
>>> api.get_workflow_status_of(part2)
'to_be_verified'
```

Since partition 1 is labelled for internal use, the primary sample has been automatically transitioned too:

```
>>> api.get_workflow_status_of(sample)
'to_be_verified'
```

While partition 1 remains in “received” status:

```
>>> api.get_workflow_status_of(part1)
'sample_received'
```

4.45 Listings

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t Listings
```

4.45.1 Test Setup

Imports:

```
>>> from operator import methodcaller
>>> from DateTime import DateTime
```

```
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
```

Functional Helpers:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def create_ar(client, **kw):
...     values = {}
...     services = []
...     for k, v in kw.iteritems():
...         if k == "Services":
...             services = map(api.get_uid, v)
...         elif api.is_object(v):
...             values[k] = api.get_uid(v)
...         else:
...             values[k] = v
...     return create_analysisrequest(client, self.request, values, services)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
```

Test User:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.45.2 Prepare Test Environment

Setupitems:

```
>>> clients = portal.clients
>>> sampletypes = setup.bika_sampletypes
>>> samplepoints = setup.bika_samplepoints
>>> analysiscategories = setup.bika_analysiscategories
>>> analysisservices = setup.bika_analysisservices
```

Create Clients:

```
>>> cl1 = api.create(clients, "Client", Name="Client1", ClientID="C1")
>>> cl2 = api.create(clients, "Client", Name="Client2", ClientID="C2")
>>> cl3 = api.create(clients, "Client", Name="Client3", ClientID="C3")
```

Create some Contact(s):

```
>>> c1 = api.create(cl1, "Contact", Firstname="Client", Surname="1")
>>> c2 = api.create(cl2, "Contact", Firstname="Client", Surname="2")
>>> c3 = api.create(cl3, "Contact", Firstname="Client", Surname="3")
```

Create some Sample Types:

```
>>> st1 = api.create(sampletypes, "SampleType", Prefix="s1", MinimumVolume="100 ml")
>>> st2 = api.create(sampletypes, "SampleType", Prefix="s2", MinimumVolume="200 ml")
>>> st3 = api.create(sampletypes, "SampleType", Prefix="s3", MinimumVolume="300 ml")
```

Create some Sample Points:

```
>>> sp1 = api.create(samplepoints, "SamplePoint", title="Sample Point 1")
>>> sp2 = api.create(samplepoints, "SamplePoint", title="Sample Point 2")
>>> sp3 = api.create(samplepoints, "SamplePoint", title="Sample Point 3")
```

Create some Analysis Categories:

```
>>> ac1 = api.create(analysiscategories, "AnalysisCategory", title="Analysis Category␣
↪1")
>>> ac2 = api.create(analysiscategories, "AnalysisCategory", title="Analysis Category␣
↪2")
>>> ac3 = api.create(analysiscategories, "AnalysisCategory", title="Analysis Category␣
↪3")
```

Create some Analysis Services:

```
>>> as1 = api.create(analysisservices, "AnalysisService", title="Analysis Service 1",␣
↪ShortTitle="AS1", Category=ac1, Keyword="AS1", Price="10")
>>> as2 = api.create(analysisservices, "AnalysisService", title="Analysis Service 2",␣
↪ShortTitle="AS1", Category=ac2, Keyword="AS1", Price="20")
>>> as3 = api.create(analysisservices, "AnalysisService", title="Analysis Service 3",␣
↪ShortTitle="AS1", Category=ac3, Keyword="AS1", Price="30")
```

Create some Analysis Requests:

```
>>> ar11 = create_ar(cl1, Contact=c1, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st1, Priority='1', Services=[as1])
>>> ar12 = create_ar(cl1, Contact=c1, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st1, Priority='2', Services=[as1])
>>> ar13 = create_ar(cl1, Contact=c1, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st1, Priority='3', Services=[as1])
```

```
>>> ar21 = create_ar(cl2, Contact=c2, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st2, Priority='1', Services=[as2])
>>> ar22 = create_ar(cl2, Contact=c2, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st2, Priority='2', Services=[as2])
>>> ar23 = create_ar(cl2, Contact=c2, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st2, Priority='3', Services=[as2])
```

```
>>> ar31 = create_ar(cl3, Contact=c3, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st3, Priority='1', Services=[as3])
>>> ar32 = create_ar(cl3, Contact=c3, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st3, Priority='2', Services=[as3])
>>> ar33 = create_ar(cl3, Contact=c3, SamplingDate=date_now, DateSampled=date_now,␣
↪SampleType=st3, Priority='3', Services=[as3])
```

4.45.3 Listing View

```
>>> from bika.lims.browser.bika_listing import BikaListingView
>>> context = portal.analysisrequests
>>> request = self.request
>>> listing = BikaListingView(context, request)
>>> listing
<bika.lims.browser.bika_listing.BikaListingView object at 0x...>
```

Setup the view to behave like the *AnalysisRequestsView*:

```
>>> from bika.lims.catalog import CATALOG_ANALYSIS_REQUEST_LISTING
```

```
>>> listing.catalog = CATALOG_ANALYSIS_REQUEST_LISTING
>>> listing.contentFilter = {
...     'sort_on': 'created',
...     'sort_order': 'reverse',
...     'path': {"query": "/", "level": 0},
...     'is_active': True,}
```

The listing view should now return all created ARs:

```
>>> results = listing.search()
>>> len(results)
9
```

Searching for a value should work:

```
>>> results = listing.search(searchterm="s1")
>>> len(results)
3
```

```
>>> map(lambda x: x.getObject().getSampleType().getPrefix(), results)
['s1', 's1', 's1']
```

```
>>> results = listing.search(searchterm="client-3")
>>> map(lambda x: x.getObject().getClient(), results)
[<Client at /plone/clients/client-3>, <Client at /plone/clients/client-3>, <Client at␣
↪/plone/clients/client-3>]
```

4.46 Permissions

All objects in Bika LIMS are permission aware. Therefore, only users with the right **roles** can view or edit contents. Each role may contain one or more **permissions**.

4.47 Test Setup

```
>>> import os
>>> import transaction
>>> from plone import api as ploneapi
>>> from zope.lifecycleevent import modified
```

(continues on next page)

(continued from previous page)

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

```
>>> portal = self.portal
>>> portal_url = portal.absolute_url()
>>> bika_setup = portal.bika_setup
>>> bika_setup_url = portal_url + "/bika_setup"
>>> browser = self.getBrowser()
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def login(user=TEST_USER_ID, password=TEST_USER_PASSWORD):
...     browser.open(portal_url + "/login_form")
...     browser.getControl(name='__ac_name').value = user
...     browser.getControl(name='__ac_password').value = password
...     browser.getControl(name='buttons.login').click()
...     assert("__ac_password" not in browser.contents)
```

```
>>> def logout():
...     browser.open(portal_url + "/logout")
...     assert("You are now logged out" in browser.contents)
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

```
>>> def create(container, portal_type, title=None):
...     # Creates a content in a container and manually calls processForm
...     title = title is None and "Test {}".format(portal_type) or title
...     _ = container.invokeFactory(portal_type, id="tmpID", title=title)
...     obj = container.get(_)
...     obj.processForm()
...     modified(obj)  # notify explicitly for the test
...     transaction.commit()  # somehow the created method did not appear until I␣
↪added this
...     return obj
```

```
>>> def get_workflows_for(context):
...     # Returns a tuple of assigned workflows for the given context
...     workflow = ploneapi.portal.get_tool("portal_workflow")
...     return workflow.getChainFor(context)
```

```
>>> def get_workflow_status_of(context):
...     # Returns the workflow status of the given context
...     return ploneapi.content.get_state(context)
```

4.48 Test Workflows and Permissions

Workflows control the allowed roles for specific permissions. A role is a container for several permissions.

4.48.1 Bika Setup

Bika Setup is a folderish object, which handles the labs' configuration items, like Laboratory information, Instruments, Analysis Services etc.

Test Workflow

A *bika_setup* lives in the root of a bika installation, or more precisely, the portal object:

```
>>> bika_setup = portal.bika_setup
```

The *setup* folder follows the *senaite_setup_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(bika_setup)
('senaite_setup_workflow',)

>>> get_workflow_status_of(bika_setup)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", bika_setup)
['Authenticated']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", bika_setup)
['Authenticated']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", bika_setup)
['Authenticated']
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", bika_setup)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles (nobody) should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", bika_setup)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

Anonymous should not be able to view the *bika_setup* folder:

```
>>> browser.open(bika_setup.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *bika_setup* folder:

```
>>> browser.open(bika_setup.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.2 Laboratory

The Laboratory object holds all needed information about the lab itself. It lives inside the *bika_setup* folder.

Test Workflow

A *laboratory* lives in the root of a bika installation, or more precisely, the portal object:

```
>>> laboratory = portal.bika_setup.laboratory
```

The *laboratory* folder follows the *senaite_laboratory_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(laboratory)
('senaite_laboratory_workflow',)

>>> get_workflow_status_of(laboratory)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", laboratory)
['Authenticated']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", laboratory)
['Authenticated']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", laboratory)
['Authenticated']
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", laboratory)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles (nobody) should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", laboratory)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous should not be able to view the *laboratory* folder:

```
browser.open(laboratory.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *laboratory* folder:

```
>>> browser.open(laboratory.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.3 Lab Contact(s)

Lab Contacts are the employees of the lab.

Test Workflow

A *labcontact* lives in the *bika_setup/bika_labcontacts* folder:

```
>>> labcontacts = bika_setup.bika_labcontacts
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> labcontact = create(labcontacts, "LabContact")
```

The *bika_labcontacts* folder follows the *senaite_one_state_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(labcontacts)
('senaite_one_state_workflow',)

>>> get_workflow_status_of(labcontacts)
'active'
```

A *labcontact* follows the *senaite_deactivable_type_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(labcontact)
('senaite_labcontact_workflow',)

>>> get_workflow_status_of(labcontacts)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", labcontacts)
['Authenticated']

>>> get_roles_for_permission("View", labcontact)
['LabClerk', 'LabManager', 'Manager', 'Publisher']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", labcontacts)
['Authenticated']

>>> get_roles_for_permission("Access contents information", labcontact)
['Authenticated']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", labcontacts)
['Authenticated']

>>> get_roles_for_permission("List folder contents", labcontact)
[]
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", labcontacts)
['LabClerk', 'LabManager', 'Manager']

>>> get_roles_for_permission("Modify portal content", labcontact)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles have should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", labcontacts)
[]

>>> get_roles_for_permission("Delete objects", labcontact)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous should not be able to view the *bika_labcontacts* folder:

```
browser.open(labcontacts.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous should not be able to view a *labcontact*:

```
browser.open(labcontact.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *bika_labcontacts* folder:

```
>>> browser.open(labcontacts.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit a *labcontact*:

```
>>> browser.open(labcontact.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.4 Clients and Contacts

Clients are the customers of the lab. A client represents another company, which has one or more natural persons as contacts.

Test Workflow

A *client* lives in the */clients* folder:

```
>>> clients = portal.clients
>>> client = create(clients, "Client")
>>> another_client = create(clients, "Client")
```

A *contact* lives in a *client*:

```
>>> contact = create(client, "Contact")
```

The *clients* folder follows *senaite_clients_workflow* workflow:

```
>>> get_workflows_for(clients)
('senaite_clients_workflow',)
```

A *client* follows the *senaite_client_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(client)
('senaite_client_workflow',)

>>> get_workflow_status_of(client)
'active'
```

A *contact* follows the *senaite_deactivable_type_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(contact)
('senaite_clientcontact_workflow',)

>>> get_workflow_status_of(contact)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission for clients folder:

```
>>> get_roles_for_permission("View", clients)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles should have a *View* permission for client object. Note that permissions for Client role are not granted, but for Owner. Lab Contacts are Owners of the Client they belong to, so client contacts only have access to the Client they belong to:

```
>>> get_roles_for_permission("View", client)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles should have a *View* permission for client contact object:

```
>>> get_roles_for_permission("View", contact)
['LabClerk', 'LabManager', 'Manager', 'Owner', 'Publisher']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", clients)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("Access contents information", client)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("Access contents information", contact)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", clients)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

(continues on next page)

(continued from previous page)

```
>>> get_roles_for_permission("List folder contents", client)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("List folder contents", contact)
[]
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", clients)
['LabClerk', 'LabManager', 'Manager', 'Owner']

>>> get_roles_for_permission("Modify portal content", client)
['LabClerk', 'LabManager', 'Manager', 'Owner']
```

Exactly these roles have should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", clients)
[]

>>> get_roles_for_permission("Delete objects", client)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

Anonymous should be able to view the *clients* folder:

```
>>> browser.open(clients.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to view a *client*:

```
>>> browser.open(client.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *bika_clients* folder:

```
>>> browser.open(clients.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit a *client*:

```
>>> browser.open(client.absolute_url() + "/base_edit")
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
Unauthorized: ...
```

Client Contacts Browser Test

Create a new user for the contact:

```
>>> user = ploneapi.user.create(email="contact-1@client-1.com", username="contact-1",␣
↪password=TEST_USER_PASSWORD, properties=dict(fullname="Test Contact 1"))
>>> transaction.commit()
```

Now we log in as the new user:

```
>>> login(user.id)
```

The user can not access the clients folder yet:

```
>>> browser.open(clients.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...

>>> browser.open(client.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

Link the user to a client contact to grant access to this client:

```
>>> contact.setUser(user)
True
>>> transaction.commit()
```

Linking a user adds this user to the *Clients* group:

```
>>> clients_group = ploneapi.group.get("Clients")
>>> user.getId() in clients_group.getAllGroupMemberIds()
True
```

This gives the user the global *Client* role:

```
>>> sorted(ploneapi.user.get_roles(user=user))
['Authenticated', 'Client', 'Member']
```

It also grants local *Owner* role on the client object:

```
>>> sorted(user.getRolesInContext(client))
['Authenticated', 'Member', 'Owner']
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') The user is able to modify the client properties:

```
browser.open(client.absolute_url() + "/base_edit")
"edit_form" in browser.contents
True
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') As well as the own contact properties:

```
browser.open(contact.absolute_url() + "/base_edit")
"edit_form" in browser.contents
True
```

But the user can not access other clients:

```
>>> browser.open(another_client.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

Or modify other clients:

```
>>> browser.open(another_client.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Unlink the user to revoke all access to the client:

```
>>> contact.unlinkUser()
True
>>> transaction.commit()
```

The user has no local owner role anymore on the client:

```
>>> sorted(user.getRolesInContext(client))
['Authenticated', 'Member']
```

The user can not access the client anymore:

```
>>> browser.open(clients.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...

>>> browser.open(client.absolute_url())
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.5 Instrument(s)

Instruments represent the measuring hardware of the lab.

Test Workflow

A *instrument* lives in the *bika_setup/bika_instruments* folder:

```
>>> instruments = bika_setup.bika_instruments
>>> instrument = create(instruments, "Instrument")
```

The *bika_instruments* folder follows the *senaite_one_state_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(instruments)
('senaite_instruments_workflow',)

>>> get_workflow_status_of(instruments)
'active'
```

A *instrument* follows the *senaite_deactivable_type_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(instrument)
('senaite_deactivable_type_workflow',)

>>> get_workflow_status_of(instruments)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", instruments)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("View", instrument)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", instruments)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("Access contents information", instrument)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", instruments)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']

>>> get_roles_for_permission("List folder contents", instrument)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Preserver', 'Publisher',
↪'RegulatoryInspector', 'Sampler', 'SamplingCoordinator', 'Verifier']
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", instruments)
['LabClerk', 'LabManager', 'Manager']

>>> get_roles_for_permission("Modify portal content", instrument)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles have should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", instruments)
[]

>>> get_roles_for_permission("Delete objects", instrument)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

Anonymous should not be able to view the *bika_instruments* folder:

```
>>> browser.open(instruments.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous should not be able to view a *instrument*:

```
browser.open(instrument.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *bika_instruments* folder:

```
>>> browser.open(instruments.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit a *instrument*:

```
>>> browser.open(instrument.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.6 Method(s)

Methods describe the sampling methods of the lab.

Methods should be viewable by unauthenticated users for information purpose.

Test Workflow

A *method* lives in the *methods* folder:

```
>>> methods = portal.methods
>>> method = create(methods, "Method")
```

The *methods* folder follows the *senaite_setup_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(methods)
('senaite_setup_workflow',)

>>> get_workflow_status_of(methods)
'active'
```

A *method* follows the *senaite_deactivable_type_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(method)
('senaite_deactivable_type_workflow',)

>>> get_workflow_status_of(methods)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", methods)
['Authenticated']

>>> get_roles_for_permission("View", method)
['Authenticated']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", methods)
['Authenticated']

>>> get_roles_for_permission("Access contents information", method)
['Authenticated']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", methods)
['Authenticated']

>>> get_roles_for_permission("List folder contents", method)
['Authenticated']
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", methods)
['LabClerk', 'LabManager', 'Manager']

>>> get_roles_for_permission("Modify portal content", method)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles have should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", methods)
[]

>>> get_roles_for_permission("Delete objects", method)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

Anonymous should not be able to view the *methods* folder:

```
>>> browser.open(methods.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous should not be able to view a *method*:

```
browser.open(method.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *methods* folder:

```
>>> browser.open(methods.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit a *method*:

```
>>> browser.open(method.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.48.7 Analysis Service(s)

Analysis services describe which "products" the lab offers.

Test Workflow

A *analysisservice* lives in the *bika_setup/bika_analysisservices* folder:

```
>>> analysisservices = bika_setup.bika_analysisservices
>>> analysisservice = create(analysisservices, "AnalysisService")
```

The *bika_analysisservices* folder follows the *senaite_one_state_workflow* and is initially in the *active* state:

```
>>> get_workflows_for(analysisservices)
('senaite_one_state_workflow',)

>>> get_workflow_status_of(analysisservices)
'active'
```

A *analysisservice* follows the *senaite_deactivable_type_workflow* and has an initial state of *active*:

```
>>> get_workflows_for(analysisservice)
('senaite_deactivable_type_workflow',)

>>> get_workflow_status_of(analysisservices)
'active'
```

Test Permissions

Exactly these roles have should have a *View* permission:

```
>>> get_roles_for_permission("View", analysisservices)
['Authenticated']

>>> get_roles_for_permission("View", analysisservice)
['Authenticated']
```

Exactly these roles have should have the *Access contents information* permission:

```
>>> get_roles_for_permission("Access contents information", analysisservices)
['Authenticated']

>>> get_roles_for_permission("Access contents information", analysisservice)
['Authenticated']
```

Exactly these roles have should have the *List folder contents* permission:

```
>>> get_roles_for_permission("List folder contents", analysisservices)
['Authenticated']

>>> get_roles_for_permission("List folder contents", analysisservice)
['Authenticated']
```

Exactly these roles have should have the *Modify portal content* permission:

```
>>> get_roles_for_permission("Modify portal content", analysisservices)
['LabClerk', 'LabManager', 'Manager']

>>> get_roles_for_permission("Modify portal content", analysisservice)
['LabClerk', 'LabManager', 'Manager']
```

Exactly these roles have should have the *Delete objects* permission:

```
>>> get_roles_for_permission("Delete objects", analysisservices)
[]

>>> get_roles_for_permission("Delete objects", analysisservice)
[]
```

Anonymous Browser Test

Ensure we are logged out:

```
>>> logout()
```

Anonymous should not be able to view the *bika_analysisservices* folder:

```
>>> browser.open(analysisservices.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

~~ TODO: Fails with LocationError: (<UnauthorizedBinding: context>, 'main_template') Anonymous are **not** allowed to view an *analysisservice*:

```
browser.open(analysisservice.absolute_url() + "/base_view")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit the *bika_analysisservices* folder:

```
>>> browser.open(analysisservices.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

Anonymous should not be able to edit a *analysisservice*:

```
>>> browser.open(analysisservice.absolute_url() + "/base_edit")
Traceback (most recent call last):
...
Unauthorized: ...
```

4.49 QC Analyses With Interim Fields On A Worksheet

Creating analysis that has interims fields so that we can test for Reference Analyses(Blank and Control) that have interim fields.

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t QCAnalysesWithInterimFieldsOnAWorksheet
```

4.49.1 Test Setup

Needed Imports:

```
>>> import re
>>> import transaction
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor
>>> from DateTime import DateTime
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bikasetup = portal.bika_setup
>>> bika_analysisservices = bika_setup.bika_analysisservices
>>> bika_calculations = bika_setup.bika_calculations
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Analyst'])
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
```

```
>>> interim_calc = api.create(bika_calculations, 'Calculation', title='Test-Total-Pest
↪')
>>> pest1 = {'keyword': 'pest1', 'title': 'Pesticide 1', 'value': 12.3, 'type': 'int',
↪ 'hidden': False, 'unit': ''}
>>> pest2 = {'keyword': 'pest2', 'title': 'Pesticide 2', 'value': 14.89, 'type': 'int
↪', 'hidden': False, 'unit': ''}
>>> pest3 = {'keyword': 'pest3', 'title': 'Pesticide 3', 'value': 16.82, 'type': 'int
↪', 'hidden': False, 'unit': ''}
>>> interims = [pest1, pest2, pest3]
>>> interim_calc.setInterimFields(interims)
>>> self.assertEqual(interim_calc.getInterimFields(), interims)
>>> interim_calc.setFormula('((([pest1] > 0.0) or ([pest2] > .05) or ([pest3] > 10.0)␣
↪) and "FAIL" or "PASS" )')
>>> total_terpenes = api.create(bika_analysisservices, 'AnalysisService', title=
↪'Total Terpenes', Keyword="TotalTerpenes")
>>> total_terpenes.setUseDefaultCalculation(False)
>>> total_terpenes.setCalculation(interim_calc)
>>> total_terpenes.setInterimFields(interims)
>>> total_terpenes
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
>>> service_uids = [total_terpenes.UID()]
```

Create a Reference Definition for blank:

```
>>> blankdef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",␣
↪title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': total_terpenes.UID(), 'result': '0', 'min': '0', 'max': '0'}
↪,]
>>> blankdef.setReferenceResults(blank_refs)
```

And for control:

```
>>> controldef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': total_terpenes.UID(), 'result': '10', 'min': '9.99', 'max
↪': '10.01'},]
>>> controldef.setReferenceResults(control_refs)
```

```
>>> blank = api.create(supplier, "ReferenceSample", title="Blank",
...                    ReferenceDefinition=blankdef,
...                    Blank=True, ExpiryDate=date_future,
...                    ReferenceResults=blank_refs)
>>> control = api.create(supplier, "ReferenceSample", title="Control",
...                      ReferenceDefinition=controldef,
...                      Blank=False, ExpiryDate=date_future,
...                      ReferenceResults=control_refs)
```

Create an Analysis Request:

```
>>> sampletype_uid = api.get_uid(sampletype)
>>> values = {
...     'Client': api.get_uid(client),
...     'Contact': api.get_uid(contact),
...     'DateSampled': date_now,
...     'SampleType': sampletype_uid,
...     'Priority': '1',
... }
```

```
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/W-0001>
>>> success = doActionFor(ar, 'receive')
```

Create a new Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet", Analyst='test_user_1_')
>>> worksheet
<Worksheet at /plone/worksheets/WS-001>
```

```
>>> analyses = map(api.get_object, ar.getAnalyses())
>>> analysis = analyses[0]
>>> analysis
<Analysis at /plone/clients/client-1/W-0001/TotalTerpenes>
>>> worksheet.addAnalysis(analysis)
>>> analysis.getWorksheet().UID() == worksheet.UID()
True
```

Add a blank and a control:

```
>>> blanks = worksheet.addReferenceAnalyses(blank, service_uids)
>>> transaction.commit()
>>> blanks.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
>>> controls = worksheet.addReferenceAnalyses(control, service_uids)
>>> transaction.commit()
>>> controls.sort(key=lambda analysis: analysis.getKeyword(), reverse=False)
>>> transaction.commit()
>>> for analysis in worksheet.getAnalyses():
...     if analysis.portal_type == 'ReferenceAnalysis':
...         if analysis.getReferenceType() == 'b' or analysis.getReferenceType() == 'c
↪':
...             # 3 is the number of interim fields on the analysis/calculation
...             if len(analysis.getInterimFields()) != 3:
...                 self.fail("Blank or Control Analyses interim field are not correct
↪")
```

4.50 Removal of Analyses from an Analysis Request

Running this test from the buildout directory:

bin/test test_textual_doctests -t RemoveAnalysesFromAnalysisRequest

4.50.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
```

(continues on next page)

(continued from previous page)

```
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

And set some settings:

```
>>> bikasetup.setSelfVerificationEnabled(True)
```

4.50.2 Remove Analyses from an Analysis Request not yet received

Create a new Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

And remove two analyses (*Cu* and *Fe*):

```
>>> ar.setAnalyses([Au])
>>> map(lambda an: an.getKeyword(), ar.getAnalyses(full_objects=True))
['Au']
```

And the Analysis Request remains in the same state

```
>>> api.get_workflow_status_of(ar)
'sample_due'
```

4.50.3 Remove Analyses from an Analysis Request with submitted and verified results

Create a new Analysis Request and receive:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Submit results for *Fe*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> analysis_fe = filter(lambda an: an.getKeyword() == "Fe", analyses)[0]
>>> analysis_fe.setResult(12)
>>> transitioned = do_action_for(analysis_fe, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_fe)
'to_be_verified'
```

The Analysis Request status is still *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Submit results for *Au*:

```
>>> analysis_au = filter(lambda an: an.getKeyword() == "Au", analyses)[0]
>>> analysis_au.setResult(14)
>>> transitioned = do_action_for(analysis_au, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_au)
'to_be_verified'
```

And verify *Au*:

```
>>> transitioned = do_action_for(analysis_au, "verify")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_au)
'verified'
```

Again, the Analysis Request status is still *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

But if we remove the analysis without result (*Cu*), the Analysis Request transitions to "to_be_verified" because follows *Fe*:

```
>>> ar.setAnalyses([Fe, Au])
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Analyses which are in the state *to_be_verified* can **not** be removed. Therefore, if we try to remove the analysis *Fe* (in *to_be_verified* state), the Analysis Request will stay in *to_be_verified* and the Analysis will still be assigned:

```
>>> ar.setAnalyses([Au])
```

```
>>> analysis_fe in ar.objectValues()
True
```

```
>>> analysis_au in ar.objectValues()
True
```

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The only way to remove the *Fe* analysis is to retract it first:

```
>>> transitioned = do_action_for(analysis_fe, "retract")
>>> api.get_workflow_status_of(analysis_fe)
'retracted'
```

And if we remove analysis *Fe*, the Analysis Request will follow *Au* analysis (that is *verified*):

```
>>> ar.setAnalyses([Au])
>>> api.get_workflow_status_of(ar)
'verified'
```

4.50.4 Remove Analyses from an Analysis Request with all remaining tests verified

Create a new Analysis Request and receive:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Submit and verify results for *Fe*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> analysis_fe = filter(lambda an: an.getKeyword() == "Fe", analyses)[0]
>>> analysis_fe.setResult(12)
>>> transitioned = do_action_for(analysis_fe, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_fe)
'to_be_verified'
>>> transitioned = do_action_for(analysis_fe, "verify")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_fe)
'verified'
```

Submit and verify results for *Au*:

```
>>> analysis_au = filter(lambda an: an.getKeyword() == "Au", analyses)[0]
>>> analysis_au.setResult(14)
>>> transitioned = do_action_for(analysis_au, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_au)
'to_be_verified'
>>> transitioned = do_action_for(analysis_au, "verify")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis_au)
'verified'
```

The Analysis Request status is still *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

But if we remove the analysis without result (*Cu*), the Analysis Request transitions to "verfied" because follows *Fe* and *Au*:

```
>>> ar.setAnalyses([Fe, Au])
>>> api.get_workflow_status_of(ar)
'verified'
```

4.51 Rolemap

Bika LIMS defines several roles for the lab context

4.51.1 How to run this test

Please execute the following command in the *buildout* directory:

```
./bin/test test_textual_doctests -t Rolemap
```

4.51.2 Test Setup

Needed Imports:

```
>>> from bika.lims import api
```

Test Variables:

```
>>> portal = api.get_portal()
>>> acl_users = api.get_tool("acl_users")
```

4.51.3 Check Bika LIMS Roles

Ensure the "Analyst" role exists:

```
>>> role = "Analyst"
>>> role in acl_users.validRoles()
True
```

Ensure the "Client" role exists:

```
>>> role = "Client"
>>> role in acl_users.validRoles()
True
```

Ensure the "LabClerk" role exists:

```
>>> role = "LabClerk"
>>> role in acl_users.validRoles()
True
```

Ensure the "LabManager" role exists:

```
>>> role = "LabManager"
>>> role in acl_users.validRoles()
True
```

Ensure the "Member" role exists:

```
>>> role = "Member"
>>> role in acl_users.validRoles()
True
```

Ensure the "Preserver" role exists:

```
>>> role = "Preserver"
>>> role in acl_users.validRoles()
True
```

Ensure the "Publisher" role exists:

```
>>> role = "Publisher"
>>> role in acl_users.validRoles()
True
```

Ensure the "RegulatoryInspector" role exists:

```
>>> role = "RegulatoryInspector"
>>> role in acl_users.validRoles()
True
```

Ensure the "Reviewer" role exists:

```
>>> role = "Reviewer"
>>> role in acl_users.validRoles()
True
```

Ensure the "Sampler" role exists:

```
>>> role = "Sampler"
>>> role in acl_users.validRoles()
True
```

Ensure the "SamplingCoordinator" role exists:

```
>>> role = "SamplingCoordinator"
>>> role in acl_users.validRoles()
True
```

Ensure the "Verifier" role exists:

```
>>> role = "Verifier"
>>> role in acl_users.validRoles()
True
```

4.52 Secondary Analysis Request

Running this test from the buildout directory:

bin/test test_textual_doctests -t SecondaryAnalysisRequest

4.52.1 Test Setup

Needed Imports:

```
>>> from DateTime import DateTime
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> from bika.lims import api
>>> from bika.lims.interfaces import IAnalysisRequestSecondary
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils.analysisrequest import create_partition
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
```

Some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ["LabManager",])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
```

(continues on next page)

(continued from previous page)

```
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.52.2 Create Secondary Analysis Request

To create a Secondary Analysis Request we need first a primary (or source Analysis Request) to which the secondary analysis request will refer to:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SamplingDate": DateTime(),
...     "DateSampled": DateTime(),
...     "SampleType": sampletype.UID() }
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> primary = create_analysisrequest(client, request, values, service_uids)
>>> primary
<AnalysisRequest at /plone/clients/client-1/W-0001>
```

Receive the primary analysis request:

```
>>> transitioned = do_action_for(primary, "receive")
>>> api.get_workflow_status_of(primary)
'sample_received'
```

Create the Secondary Analysis Request:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SampleType": sampletype.UID(),
...     "PrimaryAnalysisRequest": primary }
```

```
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> secondary = create_analysisrequest(client, request, values, service_uids)
>>> secondary
<AnalysisRequest at /plone/clients/client-1/W-0001-S01>
```

```
>>> secondary.getPrimaryAnalysisRequest()
<AnalysisRequest at /plone/clients/client-1/W-0001>
```

The secondary AnalysisRequest also provides *IAnalysisRequestSecondary*:

```
>>> IAnalysisRequestSecondary.providedBy(secondary)
True
```

Dates match with those from the primary Analysis Request:

```
>>> secondary.getDateSampled() == primary.getDateSampled()
True
```

```
>>> secondary.getSamplingDate() == primary.getSamplingDate()
True
```

The secondary sample is automatically transitioned to *sample_received*:

```
>>> api.get_workflow_status_of(secondary)
'sample_received'
```

The SampleReceived date matches with the primary's:

```
>>> secondary.getDateReceived() == primary.getDateReceived()
True
```

Analyses have been also initialized automatically:

```
>>> analyses = secondary.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

If I create another secondary sample using same AR as the primary:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SampleType": sampletype.UID(),
...     "PrimaryAnalysisRequest": primary }
```

```
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> secondary = create_analysisrequest(client, request, values, service_uids)
```

The ID suffix of the new secondary sample increases in one unit:

```
>>> secondary.getId()
'W-0001-S02'
```

If I create a secondary sample from another secondary AR as the primary:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SampleType": sampletype.UID(),
...     "PrimaryAnalysisRequest": secondary }
```

```
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> third = create_analysisrequest(client, request, values, service_uids)
```

The ID suffix is extended accordingly:

```
>>> third.getId()
'W-0001-S02-S01'
```

And the associated primary AR is the secondary sample we created earlier:

```
>>> third.getPrimaryAnalysisRequest()
<AnalysisRequest at /plone/clients/client-1/W-0001-S02>
```

And of course, keeps same date values:

```
>>> third.getDateSampled() == secondary.getDateSampled()
True
```

```
>>> third.getSamplingDate() == secondary.getSamplingDate()
True
```

```
>>> third.getDateReceived() == secondary.getDateReceived()
True
```

If we change the dates from the root Primary:

```
>>> primary.setSamplingDate(DateTime() + 5)
>>> primary.setDateSampled(DateTime() + 10)
>>> primary.setDateReceived(DateTime() + 15)
```

Dates for secondaries are updated in accordance:

```
>>> third.getSamplingDate() == secondary.getSamplingDate() == primary.
↪getSamplingDate()
True
>>> third.getDateSampled() == secondary.getDateSampled() == primary.getDateSampled()
True
>>> third.getDateReceived() == secondary.getDateReceived() == primary.
↪getDateReceived()
True
```

4.52.3 Secondary Analysis Requests and partitions

When partitions are created from a secondary Analysis Request, the partitions themselves are not considered secondaries from the primary AR, but partitions of a Secondary Analysis Request.

Create a secondary Analysis Request:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SampleType": sampletype.UID(),
...     "PrimaryAnalysisRequest": primary }
```

```
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> secondary = create_analysisrequest(client, request, values, service_uids)
>>> secondary
<AnalysisRequest at /plone/clients/client-1/W-0001-S03>
```

Create a single partition from the secondary Analysis Request:

```
>>> analyses = secondary.getAnalyses()
>>> analyses_1 = analyses[0:1]
>>> analyses_2 = analyses[1:]
>>> partition = create_partition(secondary, request, analyses_1)
>>> partition
<AnalysisRequest at /plone/clients/client-1/W-0001-S03-P01>
```

```
>>> partition.isPartition()
True
```

```
>>> partition.getParentAnalysisRequest()
<AnalysisRequest at /plone/clients/client-1/W-0001-S03>
```

Partition does not provide *IAnalysisRequestSecondary*:

```
>>> IAnalysisRequestSecondary.providedBy(partition)
False
```

And does not keep the original Primary Analysis Request:

```
>>> partition.getPrimaryAnalysisRequest() is None
True
```

If we create another partition, the generated ID is increased in one unit:

```
>>> partition = create_partition(secondary, request, analyses_2)
>>> partition
<AnalysisRequest at /plone/clients/client-1/W-0001-S03-P02>
```

We can even create a secondary Analysis Request from a partition as the source:

```
>>> values = {
...     "Client": client.UID(),
...     "Contact": contact.UID(),
...     "SampleType": sampletype.UID(),
...     "PrimaryAnalysisRequest": partition }
```

```
>>> service_uids = map(api.get_uid, [Cu, Fe, Au])
>>> secondary = create_analysisrequest(client, request, values, service_uids)
>>> secondary
<AnalysisRequest at /plone/clients/client-1/W-0001-S03-P02-S01>
```

But note this new secondary is not considered a partition of a partition:

```
>>> secondary.isPartition()
False
```

But keeps the partition as the primary:

```
>>> secondary.getPrimaryAnalysisRequest()
<AnalysisRequest at /plone/clients/client-1/W-0001-S03-P02>
```

We can also create new partitions from this weird secondary:

```
>>> partition = create_partition(secondary, request, secondary.getAnalyses())
>>> partition
<AnalysisRequest at /plone/clients/client-1/W-0001-S03-P02-S01-P01>
```

4.53 Infinite recursion when fetching dependencies from Service

This test checks that no infinite recursion error arises when fetching the dependencies of a Service (via Calculation) that itself contains a keyword in a calculation from another service bound to a calculation that refers to the first one as well.

Running this test from the buildout directory:

bin/test test_textual_doctests -t ServicesCalculationRecursion.rst

4.53.1 Test Setup

Needed imports:

```
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from bika.lims import api
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = api.get_setup()
```

Create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
```

4.53.2 Creation of Service with a Calculation that refers to itself

The most common case is when the Calculation is assigned to the same Analysis that is referred in the Calculation's formula:

```
>>> Ca = api.create(setup.bika_analysisservices, "AnalysisService", title="Calcium",␣
↪Keyword="Ca", Price="20", Category=category.UID())
>>> Mg = api.create(setup.bika_analysisservices, "AnalysisService", title="Magnesium",
↪ Keyword="Mg", Price="20", Category=category.UID())
```

```
>>> calc = api.create(setup.bika_calculations, "Calculation", title="Total Hardness")
>>> calc.setFormula("[Ca] + [Mg]")
>>> calc.getFormula()
'[Ca] + [Mg]'
```

```
>>> Ca.setCalculation(calc)
>>> Ca.getCalculation()
<Calculation at /plone/bika_setup/bika_calculations/calculation-1>
```

```
>>> deps = Ca.getServiceDependencies()
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

```
>>> deps = calc.getCalculationDependencies()
>>> len(deps.keys())
2
```

```
>>> deps = calc.getCalculationDependencies(flat=True)
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

The other case is when the initial Service is referred indirectly, through a calculation a dependency is bound to:

```
>>> calc_mg = api.create(setup.bika_calculations, "Calculation", title="Test")
>>> calc_mg.setFormula("[Ca] + [Ca]")
>>> calc_mg.getFormula()
'[Ca] + [Ca]'
```

```
>>> Mg.setCalculation(calc_mg)
>>> Mg.getCalculation()
<Calculation at /plone/bika_setup/bika_calculations/calculation-2>
```

```
>>> deps = Mg.getServiceDependencies()
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

```
>>> deps = calc_mg.getCalculationDependencies()
>>> len(deps.keys())
2
```

```
>>> deps = calc_mg.getCalculationDependencies(flat=True)
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

```
>>> deps = Ca.getServiceDependencies()
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

```
>>> deps = calc.getCalculationDependencies()
>>> len(deps.keys())
2
```

```
>>> deps = calc.getCalculationDependencies(flat=True)
>>> sorted(map(lambda d: d.getKeyword(), deps))
['Ca', 'Mg']
```

4.54 Show or Hide Prices

There's a setting in BikaSetup called 'Include and display pricing information'. If this setting is disabled, then no mention of pricing or invoicing should appear in the system. I still allowed the fields for Price to appear in AnalysisService edit form, so that they may be modified while still remaining hidden elsewhere.

Running this test from the buildout directory:

bin/test -t ShowPrices

4.54.1 Test Setup

Needed Imports:

```
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from time import sleep
>>> import transaction
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def enableShowPrices():
...     self.portal.bika_setup.setShowPrices(True)
...     transaction.commit()
```

```
>>> def disableShowPrices():
...     self.portal.bika_setup.setShowPrices(False)
...     transaction.commit()
```

Variables:

```
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> request = self.request
>>> portal = self.portal
>>> bs = portal.bika_setup
>>> laboratory = bs.laboratory
>>> portal_url = portal.absolute_url()
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

Now we need to create some basic content for our tests:

```
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(portal.bika_setup.bika_sampletypes, "SampleType", title=
↪"Water", Prefix="W")
>>> labcontact = api.create(portal.bika_setup.bika_labcontacts, "LabContact",␣
↪Firstname="Lab", Lastname="Manager")
>>> department = api.create(portal.bika_setup.bika_departments, "Department", title=
↪"Chemistry", Manager=labcontact)
>>> category = api.create(portal.bika_setup.bika_analysiscategories, "AnalysisCategory
↪", title="Metals", Department=department)
>>> Cu = api.create(portal.bika_setup.bika_analysisservices, "AnalysisService", title=
↪"Copper", Keyword="Cu", Price="409.17", Category=category.UID(), Accredited=True)
>>> Fe = api.create(portal.bika_setup.bika_analysisservices, "AnalysisService", title=
↪"Iron", Keyword="Fe", Price="208.20", Category=category.UID())
>>> profile = api.create(portal.bika_setup.bika_analysisprofiles, "AnalysisProfile",␣
↪title="Profile", Service=[Fe.UID(), Cu.UID()])
```

(continues on next page)

(continued from previous page)

```
>>> template = api.create(portal.bika_setup.bika_artemplates, "ARTemplate", title=
↪"Template", AnalysisProfile=[profile.UID()])
```

Enable accreditation for the lab

```
>>> laboratory.setLaboratoryAccredited(True)
```

And start a browser:

```
>>> transaction.commit()
>>> browser = self.getBrowser()
```

4.54.2 Analysis Request Add form

Verify that the price and invoice fields are present when ShowPrices is enabled:

```
>>> enableShowPrices()
>>> browser.open(client.absolute_url() + "/ar_add")
```

```
>>> True if "Discount" in browser.contents else "ShowPrices is True, and Discount␣
↪field is missing from AR Add."
True
>>> True if "Subtotal" in browser.contents else "ShowPrices is True, and Subtotal␣
↪field is missing from AR Add."
True
>>> True if "VAT" in browser.contents else "ShowPrices is True, and VAT field is␣
↪missing from AR Add."
True
>>> True if "Total" in browser.contents else "ShowPrices is True, and Total field is␣
↪missing from AR Add."
True
>>> True if "Invoice Exclude" in browser.contents else "ShowPrices is True, and␣
↪Invoice Exclude field is missing from AR Add."
True
```

And then that the opposite is true:

```
>>> disableShowPrices()
>>> browser.open(client.absolute_url() + "/ar_add")
```

```
>>> True if "Discount" not in browser.contents else "ShowPrices is False, Discount␣
↪field should not be present in AR Add."
True
>>> True if "Subtotal" not in browser.contents else "ShowPrices is False, Subtotal␣
↪field should not be present in AR Add."
True
>>> True if "VAT" not in browser.contents else "ShowPrices is False, VAT field should␣
↪not be present in AR Add."
True
>>> True if "Total" not in browser.contents else "ShowPrices is False, Total field␣
↪should not be present in AR Add."
True
>>> True if "Invoice Exclude" not in browser.contents else "ShowPrices is False,␣
↪Invoice Exclude field should not be present in AR Add."
True
```

Disable MemberDiscountApplies, and verify that it always vanishes from AR add:

```
>>> client.setMemberDiscountApplies(False)
>>> transaction.commit()
```

```
>>> enableShowPrices()
>>> browser.open(client.absolute_url() + "/ar_add")
>>> True if "Discount" not in browser.contents else "Discount field should be hidden."
True
>>> disableShowPrices()
>>> browser.open(client.absolute_url() + "/ar_add")
>>> True if "Discount" not in browser.contents else "Discount field should be hidden."
True
```

4.54.3 Analysis Request View

Test show/hide prices when viewing an AR. First, create an AR:

```
>>> values = {
...     'Client': client.UID(),
...     'Contact': contact.UID(),
...     'DateSampled': date_now,
...     'SampleType': sampletype.UID()}
>>> service_uids = [Cu.UID(), Fe.UID()]
>>> ar = create_analysisrequest(client, request, values, service_uids)
```

~~ TODO: Fails because barceloeanata theme loaded?! With ShowPrices enabled, the Invoice tab should be rendered:

> enableShowPrices() browser.open(ar.absolute_url()) True if 'contentview-invoice' in browser.contents else "Invoice Tab is not visible, but ShowPrices is True." True

And when ShowPrices is off, the Invoice tab should not be present at all:

> disableShowPrices() browser.open(ar.absolute_url()) True if 'contentview-invoice' not in browser.contents else "Invoice Tab is visible, but ShowPrices is False." True

4.54.4 Client discount fields show/hide

```
>>> enableShowPrices()
>>> browser.open(client.absolute_url() + "/edit")
>>> True if 'discount' in browser.contents else "Client discount field should be␣
↪visible, but is not"
True
```

```
>>> disableShowPrices()
>>> browser.open(client.absolute_url() + "/edit")
>>> True if 'discount' not in browser.contents else "Client discount field should not␣
↪be visible, but here it is"
True
```

4.55 Specification and Results Ranges with Samples and analyses

Specification is an object containing a list of results ranges, each one refers to the min/max/min_warn/max_warn values to apply for a given analysis service. User can assign a Specification to a Sample, so the results of it's Analyses will be checked against the results ranges provided by the Specification.

Running this test from the buildout directory:

bin/test test_textual_doctests -t SpecificationAndResultsRanges.rst

4.55.1 Test Setup

Needed imports:

```
>>> import transaction
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.utils.analysisrequest import create_partition
>>> from bika.lims.workflow import doActionFor as do_action_for
```

Functional Helpers:

```
>>> def new_sample(services, specification=None, results_ranges=None):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime().strftime("%Y-%m-%d"),
...         'SampleType': sampletype.UID(),
...         'Analyses': map(api.get_uid, services),
...         'Specification': specification or None }
...
...     ar = create_analysisrequest(client, request, values, results_ranges=results_
↪ranges)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def get_analysis_from(sample, service):
...     service_uid = api.get_uid(service)
...     for analysis in sample.getAnalyses(full_objects=True):
...         if analysis.getServiceUID() == service_uid:
...             return analysis
...     return None
```

```
>>> def get_results_range_from(obj, service):
...     field = obj.getField("ResultsRange")
...     return field.get(obj, search_by=api.get_uid(service))
```

```
>>> def set_results_range_for(obj, results_range):
...     rrs = obj.getResultsRange()
...     uid = results_range["uid"]
...     rrs = filter(lambda rr: rr["uid"] != uid, rrs)
```

(continues on next page)

(continued from previous page)

```
...     rrs.append(results_range)
...     obj.setResultsRange(rrs)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = api.get_setup()
```

Create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID())
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Mg = api.create(setup.bika_analysisservices, "AnalysisService", title="Magnesium",
↪ Keyword="Mg", Price="20", Category=category.UID())
>>> Zn = api.create(setup.bika_analysisservices, "AnalysisService", title="Zinc",␣
↪Keyword="Zn", Price="10", Category=category.UID())
```

Create an Analysis Specification for *Water*:

```
>>> sampletype_uid = api.get_uid(sampletype)
>>> rr1 = {"uid": api.get_uid(Au), "min": 10, "max": 20, "warn_min": 5, "warn_max":␣
↪25}
>>> rr2 = {"uid": api.get_uid(Cu), "min": 20, "max": 30, "warn_min": 15, "warn_max":␣
↪35}
>>> rr3 = {"uid": api.get_uid(Fe), "min": 30, "max": 40, "warn_min": 25, "warn_max":␣
↪45}
>>> rr4 = {"uid": api.get_uid(Mg), "min": 40, "max": 50, "warn_min": 35, "warn_max":␣
↪55}
>>> rr5 = {"uid": api.get_uid(Zn), "min": 50, "max": 60, "warn_min": 45, "warn_max":␣
↪65}
>>> rr = [rr1, rr2, rr3, rr4, rr5]
>>> specification = api.create(setup.bika_analysisspecs, "AnalysisSpec", title="Lab␣
↪Water Spec", SampleType=sampletype_uid, ResultsRange=rr)
```

4.55.2 Creation of a Sample with Specification

A given Specification can be assigned to the Sample during the creation process. The results ranges of the mentioned Specification will be stored in ResultsRange field from the Sample and the analyses will acquire those results ranges individually.

Specification from Sample is history-aware, so even if the Specification object is changed after its assignment to the Sample, the Results Ranges from either the Sample and its Analyses will remain untouched.

Create a Sample and receive:

```
>>> services = [Au, Cu, Fe, Mg]
>>> sample = new_sample(services, specification=specification)
```

The sample has the specification assigned:

```
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

And its results ranges match with the sample's *ResultsRange* field value:

```
>>> specification.getResultsRange() == sample.getResultsRange()
True
```

And the analyses the sample contains have the results ranges properly set:

```
>>> au = get_analysis_from(sample, Au)
>>> au.getResultsRange() == get_results_range_from(specification, Au)
True
```

```
>>> cu = get_analysis_from(sample, Cu)
>>> cu.getResultsRange() == get_results_range_from(specification, Cu)
True
```

```
>>> fe = get_analysis_from(sample, Fe)
>>> fe.getResultsRange() == get_results_range_from(specification, Fe)
True
```

```
>>> mg = get_analysis_from(sample, Mg)
>>> mg.getResultsRange() == get_results_range_from(specification, Mg)
True
```

We can change a result range by using properties:

```
>>> rr_au = au.getResultsRange()
>>> rr_au.min = 11
>>> rr_au.max = 21
>>> (rr_au.min, rr_au.max)
(11, 21)
```

Or using it as a dict:

```
>>> rr_au["min"] = 15
>>> rr_au["max"] = 25
>>> (rr_au["min"], rr_au["max"])
(15, 25)
```

If we change this results range in the Specification object, this won't take any effect to neither the Sample nor analyses:

```
>>> set_results_range_for(specification, rr_au)
>>> specification.getResultsRange() == sample.getResultsRange()
False
```

```
>>> au.getResultsRange() == get_results_range_from(specification, Au)
False
```

```
>>> get_results_range_from(sample, Au) == au.getResultsRange()
True
```

```
>>> rr_sample_au = au.getResultsRange()
>>> (rr_sample_au.min, rr_sample_au.max)
(10, 20)
```

If we re-apply the Specification, nothing will change though, because its *uid* is still the same:

```
>>> sample.setSpecification(specification)
>>> specification.getResultsRange() == sample.getResultsRange()
False
```

But the ResultsRange value from Sample is updated accordingly if we set the specification to *None* first:

```
>>> sample.setSpecification(None)
>>> sample.setSpecification(specification)
>>> specification.getResultsRange() == sample.getResultsRange()
True
```

As well as the analyses the sample contains:

```
>>> au.getResultsRange() == get_results_range_from(specification, Au)
True
```

```
>>> rr_sample_au = au.getResultsRange()
>>> (rr_sample_au.min, rr_sample_au.max)
(15, 25)
```

4.55.3 Removal of Analyses from a Sample with Specifications

User can remove analyses from the Sample. If the user removes one of the analyses, the Specification assigned to the Sample will remain intact, as well as Sample's Results Range:

```
>>> sample.setAnalyses([Au, Cu, Fe])
>>> analyses = sample.objectValues()
>>> sorted(analyses, key=lambda an: an.getKeyword())
[<Analysis at /plone/clients/client-1/W-0001/Au>, <Analysis at /plone/clients/client-
↪1/W-0001/Cu>, <Analysis at /plone/clients/client-1/W-0001/Fe>]
```

```
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> specification.getResultsRange() == sample.getResultsRange()
True
```

4.55.4 Addition of Analyses to a Sample with Specifications

User can add new analyses to the Sample as well. If the Sample has an Specification set and the specification had a results range registered for such analysis, the result range for the new analysis will be set automatically:

```
>>> sample.setAnalyses([Au, Cu, Fe, Zn])
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> zn = get_analysis_from(sample, Zn)
>>> zn.getResultsRange() == get_results_range_from(specification, Zn)
True
```

If we reset an Analysis with it's own ResultsRange, different from the range defined by the Specification, the system does not clear the Specification:

```
>>> rr_zn = zn.getResultsRange()
>>> rr_zn.min = 55
>>> sample.setAnalyses([Au, Cu, Fe, Zn], specs=[rr_zn])
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

and Sample's ResultsRange is kept unchanged:

```
>>> sample_rr = sample.getResultsRange()
>>> len(sample_rr)
5
```

with result range for *Zn* unchanged:

```
>>> sample_rr_zn = sample.getResultsRange(search_by=api.get_uid(Zn))
>>> sample_rr_zn.min
50
```

But analysis' result range has indeed changed:

```
>>> zn.getResultsRange().min
55
```

If we re-apply the Specification, the result range for *Zn*, as well as for the Sample, are reestablished:

```
>>> sample.setSpecification(None)
>>> sample.setSpecification(specification)
>>> specification.getResultsRange() == sample.getResultsRange()
True
```

```
>>> zn.getResultsRange() == get_results_range_from(specification, Zn)
True
```

```
>>> zn.getResultsRange().min
50
```

4.55.5 Sample with Specifications and Partitions

When a sample has partitions, the Specification set to the root Sample is populated to all its descendants:

```
>>> partition = create_partition(sample, request, [zn])
>>> partition
<AnalysisRequest at /plone/clients/client-1/W-0001-P01>
```

```
>>> zn = get_analysis_from(partition, Zn)
>>> zn
<Analysis at /plone/clients/client-1/W-0001-P01/Zn>
```

The partition keeps the Specification and ResultsRange by its own:

```
>>> partition.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> partition.getResultsRange() == specification.getResultsRange()
True
```

If we reset an Analysis with it's own ResultsRange, different from the range defined by the Specification, the system does not clear the Specification, neither from the root sample nor the partition:

```
>>> rr_zn = zn.getResultsRange()
>>> rr_zn.min = 56
>>> partition.setAnalyses([Zn], specs=[rr_zn])
```

```
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> partition.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

And Results Range from both Sample and partition are kept untouched:

```
>>> sample.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> sample.getResultsRange() == specification.getResultsRange()
True
```

```
>>> partition.getSpecification()
<AnalysisSpec at /plone/bika_setup/bika_analysisspecs/analysisspec-1>
```

```
>>> partition.getResultsRange() == specification.getResultsRange()
True
```

4.56 Sysmex xt i1800 import interface

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t SysmexXTi1800ImportInterface
```

4.56.1 Test Setup

Needed imports:

```
>>> import os
>>> import transaction
>>> from Products.CMFCore.utils import getToolByName
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from DateTime import DateTime

>>> import codecs
>>> from senaite.core.exportimport import instruments
>>> from senaite.core.exportimport.instruments.sysmex.xt import SysmexXTImporter
>>> from senaite.core.exportimport.instruments.sysmex.xt.i1800 import TX1800iParser
>>> from bika.lims.browser.resultsimport.resultsimport import ConvertToUploadFile
```

Functional helpers:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_instruments = bika_setup.bika_instruments
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.56.2 Availability of instrument interface

Check that the instrument interface is available:

```
>>> exims = []
>>> for exim_id in instruments.__all__:
...     exims.append(exim_id)
>>> 'sysmex.xt.i1800' in exims
True
```

4.56.3 Assigning the Import Interface to an Instrument

Create an *Instrument* and assign to it the tested Import Interface:

```
>>> instrument = api.create(bika_instruments, "Instrument", title="Instrument-1")
>>> instrument
<Instrument at /plone/bika_setup/bika_instruments/instrument-1>
```

(continues on next page)

(continued from previous page)

```
>>> instrument.setImportDataInterface(['sysmex.xt.i1800'])
>>> instrument.getImportDataInterface()
['sysmex.xt.i1800']
```

4.56.4 Import test

Required steps: Create and receive Analysis Request for import test

An *AnalysisRequest* can only be created inside a *Client*, and it also requires a *Contact* and a *SampleType*:

```
>>> clients = self.portal.clients
>>> client = api.create(clients, "Client", Name="NARALABS", ClientID="NLABS")
>>> client
<Client at /plone/clients/client-1>
>>> contact = api.create(client, "Contact", Firstname="Juan", Surname="Gallostra")
>>> contact
<Contact at /plone/clients/client-1/contact-1>
>>> sampletype = api.create(bika_sampletypes, "SampleType", Prefix="H2O",␣
↪MinimumVolume="100 ml")
>>> sampletype
<SampleType at /plone/bika_setup/bika_sampletypes/sampletype-1>
```

Create an *AnalysisCategory* (which categorizes different *AnalysisServices*), and add to it some of the *AnalysisServices* that are found in the results file:

```
>>> analysiscategory = api.create(bika_analysiscategories, "AnalysisCategory", title=
↪"Water")
>>> analysiscategory
<AnalysisCategory at /plone/bika_setup/bika_analysiscategories/analysiscategory-1>
>>> analysisservice_1 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="WBC",
...                                ShortTitle="wbc",
...                                Category=analysiscategory,
...                                Keyword="WBC")
>>> analysisservice_1
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-1>
>>> analysisservice_2 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="RBC",
...                                ShortTitle="rbc",
...                                Category=analysiscategory,
...                                Keyword="RBC")
>>> analysisservice_2
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-2>
>>> analysisservice_3 = api.create(bika_analysisservices,
...                                "AnalysisService",
...                                title="HGB",
...                                ShortTitle="hgb",
...                                Category=analysiscategory,
...                                Keyword="HGB")
>>> analysisservice_3
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-3>
>>> analysisservice_4 = api.create(bika_analysisservices,
```

(continues on next page)

(continued from previous page)

```
...                                "AnalysisService",
...                                title="HCT",
...                                ShortTitle="hct",
...                                Category=analysiscategory,
...                                Keyword="HCT")
>>> analysisservice_4
<AnalysisService at /plone/bika_setup/bika_analysisservices/analysisservice-4>
>>> analysisservices = [analysisservice_1, analysisservice_2, analysisservice_3,␣
↪analysisservice_4]
```

Create an *AnalysisRequest* with this *AnalysisService* and receive it:

```
>>> values = {
...            'Client': client.UID(),
...            'Contact': contact.UID(),
...            'SamplingDate': date_now,
...            'DateSampled': date_now,
...            'SampleType': sampletype.UID()
...           }
>>> service_uids = [analysisservice.UID() for analysisservice in analysisservices]
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> ar
<AnalysisRequest at /plone/clients/client-1/H2O-0001>
>>> ar.getReceivedBy()
''
>>> wf = getToolByName(ar, 'portal_workflow')
>>> wf.doActionFor(ar, 'receive')
>>> ar.getReceivedBy()
'test_user_1_'
```

Import test

Load results test file and import the results:

```
>>> dir_path = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..', 'files
↪'))
>>> temp_file = codecs.open(dir_path + '/2012-05-09_11-06-14-425_CBDB6A.txt',
...                         encoding='utf-8-sig')
>>> test_file = ConvertToUploadFile(temp_file)
>>> tx1800i_parser = TX1800iParser(test_file)
>>> importer = SysmexXTImporter(parser=tx1800i_parser,
...                             context=portal,
...                             allowed_ar_states=['sample_received', 'attachment_due
↪', 'to_be_verified'],
...                             allowed_analysis_states=None,
...                             override=[True, True])
>>> importer.process()
```

Check from the importer logs that the file from where the results have been imported is indeed the specified file:

```
>>> '2012-05-09_11-06-14-425_CBDB6A.txt' in importer.logs[0]
True
```

Check the rest of the importer logs to verify that the values were correctly imported:

```
>>> importer.logs[1:]
['End of file reached successfully: 1 objects, 21 analyses, 1 results',
 'Allowed Sample states: sample_received, attachment_due, to_be_verified',
 'Allowed analysis states: unassigned, assigned, to_be_verified',
 "H2O-0001: [u'Analysis HCT', u'Analysis RBC', u'Analysis WBC', u'Analysis HGB']␣
↪imported sucessfully",
 'Import finished successfully: 1 Samples and 4 results updated']
```

And finally check if indeed the analysis has the imported results:

```
>>> analyses = ar.getAnalyses()
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title=='WBC'][0]
>>> an.getResult()
'6.01'
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title=='RBC'][0]
>>> an.getResult()
'5.02'
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title=='HGB'][0]
>>> an.getResult()
'13.2'
>>> an = [analysis.getObject() for analysis in analyses if analysis.Title=='HCT'][0]
>>> an.getResult()
'40.0'
```

4.57 Sysmex xt i4000 import interface

Running this test from the buildout directory:

```
bin/test test_textual_doctests -t SysmexXTi4000ImportInterface
```

4.57.1 Notes

Since the Sysmex xt i4000 uses the same parser and importer than the Sysmex xt i1800 this test only tests that the import interface of the i4000 can be assigned to an instrument. The functional tests for the parser and importer can be found in the tests for the Sysmex xt i1800.

4.57.2 Test Setup

Needed imports:

```
>>> import os
>>> import transaction
>>> from Products.CMFCore.utils import getToolByName
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from DateTime import DateTime

>>> import codecs
>>> from senaite.core.exportimport import instruments
>>> from senaite.core.exportimport.instruments.sysmex.xt import SysmexXTImporter
>>> from senaite.core.exportimport.instruments.sysmex.xt.i1800 import TX1800iParser
>>> from bika.lims.browser.resultsimport.resultsimport import ConvertToUploadFile
```

Functional helpers:

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Variables:

```
>>> date_now = timestamp()
>>> portal = self.portal
>>> request = self.request
>>> bika_setup = portal.bika_setup
>>> bika_instruments = bika_setup.bika_instruments
>>> bika_sampletypes = bika_setup.bika_sampletypes
>>> bika_samplepoints = bika_setup.bika_samplepoints
>>> bika_analysiscategories = bika_setup.bika_analysiscategories
>>> bika_analysisservices = bika_setup.bika_analysisservices
```

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager:

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
```

4.57.3 Availability of instrument interface

Check that the instrument interface is available:

```
>>> exims = []
>>> for exim_id in instruments.__all__:
...     exims.append(exim_id)
>>> 'sysmex.xt.i4000' in exims
True
```

4.57.4 Assigning the Import Interface to an Instrument

Create an *Instrument* and assign to it the tested Import Interface:

```
>>> instrument = api.create(bika_instruments, "Instrument", title="Instrument-1")
>>> instrument
<Instrument at /plone/bika_setup/bika_instruments/instrument-1>
>>> instrument.setImportDataInterface(['sysmex.xt.i4000'])
>>> instrument.getImportDataInterface()
['sysmex.xt.i4000']
```

4.58 UIDReferenceField

UIDReferenceField is a drop-in replacement for Plone's ReferenceField which uses a StringField to store a UID or a list of UIDs.

Running this test from the buildout directory:

    bin/test test_textual_doctests -t UIDReferenceField

Needed Imports:

```
>>> import re
>>> from bika.lims import api
>>> from bika.lims.browser.fields.uidreferencefield import get_backreferences
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bika_calculations = portal.bika_setup.bika_setup.bika_calculations
>>> bika_analysisservices = portal.bika_setup.bika_setup.bika_analysisservices
```

Test user:

We need certain permissions to create and access objects used in this test, so here we will assume the role of Lab Manager.

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import setRoles
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

I'll test using the relationship between Calculations and AnalysisServices.

First I'll create some AnalysisServices and Calculations:

```
>>> as1 = api.create(bika_analysisservices, "AnalysisService", title="AS 1")
>>> as1.setKeyword("as1")
>>> as1.reindexObject()
>>> as2 = api.create(bika_analysisservices, "AnalysisService", title="AS 2")
>>> as2.setKeyword("as2")
>>> as2.reindexObject()
>>> as3 = api.create(bika_analysisservices, "AnalysisService", title="AS 3")
>>> as3.setKeyword("as3")
>>> as3.reindexObject()
>>> c1 = api.create(bika_calculations, "Calculation", title="C 1")
>>> c2 = api.create(bika_calculations, "Calculation", title="C 2")
```

Put some AS Keywords into the *Formula* field of the calculations, which will cause their DependentServices field (a UIDReferenceField) to be populated.

```
>>> c1.setFormula("[as1] + [as2] + [as3]")
>>> c2.setFormula("[as1] + [as2]")
```

c1 now depends on three services:

```
>>> deps = [s.Title() for s in c1.getDependentServices()]
>>> deps.sort()
>>> deps
['AS 1', 'AS 2', 'AS 3']
```

c2 now depends on two services:

```
>>> deps = [s.Title() for s in c2.getDependentServices()]
>>> deps.sort()
>>> deps
['AS 1', 'AS 2']
```

Backreferences are stored on each object which is a target of a UIDReferenceField. This allows a service to ask, "which calculations include me in their DependentServices?":

```
>>> get_backreferences(as1, 'CalculationDependentServices')
['...', '...']
```

It also allows to find out which services have selected a particular calculation as their primary Calculation field's value:

```
>>> as3.setCalculation(c2)
>>> get_backreferences(c2, 'AnalysisServiceCalculation')
['...']
```

The value will always be a list of UIDs, unless as_brains is True:

```
>>> get_backreferences(c2, 'AnalysisServiceCalculation', as_brains=1)
[<Products.ZCatalog.Catalog.mybrains object at ...>]
```

If no relationship is specified when calling get_backreferences, then a dict is returned (by reference) containing UIDs of all references for all relations. Modifying this dict in-place, will cause the backreferences to be changed!

```
>>> get_backreferences(as1)
{'CalculationDependentServices': ['...', '...']}
```

When requesting the entire set of all backreferences only UIDs may be returned, and it is an error to request brains:

```
>>> get_backreferences(as1, as_brains=True)
Traceback (most recent call last):
...
AssertionError: You cannot use as_brains with no relationship
```

4.59 Versioning

Some Bika LIMS contents support versioning.

Each edit & save process creates a new version, which is triggered by the *ObjectEditedEvent* from *Products.CMFEditions* package.

4.59.1 Test Setup

```
>>> from Acquisition import aq_base
>>> from plone import api as ploneapi
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
```

```
>>> portal = self.portal
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Manager', 'Owner'])
```

```
>>> def is_versionable(obj):
...     pr = ploneapi.portal.get_tool("portal_repository")
...     return pr.supportsPolicy(obj, 'at_edit_autoversion') and pr.isVersionable(obj)
```

```
>>> def get_version(obj):
...     if not is_versionable(obj):
...         return None
...     return getattr(aq_base(obj), "version_id", None)
```

4.59.2 Versionable Types

The following types support versioning:

```
>>> versionable_types = ["AnalysisService", "Calculation"]
```

4.59.3 Analysis Services

Create a analysis service for testing:

```
>>> analysisservices = self.portal.bika_setup.bika_analysisservices
>>> _ = analysisservices.invokeFactory("AnalysisService", id="tempId", title="Test␣
↪Analysis Service 1")
>>> analysisservice = analysisservices.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> analysisservice.processForm()
```

Calcuations should be versionable:

```
>>> is_versionable(analysisservice)
True

>>> get_version(analysisservice)
0
```

Create a new version – for testing, it is sufficient to call the *process_form* method, as this is also called after the content has been edited:

```
>>> analysisservice.processForm()

>>> get_version(analysisservice)
1
```

4.59.4 Calculations

Create a calculation for testing:

```
>>> calculations = self.portal.bika_setup.bika_calculations
>>> _ = calculations.invokeFactory("Calculation", id="tempId", title="Test␣
↪Calculation 1")
>>> calculation = calculations.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> calculation.processForm()
```

Calcuations should be versionable:

```
>>> is_versionable(calculation)
True

>>> get_version(calculation)
0
```

Create a new version – for testing, it is sufficient to call the *process_form* method, as this is also called after the content has been edited:

```
>>> calculation.processForm()

>>> get_version(calculation)
1
```

4.59.5 Non Versionable Types

The following types used to be versionable in Bika LIMS in the past.

Methods

Create a method for testing:

```
>>> methods = self.portal.methods
>>> _ = methods.invokeFactory("Method", id="tempId", title="Test Method 1")
>>> method = methods.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> method.processForm()
```

Methods shouldn't be versionable:

```
>>> is_versionable(method)
False
```

Sample Points

Create a sample point for testing:

```
>>> samplepoints = self.portal.bika_setup.bika_samplepoints
>>> _ = samplepoints.invokeFactory("SamplePoint", id="tempId", title="Test Sample␣
↪Point 1")
>>> samplepoint = samplepoints.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> samplepoint.processForm()
```

Calcuations should be versionable:

```
>>> is_versionable(samplepoint)
False
```

Sample Types

Create a sample type for testing:

```
>>> sampletypes = self.portal.bika_setup.bika_sampletypes
>>> _ = sampletypes.invokeFactory("SampleType", id="tempId", title="Test Sample Point␣
↪1")
>>> sampletype = sampletypes.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> sampletype.processForm()
```

Calcuations should be versionable:

```
>>> is_versionable(sampletype)
False
```

Storage Locations

Create a sample type for testing:

```
>>> storagelocations = self.portal.bika_setup.bika_storagelocations
>>> _ = storagelocations.invokeFactory("StorageLocation", id="tempId", title="Test␣
↪Sample Point 1")
>>> storagelocation = storagelocations.get(_)
```

Process Form to notify Bika about the new content type:

```
>>> storagelocation.processForm()
```

Calcuations should be versionable:

```
>>> is_versionable(storagelocation)
False
```

4.60 Analysis assign guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisAssign

4.60.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.60.2 Assign transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
```

The status of the analyses is *unassigned*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
>>> sorted((map(lambda an: an.getKeyword(), worksheet.getAnalyses())))
['Au', 'Cu', 'Fe']
```

Analyses have been transitioned to *assigned*:

```
>>> map(api.get_workflow_status_of, analyses)
['assigned', 'assigned', 'assigned']
```

And all them are associated to the worksheet:

```
>>> ws_uid = api.get_uid(worksheet)
>>> filter(lambda an: an.getWorksheetUID() != ws_uid, analyses)
[]
```

Analyses do not have an Analyst assigned, though:

```
>>> filter(lambda an: an.getAnalyst(), analyses)
[]
```

If I assign a user to the Worksheet, same user will be assigned to analyses:

```
>>> worksheet.setAnalyst(TEST_USER_ID)
>>> worksheet.getAnalyst() == TEST_USER_ID
True
```

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, analyses)
[]
```

I can remove an analysis from the Worksheet:

```
>>> cu = filter(lambda an: an.getKeyword() == "Cu", analyses)[0]
>>> cu_uid = api.get_uid(cu)
>>> worksheet.removeAnalysis(cu)
>>> filter(lambda an: api.get_uid(an) == cu_uid, worksheet.getAnalyses())
[]
```

So the state of cu is now *unassigned*:

```
>>> api.get_workflow_status_of(cu)
'unassigned'
```

The Analyst is no longer assigned to the analysis:

```
>>> cu.getAssignedAnalyst()
''
```

From *assigned* state I can do submit:

```
>>> au = filter(lambda an: an.getKeyword() == "Au", analyses)[0]
>>> api.get_workflow_status_of(au)
'assigned'
>>> au.setResult(20)
>>> try_transition(au, "submit", "to_be_verified")
True
```

And the analysis transitions to *to_be_verified*:

```
>>> api.get_workflow_status_of(au)
'to_be_verified'
```

While keeping the Analyst that was assigned to the worksheet:

```
>>> au.getAnalyst() == TEST_USER_ID
True
```

And since there is still one analysis in the Worksheet not yet submitted, the Worksheet remains in *open* state:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

But if I remove the remaining analysis, the status of the Worksheet is promoted to *to_be_verified*, cause all the analyses assigned are in this state:

```
>>> fe = filter(lambda an: an.getKeyword() == "Fe", analyses)[0]
>>> worksheet.removeAnalysis(fe)
>>> fe.getWorksheet() is None
```

(continues on next page)

(continued from previous page)

```
True
>>> api.get_workflow_status_of(fe)
'unassigned'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

In *to_be_verified* status, I cannot remove analyses:

```
>>> worksheet.removeAnalysis(au)
>>> map(lambda an: an.getKeyword(), worksheet.getAnalyses())
['Au']
>>> au.getWorksheetUID() == api.get_uid(worksheet)
True
>>> api.get_workflow_status_of(au)
'to_be_verified'
```

But I can still add more analyses:

```
>>> worksheet.addAnalysis(fe)
>>> filter(lambda an: an.getKeyword() == "Fe", worksheet.getAnalyses())
[<Analysis at /plone/clients/client-1/W-0001/Fe>]
```

Causing the Worksheet to roll back to *open* status:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

If I remove *Fe* analysis again, worksheet is promoted to *to_be_verified*:

```
>>> worksheet.removeAnalysis(fe)
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

Let's create another worksheet and add the remaining analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalysis(cu)
>>> worksheet.addAnalysis(fe)
>>> sorted((map(lambda an: an.getKeyword(), worksheet.getAnalyses())))
['Cu', 'Fe']
```

The status of the analyses is now *assigned*:

```
>>> api.get_workflow_status_of(cu)
'assigned'
>>> api.get_workflow_status_of(fe)
'assigned'
```

And I cannot re-assign:

```
>>> isTransitionAllowed(cu, "assign")
False
```

Submit results:

```
>>> cu.setResult(12)
>>> fe.setResult(12)
>>> try_transition(cu, "submit", "to_be_verified")
True
>>> try_transition(fe, "submit", "to_be_verified")
True
```

State of the analyses and worksheet is *to_be_verified*:

```
>>> api.get_workflow_status_of(cu)
'to_be_verified'
>>> api.get_workflow_status_of(fe)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

4.60.3 Check permissions for Assign transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
```

The status of the analysis is *registered*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['registered']
```

But *assign* is not allowed unless we receive the Analysis Request so the analysis is automatically transitioned to *unassigned* state:

```
>>> isTransitionAllowed(analysis, "assign")
False
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned']
```

Exactly these roles can assign:

```
>>> analysis = analyses[0]
>>> get_roles_for_permission("senaite.core: Transition: Assign Analysis", analysis)
['Analyst', 'LabClerk', 'LabManager', 'Manager']
```

Current user can assign because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "assign")
True
```

Users with roles *Analyst* or *LabClerk* can assign too:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst',])
>>> isTransitionAllowed(analysis, "assign")
True
>>> setRoles(portal, TEST_USER_ID, ['LabClerk',])
```

(continues on next page)

(continued from previous page)

```
>>> isTransitionAllowed(analysis, "assign")
True
```

Although other roles cannot:

```
>>> setRoles(portal, TEST_USER_ID, ['Authenticated', 'Owner'])
>>> isTransitionAllowed(analysis, "assign")
False
```

Reset settings:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.61 Analysis multi-verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisMultiVerify

4.61.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getReviewHistory
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
```

(continues on next page)

(continued from previous page)

```
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.61.2 Multiverify not allowed if multi-verification is not enabled

Enable the self verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu])
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> analysis = analyses[0]
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

I cannot multi verify the analysis because multi-verification is not set:

```
>>> isTransitionAllowed(analysis, "multi_verify")
False
>>> try_transition(analysis, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

But I can verify:

```
>>> isTransitionAllowed(analysis, "verify")
True
>>> try_transition(analysis, "verify", "verified")
True
```

And the status of the analysis and others is now *verified*:

```
>>> api.get_workflow_status_of(analysis)
'verified'
>>> api.get_workflow_status_of(ar)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.61.3 Multiverify transition with multi-verification enabled

The system allows to set multiple verifiers, both at Setup or Analysis Service level. If set, the analysis will transition to verified when the total number of verifications equals to the value set in multiple-verifiers.

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Not allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_disabled')
```

Create an Analysis Request, a worksheet and submit results:

```
>>> ar = new_ar([Cu])
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     worksheet.addAnalysis(analysis)
>>> submit_analyses(ar)
```

The status of the Analysis Request, the Worksheet and analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

I cannot *verify*:

```
>>> isTransitionAllowed(analysis, "verify")
False
>>> try_transition(analysis, "verify", "verified")
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

Because multi-verification is enabled:

```
>>> bikasetup.getNumberOfRequiredVerifications()
3
```

And there are 3 verifications remaining:

```
>>> analysis.getNumberOfRemainingVerifications()
3
```

But I can multi-verify:

```
>>> isTransitionAllowed(analysis, "multi_verify")
True
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
```

The status of the analysis and others is still *to_be_verified*:

```
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And my user id is recorded as such:

```
>>> action = getReviewHistory(analysis)[0]
>>> action['actor'] == TEST_USER_ID
True
```

And now, there are two verifications remaining:

```
>>> analysis.getNumberOfRemainingVerifications()
2
```

So, I cannot verify yet:

```
>>> isTransitionAllowed(analysis, "verify")
False
>>> try_transition(analysis, "verify", "verified")
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

But I cannot multi-verify neither, cause I am the same user who did the last multi-verification:

```
>>> isTransitionAllowed(analysis, "multi_verify")
False
>>> try_transition(analysis, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

And the system is configured to not allow same user to verify multiple times:

```
>>> bikasetup.getTypeOfmultiVerification()
'self_multi_disabled'
```

But I can multi-verify if I change the type of multi-verification:

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
>>> isTransitionAllowed(analysis, "multi_verify")
True
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
```

The status of the analysis and others is still *to_be_verified*:

```
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And now, there is only verifications remaining:

```
>>> analysis.getNumberOfRemainingVerifications()
1
```

Since there is only one verification remaining, I cannot multi-verify again:

```
>>> isTransitionAllowed(analysis, "multi_verify")
False
>>> try_transition(analysis, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

But now, I can verify:

```
>>> isTransitionAllowed(analysis, "verify")
True
>>> try_transition(analysis, "verify", "verified")
True
```

There is no verifications remaining:

```
>>> analysis.getNumberOfRemainingVerifications()
0
```

And the status of the analysis and others is now *verified*:

```
>>> api.get_workflow_status_of(analysis)
'verified'
>>> api.get_workflow_status_of(ar)
'verified'
>>> api.get_workflow_status_of(worksheet)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.61.4 Check permissions for Multi verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to “Allow same user to verify multiple times”:

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
```

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu])
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['to_be_verified']
```

Exactly these roles can multi-verify:

```
>>> analysis = analyses[0]
>>> get_roles_for_permission("senaite.core: Transition: Verify", analysis)
['LabManager', 'Manager', 'Verifier']
```

Current user can multi-verify because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "multi_verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(analysis, "multi_verify")
True
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(analysis, "multi_verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(analysis, "multi_verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(analysis, "multi_verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(analysis, "multi_verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.62 Analysis publication guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisPublish

4.62.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getAllowedTransitions
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
```

(continues on next page)

(continued from previous page)

```
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def verify_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         do_action_for(analysis, "verify")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> bikasetup.setSelfVerificationEnabled(True)
```

4.62.2 Publish transition and guard basic constraints

Create an Analysis Request, submit results and verify:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> submit_analyses(ar)
>>> verify_analyses(ar)
>>> api.get_workflow_status_of(ar)
'verified'
```

I cannot publish the analyses individually:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> try_transition(analyses[0], "publish", "published")
False
>>> api.get_workflow_status_of(analyses[0])
'verified'
```

```
>>> try_transition(analyses[1], "publish", "published")
False
>>> api.get_workflow_status_of(analyses[1])
'verified'
```

```
>>> try_transition(analyses[2], "publish", "published")
False
>>> api.get_workflow_status_of(analyses[2])
'verified'
```

But if we publish the Analysis Request, analyses will follow:

```
>>> success = do_action_for(ar, "publish")
>>> api.get_workflow_status_of(ar)
'published'
>>> map(api.get_workflow_status_of, analyses)
['published', 'published', 'published']
```

4.62.3 Check permissions for Published state

In published state, exactly these roles can view results:

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> api.get_workflow_status_of(analysis)
'published'
>>> get_roles_for_permission("senaite.core: View Results", analysis)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'Owner', 'RegulatoryInspector']
```

And no transition can be done from this state:

```
>>> getAllowedTransitions(analysis)
[]
```

4.63 Analysis retract guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisReject

4.63.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getAllowedTransitions
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
```

(continues on next page)

(continued from previous page)

```
...       state = api.get_workflow_status_of(object)
...       return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.63.2 Reject transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Reject one of the analysis:

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> try_transition(analysis, "reject", "rejected")
True
```

The analysis state is now *rejected* while the AR remains in *sample_received*:

```
>>> api.get_workflow_status_of(analysis)
'rejected'
>>> api.get_workflow_status_of(ar)
'sample_received'
```

I cannot submit a result for the rejected analysis:

```
>>> analysis.setResult(12)
>>> try_transition(analysis, "submit", "to_be_verified")
False
>>> api.get_workflow_status_of(analysis)
'rejected'
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Submit results for the rest of the analyses:

```
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> sorted(map(api.get_workflow_status_of, analyses))
['rejected', 'to_be_verified', 'to_be_verified']
```

Reject one of the analyses that are in 'to_be_verified' state:

```
>>> analysis = filter(lambda an: an != analysis, analyses)[0]
>>> try_transition(analysis, "reject", "rejected")
True
>>> api.get_workflow_status_of(analysis)
'rejected'
```

The Analysis Request remains in *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

I cannot 'reject' a verified analysis:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
>>> analysis = filter(lambda an: api.get_workflow_status_of(an) == "to_be_verified",␣
↪analyses)[0]
>>> try_transition(analysis, "verify", "verified")
True
>>> try_transition(analysis, "reject", "rejected")
False
>>> api.get_workflow_status_of(analysis)
'verified'
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.63.3 Rejection of an analysis causes the duplicates to be removed

When the analysis a duplicate comes from is rejected, the duplicate is rejected too, regardless of its state.

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'open'
```

```
>>> ar_ans = ar.getAnalyses(full_objects=True)
>>> an_au = filter(lambda an: an.getKeyword() == 'Au', ar_ans)[0]
>>> an_cu = filter(lambda an: an.getKeyword() == 'Cu', ar_ans)[0]
>>> an_fe = filter(lambda an: an.getKeyword() == 'Fe', ar_ans)[0]
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> du_au = filter(lambda dup: dup.getKeyword() == 'Au', duplicates)[0]
>>> du_cu = filter(lambda dup: dup.getKeyword() == 'Cu', duplicates)[0]
>>> du_fe = filter(lambda dup: dup.getKeyword() == 'Fe', duplicates)[0]
```

When the analysis *Cu* (*to_be_verified*) is rejected, the duplicate is removed:

```
>>> du_cu_uid = api.get_uid(du_cu)
>>> try_transition(an_cu, "reject", "rejected")
True
>>> du_cu in worksheet.getDuplicateAnalyses()
False
>>> api.get_object_by_uid(du_cu_uid, None) is None
True
```

Submit the result for duplicate *Au* and reject *Au* analysis afterwards:

```
>>> du_au_uid = api.get_uid(du_au)
>>> du_au.setResult(12)
>>> try_transition(du_au, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(du_au)
'to_be_verified'
>>> try_transition(an_au, "reject", "rejected")
True
>>> api.get_workflow_status_of(an_au)
'rejected'
>>> du_au in worksheet.getDuplicateAnalyses()
False
>>> api.get_object_by_uid(du_au_uid, None) is None
True
```

Submit and verify the result for duplicate *Fe* and reject *Fe* analysis:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> du_fe_uid = api.get_uid(du_fe)
>>> du_fe.setResult(12)
>>> try_transition(du_fe, "submit", "to_be_verified")
True
>>> try_transition(du_fe, "verify", "verified")
```

(continues on next page)

(continued from previous page)

```
True
>>> try_transition(an_fe, "reject", "rejected")
True
>>> api.get_workflow_status_of(an_fe)
'rejected'
>>> du_fe in worksheet.getDuplicateAnalyses()
False
>>> api.get_object_by_uid(du_fe_uid, None) is None
True
>>> bikasetup.setSelfVerificationEnabled(False)
```

4.63.4 Rejection of analyses with dependents

When rejecting an analysis other analyses depends on (dependents), then the rejection of a dependency causes the auto-rejection of its dependents.

Prepare a calculation that depends on *Cu'and assign it to 'Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> calc_au.setFormula("([Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu = filter(lambda an: an.getKeyword()=="Cu", analyses)[0]
>>> fe = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> au = filter(lambda an: an.getKeyword()=="Au", analyses)[0]
```

When *Fe* is rejected, *Au* analysis follows too:

```
>>> try_transition(fe, "reject", "rejected")
True
>>> api.get_workflow_status_of(fe)
'rejected'
>>> api.get_workflow_status_of(au)
'rejected'
```

While *Cu* analysis remains in *unassigned* state:

```
>>> api.get_workflow_status_of(cu)
'unassigned'
>>> api.get_workflow_status_of(ar)
'sample_received'
```

If we submit *Cu* and reject thereafter:

```
>>> cu.setResult(12)
>>> try_transition(cu, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> try_transition(cu, "reject", "rejected")
True
>>> api.get_workflow_status_of(cu)
'rejected'
```

The Analysis Request rolls-back to *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Reset calculations:

```
>>> Fe.setCalculation(None)
>>> Au.setCalculation(None)
```

4.63.5 Effects of rejection of analysis to Analysis Request

Rejection of analyses have implications in the Analysis Request workflow, cause they will not be considered anymore in regular transitions of Analysis Request that rely on the states of its analyses.

When an Analysis is rejected, the analysis is not considered on submit:

```
>>> ar = new_ar([Cu, Fe])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu = filter(lambda an: an.getKeyword() == 'Cu', analyses)[0]
>>> fe = filter(lambda an: an.getKeyword() == 'Fe', analyses)[0]
>>> success = do_action_for(cu, "reject")
>>> api.get_workflow_status_of(cu)
'rejected'
>>> fe.setResult(12)
>>> success = do_action_for(fe, "submit")
>>> api.get_workflow_status_of(fe)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Neither considered on verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> success = do_action_for(fe, "verify")
>>> api.get_workflow_status_of(fe)
'verified'
>>> api.get_workflow_status_of(ar)
'verified'
```

Neither considered on publish:

```
>>> success = do_action_for(ar, "publish")
>>> api.get_workflow_status_of(ar)
'published'
```

Reset self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
```

4.63.6 Rejection of retests

Create an Analysis Request, receive and submit all results:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> success = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
>>> for analysis in analyses:
...     analysis.setResult(12)
...     success = do_action_for(analysis, "submit")
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Retract one of the analyses:

```
>>> analysis = analyses[0]
>>> success = do_action_for(analysis, "retract")
>>> api.get_workflow_status_of(analysis)
'retracted'
```

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Reject the retest:

```
>>> retest = analysis.getRetest()
>>> success = do_action_for(retest, "reject")
>>> api.get_workflow_status_of(retest)
'rejected'
```

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Verify remaining analyses:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> success = do_action_for(analyses[1], "verify")
>>> success = do_action_for(analyses[2], "verify")
>>> bikasetup.setSelfVerificationEnabled(False)
```

```
>>> api.get_workflow_status_of(ar)
'verified'
```

4.63.7 Check permissions for Reject transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> allowed_roles = ['LabManager', 'Manager']
```

(continues on next page)

(continued from previous page)

```
>>> non_allowed_roles = ['Analyst', 'Authenticated', 'LabClerk', 'Owner',
...                      'RegulatoryInspector', 'Sampler', 'Verifier']
```

In unassigned state

In *unassigned* state, exactly these roles can reject:

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
>>> get_roles_for_permission("Reject", analysis)
['LabManager', 'Manager']
```

Current user can reject because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "reject")
True
```

Also if the user has the role *Manager*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(analysis, "reject")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, non_allowed_roles)
>>> isTransitionAllowed(analysis, "reject")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

In assigned state

In *assigned* state, exactly these roles can reject:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalysis(analysis)
>>> api.get_workflow_status_of(analysis)
'assigned'
>>> get_roles_for_permission("Reject", analysis)
['LabManager', 'Manager']
>>> isTransitionAllowed(analysis, "reject")
True
```

Current user can reject because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "reject")
True
```

Also if the user has the role *Manager*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(analysis, "reject")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, non_allowed_roles)
>>> isTransitionAllowed(analysis, "reject")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

In to_be_verified state

In *to_be_verified* state, exactly these roles can reject:

```
>>> analysis.setResult(13)
>>> success = do_action_for(analysis, "submit")
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
>>> get_roles_for_permission("Reject", analysis)
['LabManager', 'Manager']
>>> isTransitionAllowed(analysis, "reject")
True
```

Current user can reject because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "reject")
True
```

Also if the user has the role *Manager*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(analysis, "reject")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, non_allowed_roles)
>>> isTransitionAllowed(analysis, "reject")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

In retracted state

In *retracted* state, the analysis cannot be rejected:

```
>>> success = do_action_for(analysis, "retract")
>>> api.get_workflow_status_of(analysis)
'retracted'
>>> get_roles_for_permission("Reject", analysis)
[]
>>> isTransitionAllowed(analysis, "reject")
False
```

In verified state

In *verified* state, the analysis cannot be rejected:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> analysis = analysis.getRetest()
>>> analysis.setResult(12)
>>> success = do_action_for(analysis, "submit")
>>> success = do_action_for(analysis, "verify")
>>> api.get_workflow_status_of(analysis)
'verified'
>>> get_roles_for_permission("Reject", analysis)
[]
>>> isTransitionAllowed(analysis, "reject")
False
```

In published state

In *published* state, the analysis cannot be rejected:

```
>>> do_action_for(ar, "publish")
(True, '')
>>> api.get_workflow_status_of(analysis)
'published'
>>> get_roles_for_permission("Reject", analysis)
[]
>>> isTransitionAllowed(analysis, "reject")
False
```

In cancelled state

In *cancelled* state, the analysis cannot be rejected:

```
>>> ar = new_ar([Cu])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> success = do_action_for(ar, "cancel")
>>> api.get_workflow_status_of(analysis)
'cancelled'
>>> get_roles_for_permission("Reject", analysis)
[]
>>> isTransitionAllowed(analysis, "reject")
False
```

Disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
```

4.63.8 Check permissions for Rejected state

In rejected state, exactly these roles can view results:

```
>>> ar = new_ar([Cu])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> success = do_action_for(analysis, "reject")
>>> api.get_workflow_status_of(analysis)
'rejected'
>>> get_roles_for_permission("senaite.core: View Results", analysis)
['LabManager', 'Manager', 'RegulatoryInspector']
```

And no transition can be done from this state:

```
>>> getAllowedTransitions(analysis)
[]
```

4.64 Analysis Request cancel guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisRequestCancel

4.64.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.64.2 Cancel transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> api.get_workflow_status_of(ar)
'sample_due'
```

Cancel the Analysis Request:

```
>>> transitioned = do_action_for(ar, "cancel")
>>> api.get_workflow_status_of(ar)
'cancelled'
```

And all analyses the Analysis Request contains are cancelled too:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['cancelled', 'cancelled', 'cancelled']
```

At this point, only "reinstate" transition is possible:

```
>>> getAllowedTransitions(ar)
['reinstate']
```

When the Analysis Request is reinstated, it status becomes the previous before the cancellation took place:

```
>>> transitioned = do_action_for(ar, "reinstate")
>>> api.get_workflow_status_of(ar)
'sample_due'
```

And the analyses are reinstated too:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

Receive the Analysis Request:

```
>>> transitioned = do_action_for(ar, "receive")
>>> api.get_workflow_status_of(ar)
'sample_received'
```

And we can cancel again:

```
>>> transitioned = do_action_for(ar, "cancel")
>>> api.get_workflow_status_of(ar)
'cancelled'
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['cancelled', 'cancelled', 'cancelled']
```

And reinstate:

```
>>> transitioned = do_action_for(ar, "reinstate")
>>> api.get_workflow_status_of(ar)
'sample_received'
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

Thus, the Analysis Request can be cancelled again:

```
>>> isTransitionAllowed(ar, "cancel")
True
```

But if we assign an analysis to a worksheet, the cancellation is no longer possible:

```
>>> analysis = analyses[0]
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.addAnalysis(analysis)
>>> api.get_workflow_status_of(analysis)
'assigned'
>>> isTransitionAllowed(ar, "cancel")
False
```

But if we unassign the analysis, the transition is possible again:

```
>>> worksheet.removeAnalysis(analysis)
>>> api.get_workflow_status_of(analysis)
'unassigned'
>>> isTransitionAllowed(ar, "cancel")
True
```

If a result for any given analysis is submitted, the Analysis Request cannot be transitioned to "cancelled" status:

```
>>> analysis.setResult(12)
>>> transitioned = do_action_for(analysis, "submit")
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
>>> isTransitionAllowed(ar, "cancel")
False
```

4.65 Analysis Request invalidate guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisRequestInvalidate

4.65.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.interfaces import IAnalysisRequestRetest
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime(),
...         'SampleType': sampletype.UID()}
```

(continues on next page)

(continued from previous page)

```
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.65.2 Invalidate transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> ar
<AnalysisRequest at /plone/clients/client-1/W-0001>
```

Analysis Request cannot be invalidated when the status is *sample_due*:

```
>>> api.get_workflow_status_of(ar)
'sample_due'
```

```
>>> isTransitionAllowed(ar, "invalidate")
False
```

Analysis Request cannot be invalidated when the status is *sample_received*:

```
>>> success = do_action_for(ar, "receive")
>>> api.get_workflow_status_of(ar)
'sample_received'
```

```
>>> isTransitionAllowed(ar, "invalidate")
False
```

Submit all analyses:

```
>>> for analysis in ar.getAnalyses(full_objects=True):
...     analysis.setResult(12)
...     success = do_action_for(analysis, "submit")
```

Analysis Request cannot be invalidated when status is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> isTransitionAllowed(ar, "invalidate")
False
```

Verify all analyses:

```
>>> setup.setSelfVerificationEnabled(True)
>>> for analysis in ar.getAnalyses(full_objects=True):
...     success = do_action_for(analysis, "verify")
>>> setup.setSelfVerificationEnabled(False)
```

Analysis Request can be invalidated if *verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
```

```
>>> isTransitionAllowed(ar, "invalidate")
True
```

When invalidated, a retest is created:

```
>>> success = do_action_for(ar, "invalidate")
>>> api.get_workflow_status_of(ar)
'invalid'
```

```
>>> retest = ar.getRetest()
>>> retest
<AnalysisRequest at /plone/clients/client-1/W-0001-R01>
```

And the retest provides *IAnalysisRequestRetest* interface:

```
>>> IAnalysisRequestRetest.providedBy(retest)
True
```

From the retest, I can go back to the invalidated Analysis Request:

```
>>> retest.getInvalidated()
<AnalysisRequest at /plone/clients/client-1/W-0001>
```

4.65.3 Check permissions for Invalidate transition

Create an Analysis Request, receive, submit results and verify them:

```
>>> ar = new_ar([Cu])
>>> success = do_action_for(ar, "receive")
>>> setup.setSelfVerificationEnabled(True)
>>> for analysis in ar.getAnalyses(full_objects=True):
...     analysis.setResult(12)
...     submitted = do_action_for(analysis, "submit")
...     verified = do_action_for(analysis, "verify")
>>> setup.setSelfVerificationEnabled(False)
>>> api.get_workflow_status_of(ar)
'verified'
```

Exactly these roles can invalidate:

```
>>> get_roles_for_permission("senaite.core: Transition: Invalidate", ar)
['LabManager', 'Manager']
```

Current user can assign because has the *LabManager* role:

```
>>> isTransitionAllowed(ar, "invalidate")
True
```

User with other roles cannot:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk', 'Owner'])
>>> isTransitionAllowed(analysis, "invalidate")
False
```

Reset settings:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.66 Analysis Request sample guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisRequestSample

4.66.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone import api as ploneapi
>>> from plone.app.testing import setRoles
```

(continues on next page)

(continued from previous page)

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def new_ar(services, ar_template=None):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'SampleType': sampletype.UID(),
...         'Template': ar_template,
...     }
...     date_key = "DateSampled"
...     if ar_template and ar_template.getSamplingRequired():
...         date_key = "SamplingDate"
...     elif bikasetup.getSamplingWorkflowEnabled():
...         date_key = "SamplingDate"
...     values[date_key] = timestamp()
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

```
>>> def roles_for_transition_check(transition_id, roles, object):
...     granted = list()
...     ungranted = list()
...     for role in roles:
...         setRoles(portal, TEST_USER_ID, [role])
...         if isTransitionAllowed(object, transition_id):
...             granted.append(role)
...         else:
...             ungranted.append(role)
...     setRoles(portal, TEST_USER_ID, ['LabManager',])
...     return granted, ungranted
```

```
>>> def are_roles_for_transition_granted(transition_id, roles, object):
...     gr, ungr = roles_for_transition_check(transition_id, roles, object)
...     return len(ungr) == 0 and len(gr) > 0
```

```
>>> def are_roles_for_transition_ungranted(transition_id, roles, object):
...     gr, ungr = roles_for_transition_check(transition_id, roles, object)
...     return len(gr) == 0 and len(ungr) > 0
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = timestamp()
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> ar_template = api.create(bikasetup.bika_artemplates, "ARTemplate", title="Test␣
↪Template", SampleType=sampletype)
>>> sampler_user = ploneapi.user.create(email="sampler1@example.com", username=
↪"sampler1", password="secret", properties=dict(fullname="Sampler 1"))
>>> setRoles(portal, "sampler1", ['Authenticated', 'Member', 'Sampler'])
```

4.66.2 Sample transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
```

By default, the Analysis Request transitions to “sample_due” status:

```
>>> api.get_workflow_status_of(ar)
'sample_due'
```

And from this status, the transition “sample” is not possible:

```
>>> isTransitionAllowed(ar, "sample")
False
```

If the value for setup setting “SamplingWorkflowEnabled” is True, the status of the Analysis Request once created is “to_be_sampled”:

```
>>> bikasetup.setSamplingWorkflowEnabled(True)
>>> ar = new_ar([Cu])
>>> api.get_workflow_status_of(ar)
'to_be_sampled'
```

But the transition is still not possible:

```
>>> isTransitionAllowed(ar, "sample")
False
```

Because we haven't set neither a Sampler nor the date the sample was collected:

```
>>> date_sampled = timestamp()
>>> ar.setDateSampled(date_sampled)
>>> isTransitionAllowed(ar, "sample")
False
>>> ar.setSampler(sampler_user.id)
>>> isTransitionAllowed(ar, "sample")
True
```

When "sample" transition is performed, the status becomes "sample_due":

```
>>> success = do_action_for(ar, "sample")
>>> api.get_workflow_status_of(ar)
'sample_due'
```

And the values for DateSampled and Sampler are kept:

```
>>> ar.getSampler() == sampler_user.id
True
>>> ar.getDateSampled().strftime("%Y-%m-%d") == date_sampled
True
```

4.66.3 Check permissions for sample transition

Declare the roles allowed and not allowed to perform the "sample" transition:

```
>>> all_roles = portal.acl_users.portal_role_manager.validRoles()
>>> allowed = ["LabManager", "Manager", "Sampler", "SamplingCoordinator"]
>>> not_allowed = filter(lambda role: role not in allowed, all_roles)
```

Create an Analysis Request by using a template with Sampling workflow enabled:

```
>>> bikasetup.setSamplingWorkflowEnabled(False)
>>> ar_template.setSamplingRequired(True)
>>> ar = new_ar([Cu], ar_template)
>>> ar.setDateSampled(timestamp())
>>> ar.setSampler(sampler_user.id)
```

Exactly these roles can Sample:

```
>>> get_roles_for_permission("senaite.core: Transition: Sample Sample", ar)
['LabManager', 'Manager', 'Sampler', 'SamplingCoordinator']
```

Current user can sample because has the *LabManager* role:

```
>>> isTransitionAllowed(ar, "sample")
True
```

The user can sample if has any of the granted roles:

```
>>> are_roles_for_transition_granted("sample", allowed, ar)
True
```

But not if the user has the rest of the roles:

```
>>> are_roles_for_transition_ungranted("sample", not_allowed, ar)
True
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.67 Analysis Request to_be_sampled guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisRequestToBeSampled

4.67.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def new_ar(services, ar_template=None):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
```

(continues on next page)

(continued from previous page)

```
...         'SampleType': sampletype.UID(),
...         'Template': ar_template,
...     }
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = timestamp()
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> ar_template = api.create(bikasetup.bika_artemplates, "ARTemplate", title="Test␣
↪Template", SampleType=sampletype)
```

4.67.2 To_be_sampled transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
```

By default, the Analysis Request transitions to "sample_due" status:

```
>>> api.get_workflow_status_of(ar)
'sample_due'
```

But if the setup setting "SamplingWorkflowEnabled" is set to True, the status of the Analysis Request once created is "to_be_sampled":

```
>>> bikasetup.setSamplingWorkflowEnabled(True)
>>> ar = new_ar([Cu])
>>> api.get_workflow_status_of(ar)
'to_be_sampled'
```

If we use a template with "SamplingRequired" setting set to False, the status of the Analysis Request once created is "sample_due", regardless of the setting from setup:

```
>>> ar_template.setSamplingRequired(False)
>>> ar = new_ar([Cu], ar_template)
>>> api.get_workflow_status_of(ar)
'sample_due'
```

And same the other way round:

```
>>> bikasetup.setSamplingWorkflowEnabled(False)
>>> ar_template.setSamplingRequired(True)
>>> ar = new_ar([Cu], ar_template)
>>> api.get_workflow_status_of(ar)
'to_be_sampled'
```

4.68 Analysis retract guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisRetract

4.68.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
```

(continues on next page)

(continued from previous page)

```
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.68.2 Retract transition and guard basic constraints

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

Retract one of the analyses:

```
>>> analysis = analyses[0]
>>> try_transition(analysis, "retract", "retracted")
True
>>> api.get_workflow_status_of(analysis)
'retracted'
```

And one new additional analysis has been added in *unassigned* state:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> sorted(map(api.get_workflow_status_of, analyses))
['retracted', 'to_be_verified', 'to_be_verified', 'unassigned']
```

And the Analysis Request has been transitioned to *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

The new analysis is a copy of retracted one:

```
>>> retest = filter(lambda an: api.get_workflow_status_of(an) == "unassigned",␣
↪analyses)[0]
>>> analysis.getRetest() == retest
True
>>> retest.getRetestOf() == analysis
True
>>> retest.getKeyword() == analysis.getKeyword()
True
```

But it does not keep the result:

```
>>> not retest.getResult()
True
```

And Result capture date is None:

```
>>> not retest.getResultCaptureDate()
True
```

If I submit the result for the new analysis:

```
>>> retest.setResult(analysis.getResult())
>>> try_transition(retest, "submit", "to_be_verified")
True
```

The status of both the analysis and the Analysis Request is "to_be_verified":

```
>>> api.get_workflow_status_of(retest)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

And I can even retract the retest:

```
>>> try_transition(retest, "retract", "retracted")
True
>>> api.get_workflow_status_of(retest)
'retracted'
```

And one new additional analysis has been added in *unassigned* state:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> sorted(map(api.get_workflow_status_of, analyses))
['retracted', 'retracted', 'to_be_verified', 'to_be_verified', 'unassigned']
```

And again, the Analysis Request has been transitioned to *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

4.68.3 Auto-rollback of Worksheet on analysis retraction

When retracting an analysis from a Worksheet that is in "to_be_verified" state causes the rollback of the worksheet to "open" state.

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a new Worksheet, assign all analyses and submit:

```
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
>>> submit_analyses(ar)
```

The state for both the Analysis Request and Worksheet is "to_be_verified":

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

Retract one analysis:

```
>>> analysis = ws.getAnalyses()[0]
>>> try_transition(analysis, "retract", "retracted")
True
```

A rollback of the state of Analysis Request and Worksheet takes place:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
>>> api.get_workflow_status_of(ws)
'open'
```

And both contain an additional analysis:

```
>>> len(ar.getAnalyses())
4
>>> len(ws.getAnalyses())
4
```

The state of this additional analysis, the retest, is "assigned":

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> retest = filter(lambda an: api.get_workflow_status_of(an) == "assigned",␣
↪analyses)[0]
>>> retest.getKeyword() == analysis.getKeyword()
True
>>> retest in ws.getAnalyses()
True
```

4.68.4 Retraction of results for analyses with dependents

When retracting an analysis other analyses depends on (dependents), then the retraction of a dependency causes the auto-retraction of its dependents. Dependencies are retracted too.

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> calc_au.setFormula("([Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu_analysis = filter(lambda an: an.getKeyword()=="Cu", analyses)[0]
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> au_analysis = filter(lambda an: an.getKeyword()=="Au", analyses)[0]
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> cu_analysis.setResult(20)
>>> fe_analysis.setResult(12)
>>> au_analysis.setResult(10)
```

Submit *Au* analysis and the rest will follow:

```
>>> try_transition(au_analysis, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(au_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(fe_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(cu_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

If I retract *Fe*, *Au* analysis is retracted automatically too:

```
>>> try_transition(fe_analysis, "retract", "retracted")
True
>>> api.get_workflow_status_of(fe_analysis)
'retracted'
>>> api.get_workflow_status_of(au_analysis)
'retracted'
```

As well as *Cu* analysis (a dependency of *Fe*):

```
>>> api.get_workflow_status_of(cu_analysis)
'retracted'
```

Hence, three new analyses are generated in accordance:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> len(analyses)
6
>>> au_analyses = filter(lambda an: an.getKeyword()=="Au", analyses)
>>> sorted(map(api.get_workflow_status_of, au_analyses))
['retracted', 'unassigned']
>>> fe_analyses = filter(lambda an: an.getKeyword()=="Fe", analyses)
>>> sorted(map(api.get_workflow_status_of, fe_analyses))
['retracted', 'unassigned']
>>> cu_analyses = filter(lambda an: an.getKeyword()=="Cu", analyses)
>>> sorted(map(api.get_workflow_status_of, cu_analyses))
['retracted', 'unassigned']
```

And the current state of the Analysis Request is *sample_received* now:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

4.69 Analysis submission guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisSubmit

4.69.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager', 'Sampler'])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
```

(continues on next page)

(continued from previous page)

```
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.69.2 Basic constraints for Analysis submission

Create an Analysis Request:

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'DateSampled': date_now,
...           'SampleType': sampletype.UID()}
>>> service_uids = map(api.get_uid, [Cu])
>>> ar = create_analysisrequest(client, request, values, service_uids)
```

Cannot submit if the Analysis Request has not been yet received:

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> analysis.setResult(12)
>>> isTransitionAllowed(analysis, "submit")
False
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
>>> api.get_workflow_status_of(analysis)
'registered'
```

But if I receive the Analysis Request:

```
>>> transitioned = do_action_for(ar, "receive")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_received'
```

I can then submit the analysis:

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

And I cannot resubmit the analysis:

```
>>> isTransitionAllowed(analysis, "submit")
False
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

4.69.3 Basic constraints for "field" Analysis submission

Set analysis *Cu* with Point of Capture "field":

```
>>> Cu.setPointOfCapture("field")
>>> Cu.getPointOfCapture()
'field'
```

And activate sampling workflow:

```
>>> bikasetup.setSamplingWorkflowEnabled(True)
>>> bikasetup.getSamplingWorkflowEnabled()
True
```

Create an Analysis Request:

```
>>> values = {'Client': client.UID(),
...           'Contact': contact.UID(),
...           'SampleType': sampletype.UID()}
>>> service_uids = map(api.get_uid, [Cu, Fe])
>>> ar = create_analysisrequest(client, request, values, service_uids)
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu = filter(lambda an: an.getKeyword() == "Cu", analyses)[0]
>>> fe = filter(lambda an: an.getKeyword() == "Fe", analyses)[0]
```

Cannot submit *Cu*, because the Analysis Request has not been yet sampled:

```
>>> cu.setResult(12)
>>> isTransitionAllowed(cu, "submit")
False
>>> api.get_workflow_status_of(ar)
'to_be_sampled'
```

I cannot submit *Fe* neither, cause the Analysis Request has not been received:

```
>>> fe.setResult(12)
>>> isTransitionAllowed(fe, "submit")
False
```

If I sample the Analysis Request:

```
>>> ar.setDateSampled(timestamp())
>>> ar.setSampler(TEST_USER_ID)
>>> transitioned = do_action_for(ar, "sample")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_due'
```

Then I can submit *Cu*:

```
>>> transitioned = do_action_for(cu, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(cu)
'to_be_verified'
```

But cannot submit *Fe*:

```
>>> cu.setResult(12)
>>> isTransitionAllowed(fe, "submit")
False
```

Unless I receive the Analysis Request:

```
>>> transitioned = do_action_for(ar, "receive")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(ar)
'sample_received'
>>> transitioned = do_action_for(fe, "submit")
>>> transitioned[0]
True
>>> api.get_workflow_status_of(fe)
'to_be_verified'
```

And I cannot resubmit again:

```
>>> isTransitionAllowed(cu, "submit")
False
>>> isTransitionAllowed(fe, "submit")
False
```

Deactivate the workflow sampling and rest *Cu* as a lab analysis:

```
>>> Cu.setPointOfCapture("lab")
>>> bikasetup.setSamplingWorkflowEnabled(False)
```

4.69.4 Auto submission of Analysis Requests when all its analyses are submitted

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Set results for some of the analyses only:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> analyses[0].setResult('12')
>>> analyses[1].setResult('12')
```

We've set some results, but all analyses are still in *unassigned*:

```
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

Transition some of them:

```
>>> transitioned = do_action_for(analyses[0], "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analyses[0])
'to_be_verified'
```

```
>>> transitioned = do_action_for(analyses[1], "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analyses[1])
'to_be_verified'
```

The Analysis Request status is still in *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

If we try to transition the remaining analysis w/o result, nothing happens:

```
>>> transitioned = do_action_for(analyses[2], "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analyses[2])
'unassigned'
```

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

Even if we try with an empty or None result:

```
>>> analyses[2].setResult('')
>>> transitioned = do_action_for(analyses[2], "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analyses[2])
'unassigned'
```

```
>>> analyses[2].setResult(None)
>>> transitioned = do_action_for(analyses[2], "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analyses[2])
'unassigned'
```

But will work if we try with a result of 0:

```
>>> analyses[2].setResult(0)
>>> transitioned = do_action_for(analyses[2], "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analyses[2])
'to_be_verified'
```

And the AR will follow:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

And we cannot re-submit analyses that have been already submitted:

```
>>> transitioned = do_action_for(analyses[2], "submit")
>>> transitioned[0]
False
```

4.69.5 Auto submission of a Worksheets when all its analyses are submitted

The same behavior as for Analysis Requests applies to the worksheet when all its analyses are submitted.

Create two Analysis Requests:

```
>>> ar0 = new_ar([Cu, Fe, Au])
>>> ar1 = new_ar([Cu, Fe])
```

Create a worksheet:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
```

And assign all the analyses from the Analysis Requests created before, except *Au* from the first Analysis Request:

```
>>> analyses_ar0 = ar0.getAnalyses(full_objects=True)
>>> analyses_ar1 = ar1.getAnalyses(full_objects=True)
>>> analyses = filter(lambda an: an.getKeyword() != 'Au', analyses_ar0)
>>> analyses += analyses_ar1
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Set results and submit all analyses from the worksheet except one:

```
>>> ws_analyses = worksheet.getAnalyses()
>>> analysis_1 = analyses[0]
>>> analysis_2 = analyses[1]
>>> analysis_3 = analyses[2]
>>> analysis_4 = analyses[3]
```

```
>>> analysis_2.setResult('5')
>>> transitioned = do_action_for(analysis_2, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis_2)
'to_be_verified'
```

```
>>> analysis_3.setResult('6')
>>> transitioned = do_action_for(analysis_3, "submit")
```

(continues on next page)

(continued from previous page)

```
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis_3)
'to_be_verified'
```

```
>>> analysis_4.setResult('7')
>>> transitioned = do_action_for(analysis_4, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis_4)
'to_be_verified'
```

The Analysis Request number 1 has been automatically transitioned because all the contained analyses have been submitted:

```
>>> api.get_workflow_status_of(ar1)
'to_be_verified'
```

While Analysis Request number 0 has not been transitioned because still have two analyses with results pending:

```
>>> api.get_workflow_status_of(ar0)
'sample_received'
```

And same with worksheet, cause there is one result pending:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

If we set a result for the pending analysis:

```
>>> analysis_1.setResult('9')
>>> transitioned = do_action_for(analysis_1, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis_1)
'to_be_verified'
```

The worksheet will follow:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

But the Analysis Request number 0 will remain *sample_received*:

```
>>> api.get_workflow_status_of(ar0)
'sample_received'
```

Unless we submit a result for *Au* analysis:

```
>>> au_an = filter(lambda an: an.getKeyword() == 'Au', analyses_ar0)[0]
>>> au_an.setResult('10')
```

(continues on next page)

(continued from previous page)

```
>>> transitioned = do_action_for(au_an, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(au_an)
'to_be_verified'
```

```
>>> api.get_workflow_status_of(ar0)
'to_be_verified'
```

4.69.6 Submission of results for analyses with interim fields set

For an analysis to be submitted successfully, it must have a result set, but if the analysis have interim fields, they are mandatory too:

```
>>> Au.setInterimFields([
...     {"keyword": "interim_1", "title": "Interim 1",},
...     {"keyword": "interim_2", "title": "Interim 2",}])
```

Create an Analysis Request:

```
>>> ar = new_ar([Au])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
```

Cannot submit if no result is set:

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

But even if we set a result, we cannot submit because interims are missing:

```
>>> analysis.setResult(12)
>>> analysis.getResult()
'12'
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

So, if the analysis has interims defined, all them are required too:

```
>>> analysis.setInterimValue("interim_1", 15)
>>> analysis.getInterimValue("interim_1")
'15'
```

```
>>> analysis.getInterimValue("interim_2")
''
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

Even if we set a non-valid (None, empty value) to an interim:

```
>>> analysis.setInterimValue("interim_2", None)
>>> analysis.getInterimValue("interim_2")
''
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

```
>>> analysis.setInterimValue("interim_2", '')
>>> analysis.getInterimValue("interim_2")
''
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

But it will work if the value is 0:

```
>>> analysis.setInterimValue("interim_2", 0)
>>> analysis.getInterimValue("interim_2")
'0'
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

And the Analysis Request follow:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Might happen the other way round. We set interims but not a result:

```
>>> ar = new_ar([Au])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> analysis.setInterimValue("interim_1", 10)
>>> analysis.setInterimValue("interim_2", 20)
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

Still, the result is required:

```
>>> analysis.setResult(12)
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

And again, the Analysis Request will follow:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

4.69.7 Submission of results for analyses with interim calculation

If an analysis have a calculation assigned, the result will be calculated automatically based on the calculation. If the calculation have interims set, only those that do not have a default value set will be required.

Prepare the calculation and set the calculation to analysis *Au*:

```
>>> Au.setInterimFields([])
>>> calc = api.create(bikasetup.bika_calculations, 'Calculation', title='Test␣
↪Calculation')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1', 'value': 10}
>>> interim_2 = {'keyword': 'IT2', 'title': 'Interim 2', 'value': 2}
>>> interim_3 = {'keyword': 'IT3', 'title': 'Interim 3', 'value': ''}
>>> interim_4 = {'keyword': 'IT4', 'title': 'Interim 4', 'value': None}
>>> interim_5 = {'keyword': 'IT5', 'title': 'Interim 5'}
>>> interims = [interim_1, interim_2, interim_3, interim_4, interim_5]
>>> calc.setInterimFields(interims)
>>> calc.setFormula("[IT1]+[IT2]+[IT3]+[IT4]+[IT5]")
>>> Au.setCalculation(calc)
```

Create an Analysis Request:

```
>>> ar = new_ar([Au])
>>> analysis = ar.getAnalyses(full_objects=True)[0]
```

Cannot submit if no result is set:

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> analysis.setResult("12")
```

Set a value for interim IT5:

```
>>> analysis.setInterimValue("IT5", 5)
```

Cannot transition because IT3 and IT4 have None/empty values as default:

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

Let's set a value for those interims:

```
>>> analysis.setInterimValue("IT3", 3)
```

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(analysis)
'unassigned'
```

```
>>> analysis.setInterimValue("IT4", 4)
```

Since interims IT1 and IT2 have default values set, the analysis will submit:

```
>>> transitioned = do_action_for(analysis, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(analysis)
'to_be_verified'
```

4.69.8 Submission of results for analyses with dependencies

If an analysis is associated to a calculation that uses the result of other analyses (dependents), then the analysis cannot be submitted unless all its dependents were previously submitted.

Reset the interim fields for analysis *Au*:

```
>>> Au.setInterimFields([])
```

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1'}
>>> calc_au.setInterimFields([interim_1])
>>> calc_au.setFormula("([IT1]+[Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu_analysis = filter(lambda an: an.getKeyword()=="Cu", analyses)[0]
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> au_analysis = filter(lambda an: an.getKeyword()=="Au", analyses)[0]
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> fe_analysis.setResult(12)
>>> au_analysis.setResult(10)
```

Cannot submit *Fe*, because there is no result for *Cu* yet:

```
>>> transitioned = do_action_for(fe_analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(fe_analysis)
'unassigned'
```

And we cannot submit *Au*, because *Cu*, a dependency of *Fe*, has no result:

```
>>> transitioned = do_action_for(au_analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(au_analysis)
'unassigned'
```

Set a result for *Cu* and submit:

```
>>> cu_analysis.setResult(12)
>>> transitioned = do_action_for(cu_analysis, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(cu_analysis)
'to_be_verified'
```

But *Fe* won't follow, cause only dependencies follow, but not dependents:

```
>>> api.get_workflow_status_of(fe_analysis)
'unassigned'
```

If we try to submit *Au*, the submission will not take place:

```
>>> transitioned = do_action_for(au_analysis, "submit")
>>> transitioned[0]
False
```

```
>>> api.get_workflow_status_of(au_analysis)
'unassigned'
```

Because of the missing interim. Set the interim for *Au*:

```
>>> au_analysis.setInterimValue("IT1", 4)
```

And now we are able to submit *Au*:

```
>>> transitioned = do_action_for(au_analysis, "submit")
>>> transitioned[0]
True
```

```
>>> api.get_workflow_status_of(au_analysis)
'to_be_verified'
```

And since *Fe* is a dependency of *Au*, *Fe* will be automatically transitioned:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

As well as the Analysis Request:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

4.69.9 Check permissions for Submit transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
```

The status of the Analysis Request is *sample_received*:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

And the status of the Analysis is *unassigned*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned']
```

Set a result:

```
>>> analysis = analyses[0]
>>> analysis.setResult(23)
```

Exactly these roles can submit:

```
>>> get_roles_for_permission("senaite.core: Edit Results", analysis)
['Analyst', 'LabManager', 'Manager']
```

```
>>> get_roles_for_permission("senaite.core: Edit Field Results", analysis)
['LabManager', 'Manager', 'Sampler']
```

And these roles can view results:

```
>>> get_roles_for_permission("senaite.core: View Results", analysis)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'RegulatoryInspector']
```

Current user can submit because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "submit")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Authenticated', 'LabClerk', 'RegulatoryInspector
↪'])
>>> isTransitionAllowed(analysis, "submit")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(analysis, "submit")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(analysis, "submit")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.70 Analysis unassign guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisUnassign

4.70.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.catalog import CATALOG_ANALYSIS_REQUEST_LISTING
>>> from bika.lims.catalog import CATALOG_WORKSHEET_LISTING
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.70.2 Unassign transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
```

The status of the analyses is *unassigned*:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['unassigned', 'unassigned', 'unassigned']
```

And the Analysis Request' assigned state index is 'unassigned':

```
>>> query = dict(assigned_state='unassigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
1
>>> query = dict(assigned_state='assigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
0
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

(continues on next page)

(continued from previous page)

```
>>> sorted((map(lambda an: an.getKeyword(), worksheet.getAnalyses())))
['Au', 'Cu', 'Fe']
>>> map(api.get_workflow_status_of, analyses)
['assigned', 'assigned', 'assigned']
```

The Analysis Request' assigned state indexer is 'assigned':

```
>>> query = dict(assigned_state='unassigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
0
>>> query = dict(assigned_state='assigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
1
```

The worksheet has now 3 analyses assigned:

```
>>> worksheet.getNumberOfRegularAnalyses()
3
>>> worksheet.getNumberOfQCAnalyses()
0
```

And metadata gets updated accordingly:

```
>>> query = dict(UID=api.get_uid(worksheet))
>>> ws_brain = api.search(query, CATALOG_WORKSHEET_LISTING)[0]
>>> ws_brain.getNumberOfRegularAnalyses
3
>>> ws_brain.getNumberOfQCAnalyses
0
>>> an_uids = sorted(map(api.get_uid, worksheet.getAnalyses()))
>>> sorted(ws_brain.getAnalysesUIDs) == an_uids
True
```

When we unassign the *Cu* analysis, the workseet gets updated:

```
>>> cu = filter(lambda an: an.getKeyword() == 'Cu', worksheet.getAnalyses())[0]
>>> succeed = do_action_for(cu, "unassign")
>>> api.get_workflow_status_of(cu)
'unassigned'
>>> cu in worksheet.getAnalyses()
False
>>> worksheet.getNumberOfRegularAnalyses()
2
>>> ws_brain = api.search(query, CATALOG_WORKSHEET_LISTING)[0]
>>> ws_brain.getNumberOfRegularAnalyses
2
>>> api.get_uid(cu) in ws_brain.getAnalysesUIDs
False
>>> len(ws_brain.getAnalysesUIDs)
2
```

And the Analysis Request' assigned state index is updated as well:

```
>>> query = dict(assigned_state='unassigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
1
```

(continues on next page)

(continued from previous page)

```
>>> query = dict(assigned_state='assigned', UID=api.get_uid(ar))
>>> len(api.search(query, CATALOG_ANALYSIS_REQUEST_LISTING))
0
```

4.70.3 Unassign of an analysis causes the duplicates to be removed

When the analysis a duplicate comes from is unassigned, the duplicate is removed from the worksheet too.

Create a Worksheet and add the analyses:

```
>>> ar = new_ar([Cu])
>>> transitioned = do_action_for(ar, "receive")
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> api.get_workflow_status_of(worksheet)
'open'
>>> cu = ar.getAnalyses(full_objects=True)[0]
>>> dcu = worksheet.getDuplicateAnalyses()[0]
```

When the analysis *Cu* is unassigned, the duplicate is removed:

```
>>> dcu_uid = api.get_uid(dcu)
>>> try_transition(cu, "unassign", "unassigned")
True
>>> api.get_workflow_status_of(cu)
'unassigned'
>>> dcu_uid in worksheet.getDuplicateAnalyses()
False
>>> api.get_object_by_uid(dcu_uid, None) is None
True
```

4.71 Analysis verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowAnalysisVerify

4.71.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
```

(continues on next page)

(continued from previous page)

```
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.71.2 Verify transition and guard basic constraints

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

I cannot verify the analyses because I am the same user who submitted them:

```
>>> try_transition(analyses[0], "verify", "verified")
False
>>> api.get_workflow_status_of(analyses[0])
'to_be_verified'
```

```
>>> try_transition(analyses[1], "verify", "verified")
False
>>> api.get_workflow_status_of(analyses[1])
'to_be_verified'
```

```
>>> try_transition(analyses[2], "verify", "verified")
False
>>> api.get_workflow_status_of(analyses[2])
'to_be_verified'
```

And I cannot verify Analysis Request neither, because the Analysis Request can only be verified once all the analyses it contains are verified (and this is done automatically):

```
>>> try_transition(ar, "verify", "verified")
False
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

But if enable the self verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Then, I will be able to verify my own results:

```
>>> try_transition(analyses[0], "verify", "verified")
True
>>> try_transition(analyses[1], "verify", "verified")
True
```

But the Analysis Request will remain in *to_be_verified* state:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

Until we verify all the analyses it contains:

```
>>> try_transition(analyses[2], "verify", "verified")
True
>>> api.get_workflow_status_of(ar)
'verified'
```

And we cannot re-verify an analysis that has been verified already:

```
>>> try_transition(analyses[2], "verify", "verified")
False
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.71.3 Auto verification of Worksheets when all its analyses are verified

The same behavior as for Analysis Requests applies to the worksheet when all its analyses are verified.

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create two Analysis Requests:

```
>>> ar0 = new_ar([Cu, Fe, Au])
>>> ar1 = new_ar([Cu, Fe])
```

Create a worksheet:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
```

And assign all the analyses from the Analysis Requests created before, except *Au* from the first Analysis Request:

```
>>> analyses_ar0 = ar0.getAnalyses(full_objects=True)
>>> analyses_ar1 = ar1.getAnalyses(full_objects=True)
>>> analyses = filter(lambda an: an.getKeyword() != 'Au', analyses_ar0)
>>> analyses += analyses_ar1
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

And submit results for all analyses:

```
>>> submit_analyses(ar0)
>>> submit_analyses(ar1)
```

Of course I cannot verify the whole worksheet, because a worksheet can only be verified once all the analyses it contains are in verified state (and this is done automatically):

```
>>> try_transition(worksheet, "verify", "verified")
False
```

And verify all analyses from worksheet except one:

```
>>> ws_analyses = worksheet.getAnalyses()
>>> analysis_1 = analyses[0]
>>> analysis_2 = analyses[1]
>>> analysis_3 = analyses[2]
>>> analysis_4 = analyses[3]
```

```
>>> try_transition(analysis_2, "verify", "verified")
True
>>> try_transition(analysis_3, "verify", "verified")
True
>>> try_transition(analysis_4, "verify", "verified")
True
```

The Analysis Request number 1 has been automatically transitioned to *verified* cause all the contained analyses have been verified:

```
>>> api.get_workflow_status_of(ar1)
'verified'
```

While Analysis Request number 0 has not been transitioned because have two analyses to be verifed still:

```
>>> api.get_workflow_status_of(ar0)
'to_be_verified'
```

And same with worksheet, cause there is one analysis pending:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And again, I cannot verify the whole worksheet by myself, because a worksheet can only be verified once all the analyses it contains are in verified state (and this is done automatically):

```
>>> try_transition(worksheet, "verify", "verified")
False
```

If we verify the pending analysis from the worksheet:

```
>>> try_transition(analysis_1, "verify", "verified")
True
```

The worksheet will follow:

```
>>> api.get_workflow_status_of(worksheet)
'verified'
```

But the Analysis Request number 0 will remain in *to_be_verified* state:

```
>>> api.get_workflow_status_of(ar0)
'to_be_verified'
```

Unless we verify the analysis *Au*:

```
>>> au_an = filter(lambda an: an.getKeyword() == 'Au', analyses_ar0)[0]
>>> try_transition(au_an, "verify", "verified")
True
```

```
>>> api.get_workflow_status_of(ar0)
'verified'
```

4.71.4 Verification of results for analyses with dependencies

If an analysis is associated to a calculation that uses the result of other analyses (dependents), then the verification of a dependency will auto-verify its dependents.

Reset the interim fields for analysis *Au*:

```
>>> Au.setInterimFields([])
```

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> calc_au.setFormula("([Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu_analysis = filter(lambda an: an.getKeyword()=="Cu", analyses)[0]
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> au_analysis = filter(lambda an: an.getKeyword()=="Au", analyses)[0]
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> cu_analysis.setResult(20)
>>> fe_analysis.setResult(12)
>>> au_analysis.setResult(10)
```

Submit *Au* analysis and the rest will follow:

```
>>> try_transition(au_analysis, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(au_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(fe_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(cu_analysis)
'to_be_verified'
```

If I verify *Au*, the rest of analyses (dependents) will follow too:

```
>>> try_transition(au_analysis, "verify", "verified")
True
>>> api.get_workflow_status_of(au_analysis)
'verified'
>>> api.get_workflow_status_of(fe_analysis)
'verified'
>>> api.get_workflow_status_of(cu_analysis)
'verified'
```

And Analysis Request is transitioned too:

```
>>> api.get_workflow_status_of(ar)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.71.5 Check permissions for Verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu])
>>> submit_analyses(ar)
```

The status of the Analysis Request and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> map(api.get_workflow_status_of, analyses)
['to_be_verified']
```

Exactly these roles can verify:

```
>>> analysis = analyses[0]
>>> get_roles_for_permission("senaite.core: Transition: Verify", analysis)
['LabManager', 'Manager', 'Verifier']
```

Current user can verify because has the *LabManager* role:

```
>>> isTransitionAllowed(analysis, "verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(analysis, "verify")
True
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(analysis, "verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(analysis, "verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(analysis, "verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(analysis, "verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.72 Duplicate Analysis assign guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowDuplicateAnalysisAssign

4.72.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
```

(continues on next page)

(continued from previous page)

```
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.72.2 Assign transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add duplicate analyses from the analyses in position 1

```
>>> duplicates = worksheet.addDuplicateAnalyses(1)
>>> len(duplicates)
3
```

The status of the duplicates is *assigned*:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> map(api.get_workflow_status_of, duplicates)
['assigned', 'assigned', 'assigned']
```

And are associated to the worksheet:

```
>>> wuid = list(set(map(lambda dup: dup.getWorksheetUID(), duplicates)))
>>> len(wuid)
1
```

(continues on next page)

(continued from previous page)

```
>>> wuid[0] == api.get_uid(worksheet)
True
```

Duplicates do not have an Analyst assigned, though:

```
>>> list(set(map(lambda dup: dup.getAnalyst(), duplicates)))
['']
```

If I assign a user to the Worksheet, same user will be assigned to analyses:

```
>>> worksheet.setAnalyst(TEST_USER_ID)
>>> worksheet.getAnalyst() == TEST_USER_ID
True
```

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, analyses)
[]
```

And to the duplicates as well:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, duplicates)
[]
```

I can remove one of the duplicates from the Worksheet:

```
>>> duplicate = duplicates[0]
>>> dup_uid = api.get_uid(duplicate)
>>> worksheet.removeAnalysis(duplicate)
>>> len(worksheet.getDuplicateAnalyses())
2
```

And the removed duplicate no longer exists:

```
>>> api.get_object_by_uid(dup_uid, None) is None
True
```

We add again duplicates for same analyses from slot 1 to slot 2:

```
>>> dup_uids = map(api.get_uid, worksheet.getDuplicateAnalyses())
>>> duplicates = worksheet.addDuplicateAnalyses(1, 2)
```

Since there is only one duplicate analysis missing in slot 2 (that we removed earlier), only one duplicate analysis is added:

```
>>> len(duplicates)
1
>>> len(worksheet.getDuplicateAnalyses())
3
>>> len(filter(lambda dup: dup in duplicates, worksheet.getDuplicateAnalyses()))
1
```

And since the worksheet has an Analyst already assigned, duplicates too:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, duplicates)
[]
```

From *assigned* state I can do submit:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> map(api.get_workflow_status_of, duplicates)
['assigned', 'assigned', 'assigned']
>>> duplicates[0].setResult(20)
>>> duplicates[1].setResult(23)
>>> try_transition(duplicates[0], "submit", "to_be_verified")
True
>>> try_transition(duplicates[1], "submit", "to_be_verified")
True
```

And duplicates transition to *to_be_verified*:

```
>>> map(api.get_workflow_status_of, duplicates)
['to_be_verified', 'to_be_verified', 'assigned']
```

While keeping the Analyst that was assigned to the worksheet:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, duplicates)
[]
```

And since there is still regular analyses in the Worksheet not yet submitted, the Worksheet remains in *open* state:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

Duplicates get removed when I unassign the analyses they come from:

```
>>> duplicate = duplicates[0]
>>> analysis = duplicate.getAnalysis()
>>> dup_uid = api.get_uid(duplicate)
>>> an_uid = api.get_uid(analysis)
>>> worksheet.removeAnalysis(analysis)
>>> api.get_workflow_status_of(analysis)
'unassigned'
>>> filter(lambda an: api.get_uid(an) == an_uid, worksheet.getAnalyses())
[]
>>> filter(lambda dup: api.get_uid(dup.getAnalysis()) == an_uid, worksheet.
↪getDuplicateAnalyses())
[]
>>> len(worksheet.getDuplicateAnalyses())
2
>>> api.get_object_by_uid(dup_uid, None) is None
True
```

I submit the results for the rest of analyses:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(10)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified']
```

And since there is a duplicate that has not been yet submitted, the Worksheet remains in *open* state:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> duplicate = filter(lambda dup: api.get_workflow_status_of(dup) == "assigned",␣
↪duplicates)
```

(continues on next page)

(continued from previous page)

```
>>> len(duplicate)
1
>>> duplicate = duplicate[0]
>>> api.get_workflow_status_of(duplicate)
'assigned'
>>> api.get_workflow_status_of(worksheet)
'open'
```

But if I remove the duplicate analysis that has not been yet submitted, the status of the Worksheet is promoted to *to_be_verified*, cause all the rest are in *to_be_verified* state:

```
>>> dup_uid = api.get_uid(duplicate)
>>> worksheet.removeAnalysis(duplicate)
>>> len(worksheet.getDuplicateAnalyses())
1
>>> api.get_object_by_uid(dup_uid, None) is None
True
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And now, I cannot add duplicates anymore:

```
>>> worksheet.addDuplicateAnalyses(1)
[]
>>> len(worksheet.getDuplicateAnalyses())
1
```

4.72.3 Check permissions for Assign transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add a Duplicate analysis of the analysis in position 1:

```
>>> len(worksheet.addDuplicateAnalyses(1))
3
```

Since a duplicate can only live inside a Worksheet, the initial state of the duplicate is *assigned* by default:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> map(api.get_workflow_status_of, duplicates)
['assigned', 'assigned', 'assigned']
```

4.73 Duplicate Analysis multi-verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowDuplicateAnalysisMultiVerify

4.73.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getReviewHistory
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.73.2 Multiverify not allowed if multi-verification is not enabled

Enable self verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

The status of duplicate and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot multi verify the analysis because multi-verification is not set:

```
>>> isTransitionAllowed(duplicate, "multi_verify")
False
>>> try_transition(duplicate, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

But I can verify:

```
>>> isTransitionAllowed(duplicate, "verify")
True
>>> try_transition(duplicate, "verify", "verified")
True
```

And the status of the duplicate is now *verified*:

```
>>> api.get_workflow_status_of(duplicate)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.73.3 Multiverify transition with multi-verification enabled

The system allows to set multiple verifiers, both at Setup or Analysis Service level. If set, the analysis will transition to verified when the total number of verifications equals to the value set in multiple-verifiers.

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Not allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_disabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

The status of duplicate and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot *verify*:

```
>>> isTransitionAllowed(duplicate, "verify")
False
>>> try_transition(duplicate, "verify", "verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

Because multi-verification is enabled:

```
>>> bikasetup.getNumberOfRequiredVerifications()
3
```

And there are 3 verifications remaining:

```
>>> duplicate.getNumberOfRemainingVerifications()
3
```

But I can multi-verify:

```
>>> isTransitionAllowed(duplicate, "multi_verify")
True
>>> try_transition(duplicate, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

And my user id is recorded as such:

```
>>> action = getReviewHistory(duplicate)[0]
>>> action['actor'] == TEST_USER_ID
True
```

And now, there are two verifications remaining:

```
>>> duplicate.getNumberOfRemainingVerifications()
2
```

So, I cannot verify yet:

```
>>> isTransitionAllowed(duplicate, "verify")
False
>>> try_transition(duplicate, "verify", "verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

But I cannot multi-verify neither, cause I am the same user who did the last multi-verification:

```
>>> isTransitionAllowed(duplicate, "multi_verify")
False
>>> try_transition(duplicate, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

And the system is configured to not allow same user to verify multiple times:

```
>>> bikasetup.getTypeOfmultiVerification()
'self_multi_disabled'
```

But I can multi-verify if I change the type of multi-verification:

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
>>> isTransitionAllowed(duplicate, "multi_verify")
True
>>> try_transition(duplicate, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

Since there is only one verification remaining, I cannot multi-verify again:

```
>>> duplicate.getNumberOfRemainingVerifications()
1
>>> isTransitionAllowed(duplicate, "multi_verify")
False
>>> try_transition(duplicate, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

But now, I can verify:

```
>>> isTransitionAllowed(duplicate, "verify")
True
>>> try_transition(duplicate, "verify", "verified")
True
```

There is no verifications remaining:

```
>>> duplicate.getNumberOfRemainingVerifications()
0
```

And the status of the duplicate is now *verified*:

```
>>> api.get_workflow_status_of(duplicate)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

If we multi-verify the regular analysis (2+1 times):

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "verify", "verified")
True
```

The rest transition to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
>>> api.get_workflow_status_of(worksheet)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.73.4 Check permissions for Multi verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

Exactly these roles can multi_verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", duplicate)
['LabManager', 'Manager', 'Verifier']
```

Current user can multi_verify because has the *LabManager* role:

```
>>> isTransitionAllowed(duplicate, "multi_verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(duplicate, "multi_verify")
True
```

TODO Workflow Verifier should be able to multi_verify a duplicate! The code below throws an *Unauthorized: Not authorized to access binding: context* error, rised by https://github.com/MatthewWilkes/Zope/blob/master/src/Shared/DC/Scripts/Bindings.py#L198

>>> setRoles(portal, TEST_USER_ID, ['Verifier',]) # >>> isTransitionAllowed(duplicate, "multi_verify") # True

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(duplicate, "multi_verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(duplicate, "multi_verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(duplicate, "multi_verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.74 Duplicate Analysis retract guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowDuplicateAnalysisRetract

4.74.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.74.2 Retract transition and guard basic constraints

Create an Analysis Request and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

Retract the duplicate:

```
>>> try_transition(duplicate, "retract", "retracted")
True
>>> api.get_workflow_status_of(duplicate)
'retracted'
```

And one new additional duplicate has been added in *assigned* state:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> sorted(map(api.get_workflow_status_of, duplicates))
['assigned', 'retracted']
```

And the Worksheet has been transitioned to *open*:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

While the Analysis Request is still in *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The new analysis is a copy of retracted one:

```
>>> retest = filter(lambda an: api.get_workflow_status_of(an) == "assigned",␣
↪duplicates)[0]
>>> retest.getKeyword() == duplicate.getKeyword()
True
>>> retest.getReferenceAnalysesGroupID() == duplicate.getReferenceAnalysesGroupID()
True
>>> retest.getRetestOf() == duplicate
True
>>> duplicate.getRetest() == retest
True
>>> retest.getAnalysis() == duplicate.getAnalysis()
True
```

And keeps the same results as the retracted one:

```
>>> retest.getResult() == duplicate.getResult()
True
```

And is located in the same slot as well:

```
>>> worksheet.get_slot_position_for(duplicate) == worksheet.get_slot_position_
↪for(retest)
True
```

If I submit the result for the new duplicate:

```
>>> try_transition(retest, "submit", "to_be_verified")
True
```

The status of both the duplicate and the Worksheet is “to_be_verified”:

```
>>> api.get_workflow_status_of(retest)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And I can even retract the retest:

```
>>> try_transition(retest, "retract", "retracted")
True
>>> api.get_workflow_status_of(retest)
'retracted'
```

And one new additional duplicate has been added in *assigned* state:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> sorted(map(api.get_workflow_status_of, duplicates))
['assigned', 'retracted', 'retracted']
```

And the Worksheet has been transitioned to *open*:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

4.74.3 Auto-rollback of Worksheet on analysis retraction

When retracting an analysis from a Worksheet that is in "to_be_verified" state causes the rollback of the worksheet to "open" state.

Create an Analysis Request and submit results:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a new Worksheet, assign all analyses and submit:

```
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
>>> submit_analyses(ar)
```

The state for both the Analysis Request and Worksheet is "to_be_verified":

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

Retract one analysis:

```
>>> analysis = ws.getAnalyses()[0]
>>> try_transition(analysis, "retract", "retracted")
True
```

A rollback of the state of Analysis Request and Worksheet takes place:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
>>> api.get_workflow_status_of(ws)
'open'
```

And both contain an additional analysis:

```
>>> len(ar.getAnalyses())
4
>>> len(ws.getAnalyses())
4
```

The state of this additional analysis, the retest, is "assigned":

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> retest = filter(lambda an: api.get_workflow_status_of(an) == "assigned",␣
↪analyses)[0]
>>> retest.getKeyword() == analysis.getKeyword()
True
>>> retest in ws.getAnalyses()
True
```

4.74.4 Retraction of results for analyses with dependents

When retracting an analysis other analyses depends on (dependents), then the retraction of a dependency causes the auto-retraction of its dependents.

Prepare a calculation that depends on *Cu*‘and assign it to ‘*Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> calc_au.setFormula("([Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu_analysis = filter(lambda an: an.getKeyword()=="Cu", analyses)[0]
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> au_analysis = filter(lambda an: an.getKeyword()=="Au", analyses)[0]
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to ‘api.analysis.calculate‘:

```
>>> cu_analysis.setResult(20)
>>> fe_analysis.setResult(12)
>>> au_analysis.setResult(10)
```

Submit *Au* analysis and the rest will follow:

```
>>> try_transition(au_analysis, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(au_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(fe_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(cu_analysis)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

If I retract *Fe*, *Au* analysis is retracted automatically too:

```
>>> try_transition(fe_analysis, "retract", "retracted")
True
>>> api.get_workflow_status_of(fe_analysis)
'retracted'
>>> api.get_workflow_status_of(au_analysis)
'retracted'
```

As well as *Cu* analysis (a dependency of *Fe*):

```
>>> api.get_workflow_status_of(cu_analysis)
'retracted'
```

Hence, three new analyses are generated in accordance:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> len(analyses)
6
>>> au_analyses = filter(lambda an: an.getKeyword()=="Au", analyses)
>>> sorted(map(api.get_workflow_status_of, au_analyses))
['retracted', 'unassigned']
>>> fe_analyses = filter(lambda an: an.getKeyword()=="Fe", analyses)
>>> sorted(map(api.get_workflow_status_of, fe_analyses))
['retracted', 'unassigned']
>>> fe_analyses = filter(lambda an: an.getKeyword()=="Cu", analyses)
>>> sorted(map(api.get_workflow_status_of, fe_analyses))
['retracted', 'unassigned']
```

And the current state of the Analysis Request is *sample_received* now:

```
>>> api.get_workflow_status_of(ar)
'sample_received'
```

4.75 Duplicate Analysis submission guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowDuplicateAnalysisSubmit

4.75.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
```

(continues on next page)

(continued from previous page)

```
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.75.2 Duplicate submission basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get a duplicate:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
```

Cannot submit a duplicate without a result:

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

Even if we try with an empty or None result:

```
>>> duplicate.setResult('')
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

```
>>> duplicate.setResult(None)
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

But will work if we try with a result of 0:

```
>>> duplicate.setResult(0)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

And we cannot re-submit a duplicate that have been submitted already:

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

4.75.3 Auto submission of a Worksheets when all its analyses are submitted

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a worksheet:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
```

And assign all analyses from the Analysis Request created before:

```
>>> for analysis in ar.getAnalyses(full_objects=True):
...     worksheet.addAnalysis(analysis)
```

Add a Duplicate of sample from position 1:

```
>>> duplicates = worksheet.addDuplicateAnalyses(1)
```

Set results and submit all analyses from the worksheet except the duplicates:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(13)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

While the Analysis Request has been transitioned to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The worksheet has not been transitioned:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

Because duplicates are still in *assigned* state:

```
>>> map(api.get_workflow_status_of, worksheet.getDuplicateAnalyses())
['assigned', 'assigned', 'assigned']
```

If we set results and submit duplicates:

```
>>> for analysis in worksheet.getDuplicateAnalyses():
...     analysis.setResult(13)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getDuplicateAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

The worksheet will automatically be submitted too:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

4.75.4 Submission of duplicates with interim fields set

Set interims to the analysis *Au*:

```
>>> Au.setInterimFields([
...     {"keyword": "interim_1", "title": "Interim 1",},
...     {"keyword": "interim_2", "title": "Interim 2",}])
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
```

Cannot submit if no result is set:

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

But even if we set a result, we cannot submit because interims are missing:

```
>>> duplicate.setResult(12)
>>> duplicate.getResult()
'12'
```

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

So, if the duplicate has interims defined, all them are required too:

```
>>> duplicate.setInterimValue("interim_1", 15)
>>> duplicate.getInterimValue("interim_1")
'15'
```

```
>>> duplicate.getInterimValue("interim_2")
''
```

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

Even if we set a non-valid (None, empty) value to an interim:

```
>>> duplicate.setInterimValue("interim_2", None)
>>> duplicate.getInterimValue("interim_2")
''
```

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

```
>>> duplicate.setInterimValue("interim_2", '')
>>> duplicate.getInterimValue("interim_2")
''
```

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

But it will work if the value is 0:

```
>>> duplicate.setInterimValue("interim_2", 0)
>>> duplicate.getInterimValue("interim_2")
'0'
```

```
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

Might happen the other way round. We set interims but not a result:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setInterimValue("interim_1", 10)
>>> duplicate.setInterimValue("interim_2", 20)
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

Still, the result is required:

```
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

4.75.5 Submission of duplicates with interim calculation

If a duplicate have a calculation assigned, the result will be calculated automatically based on the calculation. If the calculation have interims set, only those that do not have a default value set will be required.

Prepare the calculation and set the calculation to analysis *Au*:

```
>>> Au.setInterimFields([])
>>> calc = api.create(bikasetup.bika_calculations, 'Calculation', title='Test␣
↪Calculation')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1', 'value': 10}
>>> interim_2 = {'keyword': 'IT2', 'title': 'Interim 2', 'value': 2}
>>> interim_3 = {'keyword': 'IT3', 'title': 'Interim 3', 'value': ''}
>>> interim_4 = {'keyword': 'IT4', 'title': 'Interim 4', 'value': None}
>>> interim_5 = {'keyword': 'IT5', 'title': 'Interim 5'}
>>> interims = [interim_1, interim_2, interim_3, interim_4, interim_5]
>>> calc.setInterimFields(interims)
>>> calc.setFormula("[IT1]+[IT2]+[IT3]+[IT4]+[IT5]")
>>> Au.setCalculation(calc)
```

Create a Worksheet with duplicate:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
```

Cannot submit if no result is set

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> duplicate.setResult(34)
```

Set a value for interim IT5:

```
>>> duplicate.setInterimValue("IT5", 5)
```

Cannot transition because IT3 and IT4 have None/empty values as default:

```
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

Let's set a value for those interims:

```
>>> duplicate.setInterimValue("IT3", 3)
>>> try_transition(duplicate, "submit", "to_be_verified")
False
```

```
>>> duplicate.setInterimValue("IT4", 4)
```

Since interims IT1 and IT2 have default values set, the analysis will submit:

```
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

4.75.6 Submission of duplicates with dependencies

Duplicates with dependencies are not allowed. Duplicates can only be created from analyses without dependents.

TODO We might consider to allow the creation of duplicates with deps

Reset the interim fields for analysis *Au*:

```
>>> Au.setInterimFields([])
```

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1'}
>>> calc_au.setInterimFields([interim_1])
>>> calc_au.setFormula("([IT1]+[Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a Worksheet with duplicate:

```
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> analyses = worksheet.getRegularAnalyses()
```

Only one duplicate created for *Cu*, cause is the only analysis that does not have dependents:

```
>>> duplicates = worksheet.getDuplicateAnalyses()
>>> len(duplicates) == 1
True
```

```
>>> duplicate = duplicates[0]
>>> duplicate.getKeyword()
'Cu'
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to ‘api.analysis.calculate‘:

```
>>> duplicate.setResult(12)
```

Cannot submit routine *Fe* cause there is no result for routine analysis *Cu* and the duplicate of *Cu* cannot be used as a dependent:

```
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> try_transition(fe_analysis, "submit", "to_be_verified")
False
```

4.75.7 Check permissions for Submit transition

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Set a result:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(23)
```

Exactly these roles can submit:

```
>>> get_roles_for_permission("senaite.core: Edit Results", duplicate)
['Analyst', 'LabManager', 'Manager']
```

And these roles can view results:

```
>>> get_roles_for_permission("senaite.core: View Results", duplicate)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'RegulatoryInspector']
```

Current user can submit because has the *LabManager* role:

```
>>> isTransitionAllowed(duplicate, "submit")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Authenticated', 'LabClerk', 'RegulatoryInspector
↪', 'Sampler'])
>>> isTransitionAllowed(duplicate, "submit")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(duplicate, "submit")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(duplicate, "submit")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.76 Duplicate Analysis verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowDuplicateAnalysisVerify

4.76.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
```

(continues on next page)

(continued from previous page)

```
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_duplicate(ar):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...     worksheet.addDuplicateAnalyses(1)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
```

(continues on next page)

(continued from previous page)

```
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.76.2 Duplicate verification basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

I cannot verify the duplicate because I am the same user who submitted:

```
>>> try_transition(duplicate, "verify", "verified")
False
>>> api.get_workflow_status_of(duplicate)
'to_be_verified'
```

And I cannot verify the Worksheet, because it can only be verified once all analyses it contains are verified (and this is done automatically):

```
>>> try_transition(worksheet, "verify", "verified")
False
```

(continues on next page)

(continued from previous page)

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

But if I enable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Then, I can verify my own result:

```
>>> try_transition(duplicate, "verify", "verified")
True
```

And the worksheet transitions to *verified*:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And we cannot re-verify a duplicate that has been verified already:

```
>>> try_transition(duplicate, "verify", "verified")
False
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.76.3 Check permissions for Verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_duplicate(ar)
>>> submit_regular_analyses(worksheet)
```

Get the duplicate and submit:

```
>>> duplicate = worksheet.getDuplicateAnalyses()[0]
>>> duplicate.setResult(12)
>>> try_transition(duplicate, "submit", "to_be_verified")
True
```

Exactly these roles can verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", duplicate)
['LabManager', 'Manager', 'Verifier']
```

Current user can verify because has the *LabManager* role:

```
>>> isTransitionAllowed(duplicate, "verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(duplicate, "verify")
True
```

TODO Workflow Verifier should be able to verify a duplicate! The code below throws an *Unauthorized: Not authorized to access binding: context* error, rised by https://github.com/MatthewWilkes/Zope/blob/master/src/Shared/DC/Scripts/Bindings.py#L198

>>> setRoles(portal, TEST_USER_ID, [‘Verifier’,]) # >>> isTransitionAllowed(duplicate, “verify”) # True

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(duplicate, "verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(duplicate, "verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(duplicate, "verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.77 Reference Analysis (Blanks) assign guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisBlankAssign

4.77.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> ref_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",␣
↪title="Blank definition", Blank=True)
>>> ref_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...             {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...             {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> ref_def.setReferenceResults(ref_refs)
>>> ref_sample = api.create(supplier, "ReferenceSample", title="Blank",
...                         ReferenceDefinition=ref_def,
...                         Blank=True, ExpiryDate=date_future,
...                         ReferenceResults=ref_refs)
```

4.77.2 Assign transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add a blank:

```
>>> ref_analyses = worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au])
>>> len(ref_analyses)
3
```

The status of the reference analyses is *assigned*:

```
>>> ref_analyses = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, ref_analyses)
['assigned', 'assigned', 'assigned']
```

All them are blanks:

```
>>> map(lambda ref: ref.getReferenceType(), ref_analyses)
['b', 'b', 'b']
```

And are associated to the worksheet:

```
>>> wuid = list(set(map(lambda ref: ref.getWorksheetUID(), ref_analyses)))
>>> len(wuid)
1
>>> wuid[0] == api.get_uid(worksheet)
True
```

Blanks do not have an Analyst assigned, though:

```
>>> list(set(map(lambda ref: ref.getAnalyst(), ref_analyses)))
['']
```

If I assign a user to the Worksheet, same user will be assigned to analyses:

```
>>> worksheet.setAnalyst(TEST_USER_ID)
>>> worksheet.getAnalyst() == TEST_USER_ID
True
```

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, analyses)
[]
```

And to the blanks as well:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, ref_analyses)
[]
```

I can remove one of the blanks from the Worksheet:

```
>>> ref = ref_analyses[0]
>>> ref_uid = api.get_uid(ref)
>>> worksheet.removeAnalysis(ref)
>>> len(worksheet.getReferenceAnalyses())
2
```

And the removed blank no longer exists:

```
>>> api.get_object_by_uid(ref_uid, None) is None
True
```

From *assigned* state I can do submit:

```
>>> ref_analyses = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, ref_analyses)
['assigned', 'assigned']
>>> ref_analyses[0].setResult(20)
>>> try_transition(ref_analyses[0], "submit", "to_be_verified")
True
```

And blanks transition to *to_be_verified*:

```
>>> map(api.get_workflow_status_of, ref_analyses)
['to_be_verified', 'assigned']
```

While keeping the Analyst that was assigned to the worksheet:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, ref_analyses)
[]
```

And since there is still regular analyses in the Worksheet not yet submitted, the Worksheet remains in *open* state:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

I submit the results for the rest of analyses:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(10)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

And since there is a blank that has not been yet submitted, the Worksheet remains in *open* state:

```
>>> ref = worksheet.getReferenceAnalyses()[1]
>>> api.get_workflow_status_of(ref)
'assigned'
>>> api.get_workflow_status_of(worksheet)
'open'
```

But if I remove the blank that has not been yet submitted, the status of the Worksheet is promoted to *to_be_verified*, cause all the rest are in *to_be_verified* state:

```
>>> ref_uid = api.get_uid(ref)
>>> worksheet.removeAnalysis(ref)
>>> len(worksheet.getReferenceAnalyses())
1
>>> api.get_object_by_uid(ref_uid, None) is None
True
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And the blank itself no longer exists in the system:

```
>>> api.get_object_by_uid(ref_uid, None) == None
True
```

And now, I cannot add blanks anymore:

```
>>> worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au])
[]
>>> len(worksheet.getReferenceAnalyses())
1
```

4.77.3 Check permissions for Assign transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add blank analyses:

```
>>> len(worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au]))
3
```

Since a reference analysis can only live inside a Worksheet, the initial state of the blank is *assigned* by default:

```
>>> duplicates = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, duplicates)
['assigned', 'assigned', 'assigned']
```

4.78 Reference Analysis (Blank) multi-verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisBlankMultiVerify

4.78.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getReviewHistory
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
```

(continues on next page)

(continued from previous page)

```
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> blank_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",
↪ title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> blank_def.setReferenceResults(blank_refs)
>>> blank_sample = api.create(supplier, "ReferenceSample", title="Blank",
...                     ReferenceDefinition=blank_def,
...                     Blank=True, ExpiryDate=date_future,
...                     ReferenceResults=blank_refs)
```

4.78.2 Multiverify not allowed if multi-verification is not enabled

Enable self verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get the blank and submit:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(0)
>>> try_transition(blank, "submit", "to_be_verified")
True
```

The status of blank and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot multi verify the blank because multi-verification is not set:

```
>>> isTransitionAllowed(blank, "multi_verify")
False
>>> try_transition(blank, "multi_verify", "to_be_verified")
False
```

(continues on next page)

(continued from previous page)

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

But I can verify:

```
>>> isTransitionAllowed(blank, "verify")
True
>>> try_transition(blank, "verify", "verified")
True
```

And the status of the blank is now *verified*:

```
>>> api.get_workflow_status_of(blank)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.78.3 Multiverify transition with multi-verification enabled

The system allows to set multiple verifiers, both at Setup or Analysis Service level. If set, the blank will transition to verified when the total number of verifications equals to the value set in multiple-verifiers.

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Not allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_disabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get the blank and submit:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(12)
>>> try_transition(blank, "submit", "to_be_verified")
True
```

The status of blank and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot *verify*:

```
>>> isTransitionAllowed(blank, "verify")
False
>>> try_transition(blank, "verify", "verified")
False
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

Because multi-verification is enabled:

```
>>> bikasetup.getNumberOfRequiredVerifications()
3
```

And there are 3 verifications remaining:

```
>>> blank.getNumberOfRemainingVerifications()
3
```

But I can multi-verify:

```
>>> isTransitionAllowed(blank, "multi_verify")
True
>>> try_transition(blank, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

And my user id is recorded as such:

```
>>> action = getReviewHistory(blank)[0]
>>> action['actor'] == TEST_USER_ID
True
```

And now, there are two verifications remaining:

```
>>> blank.getNumberOfRemainingVerifications()
2
```

So, I cannot verify yet:

```
>>> isTransitionAllowed(blank, "verify")
False
>>> try_transition(blank, "verify", "verified")
False
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

But I cannot multi-verify neither, cause I am the same user who did the last multi-verification:

```
>>> isTransitionAllowed(blank, "multi_verify")
False
>>> try_transition(blank, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

And the system is configured to not allow same user to verify multiple times:

```
>>> bikasetup.getTypeOfmultiVerification()
'self_multi_disabled'
```

But I can multi-verify if I change the type of multi-verification:

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
>>> isTransitionAllowed(blank, "multi_verify")
True
>>> try_transition(blank, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

Since there is only one verification remaining, I cannot multi-verify again:

```
>>> blank.getNumberOfRemainingVerifications()
1
>>> isTransitionAllowed(blank, "multi_verify")
False
>>> try_transition(blank, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

But now, I can verify:

```
>>> isTransitionAllowed(blank, "verify")
True
>>> try_transition(blank, "verify", "verified")
True
```

There is no verifications remaining:

```
>>> blank.getNumberOfRemainingVerifications()
0
```

And the status of the blank is now *verified*:

```
>>> api.get_workflow_status_of(blank)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

If we multi-verify the regular analysis (2+1 times):

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "verify", "verified")
True
```

The rest transition to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
>>> api.get_workflow_status_of(worksheet)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.78.4 Check permissions for Multi verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get the blank and submit:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(12)
>>> try_transition(blank, "submit", "to_be_verified")
True
```

Exactly these roles can multi_verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", blank)
['LabManager', 'Manager', 'Verifier']
```

Current user can multi_verify because has the *LabManager* role:

```
>>> isTransitionAllowed(blank, "multi_verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(blank, "multi_verify")
True
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(blank, "multi_verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(blank, "multi_verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(blank, "multi_verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(blank, "multi_verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.79 Reference Analysis (Blanks) submission guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisBlankSubmit

4.79.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...       success = do_action_for(object, transition_id)[0]
...       state = api.get_workflow_status_of(object)
...       return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> blank_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",
↪ title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> blank_def.setReferenceResults(blank_refs)
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> blank_sample = api.create(supplier, "ReferenceSample", title="Blank",
...                     ReferenceDefinition=blank_def,
...                     Blank=True, ExpiryDate=date_future,
...                     ReferenceResults=blank_refs)
>>> control_sample = api.create(supplier, "ReferenceSample", title="Control",
...                       ReferenceDefinition=control_def,
```

(continues on next page)

(continued from previous page)

```
...                         Blank=False, ExpiryDate=date_future,
...                         ReferenceResults=control_refs)
```

4.79.2 Blank submission basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get blank analyses:

```
>>> blanks = worksheet.getReferenceAnalyses()
>>> blank_1 = blanks[0]
>>> blank_2 = blanks[1]
>>> blank_3 = blanks[2]
```

Cannot submit a blank without a result:

```
>>> try_transition(blank_1, "submit", "to_be_verified")
False
```

Even if we try with an empty or None result:

```
>>> blank_1.setResult('')
>>> try_transition(blank_1, "submit", "to_be_verified")
False
```

```
>>> blank_1.setResult(None)
>>> try_transition(blank_1, "submit", "to_be_verified")
False
```

But will work if we try with a result of 0:

```
>>> blank_1.setResult(0)
>>> try_transition(blank_1, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(blank_1)
'to_be_verified'
```

And we cannot re-submit a blank that have been submitted already:

```
>>> try_transition(blank_1, "submit", "to_be_verified")
False
```

4.79.3 Auto submission of a Worksheets when all its analyses are submitted

Create a Worksheet:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
```

Set results and submit all analyses from the worksheet except blanks:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(13)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

While the Analysis Request has been transitioned to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The worksheet has not been transitioned:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

Because blanks are still in *assigned* state:

```
>>> map(api.get_workflow_status_of, worksheet.getReferenceAnalyses())
['assigned', 'assigned', 'assigned']
```

If we set results and submit blanks:

```
>>> for analysis in worksheet.getReferenceAnalyses():
...     analysis.setResult(0)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getReferenceAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

The worksheet will automatically be submitted too:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

4.79.4 Submission of blanks with interim fields set

Set interims to the analysis *Au*:

```
>>> Au.setInterimFields([
...     {"keyword": "interim_1", "title": "Interim 1",},
...     {"keyword": "interim_2", "title": "Interim 2",}])
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get blank analyses:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
```

Cannot submit if no result is set:

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

But even if we set a result, we cannot submit because interims are missing:

```
>>> blank.setResult(12)
>>> blank.getResult()
'12'
```

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

So, if the blank has interims defined, all them are required too:

```
>>> blank.setInterimValue("interim_1", 15)
>>> blank.getInterimValue("interim_1")
'15'
```

```
>>> blank.getInterimValue("interim_2")
''
```

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

Even if we set a non-valid (None, empty) value to an interim:

```
>>> blank.setInterimValue("interim_2", None)
>>> blank.getInterimValue("interim_2")
''
```

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

```
>>> blank.setInterimValue("interim_2", '')
>>> blank.getInterimValue("interim_2")
''
```

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

But it will work if the value is 0:

```
>>> blank.setInterimValue("interim_2", 0)
>>> blank.getInterimValue("interim_2")
'0'
```

```
>>> try_transition(blank, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

Might happen the other way round. We set interims but not a result:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setInterimValue("interim_1", 10)
>>> blank.setInterimValue("interim_2", 20)
>>> try_transition(blank, "submit", "to_be_verified")
False
```

Still, the result is required:

```
>>> blank.setResult(12)
>>> try_transition(blank, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

4.79.5 Submission of blank analysis with interim calculation

If a blank analysis have a calculation assigned, the result will be calculated automatically based on the calculation. If the calculation have interims set, only those that do not have a default value set will be required.

Prepare the calculation and set the calculation to analysis *Au*:

```
>>> Au.setInterimFields([])
>>> calc = api.create(bikasetup.bika_calculations, 'Calculation', title='Test␣
↪Calculation')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1', 'value': 10}
>>> interim_2 = {'keyword': 'IT2', 'title': 'Interim 2', 'value': 2}
>>> interim_3 = {'keyword': 'IT3', 'title': 'Interim 3', 'value': ''}
>>> interim_4 = {'keyword': 'IT4', 'title': 'Interim 4', 'value': None}
>>> interim_5 = {'keyword': 'IT5', 'title': 'Interim 5'}
>>> interims = [interim_1, interim_2, interim_3, interim_4, interim_5]
>>> calc.setInterimFields(interims)
>>> calc.setFormula("[IT1]+[IT2]+[IT3]+[IT4]+[IT5]")
>>> Au.setCalculation(calc)
```

Create a Worksheet with blank:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
```

Cannot submit if no result is set

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> try_transition(blank, "submit", "to_be_verified")
False
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> blank.setResult(34)
```

Set a value for interim IT5:

```
>>> blank.setInterimValue("IT5", 5)
```

Cannot transition because IT3 and IT4 have None/empty values as default:

```
>>> try_transition(blank, "submit", "to_be_verified")
False
```

Let's set a value for those interims:

```
>>> blank.setInterimValue("IT3", 3)
>>> try_transition(blank, "submit", "to_be_verified")
False
```

```
>>> blank.setInterimValue("IT4", 4)
```

Since interims IT1 and IT2 have default values set, the analysis will submit:

```
>>> try_transition(blank, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

4.79.6 Submission of blanks with dependencies

Blanks with dependencies are not allowed. Blanks can only be created from analyses without dependents.

TODO Might we consider to allow the creation of blanks with dependencies?

Reset the interim fields for analysis *Au*:

```
>>> Au.setInterimFields([])
```

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1'}
>>> calc_au.setInterimFields([interim_1])
>>> calc_au.setFormula("([IT1]+[Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a Worksheet with blank:

```
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> analyses = worksheet.getRegularAnalyses()
```

Only one blank created for *Cu*, cause is the only analysis that does not have dependents:

```
>>> blanks = worksheet.getReferenceAnalyses()
>>> len(blanks) == 1
True
```

```
>>> blank = blanks[0]
>>> blank.getKeyword()
'Cu'
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> blank.setResult(0)
```

Cannot submit routine *Fe* cause there is no result for routine analysis *Cu* and the blank of *Cu* cannot be used as a dependent:

```
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> try_transition(fe_analysis, "submit", "to_be_verified")
False
```

4.79.7 Check permissions for Submit transition

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Set a result:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(23)
```

Exactly these roles can submit:

```
>>> get_roles_for_permission("senaite.core: Edit Results", blank)
['Analyst', 'LabManager', 'Manager']
```

And these roles can view results:

```
>>> get_roles_for_permission("senaite.core: View Results", blank)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'RegulatoryInspector']
```

Current user can submit because has the *LabManager* role:

```
>>> isTransitionAllowed(blank, "submit")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Authenticated', 'LabClerk', 'RegulatoryInspector
↪', 'Sampler'])
>>> isTransitionAllowed(blank, "submit")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(blank, "submit")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(blank, "submit")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.80 Reference Analysis (Blanks) verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisBlankVerify

4.80.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
```

(continues on next page)

(continued from previous page)

```
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> blank_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",
↪ title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> blank_def.setReferenceResults(blank_refs)
>>> blank_sample = api.create(supplier, "ReferenceSample", title="Blank",
...                     ReferenceDefinition=blank_def,
...                     Blank=True, ExpiryDate=date_future,
...                     ReferenceResults=blank_refs)
```

4.80.2 Blank verification basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get the blank and submit:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(0)
>>> try_transition(blank, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

I cannot verify the blank because I am the same user who submitted:

```
>>> try_transition(blank, "verify", "verified")
False
>>> api.get_workflow_status_of(blank)
'to_be_verified'
```

And I cannot verify the Worksheet, because it can only be verified once all analyses it contains are verified (and this is done automatically):

```
>>> try_transition(worksheet, "verify", "verified")
False
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

But if I enable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Then, I can verify my own result:

```
>>> try_transition(blank, "verify", "verified")
True
```

And the worksheet transitions to *verified*:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And we cannot re-verify a blank that has been verified already:

```
>>> try_transition(blank, "verify", "verified")
False
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.80.3 Check permissions for Verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, blank_sample)
>>> submit_regular_analyses(worksheet)
```

Get the blank and submit:

```
>>> blank = worksheet.getReferenceAnalyses()[0]
>>> blank.setResult(12)
>>> try_transition(blank, "submit", "to_be_verified")
True
```

Exactly these roles can verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", blank)
['LabManager', 'Manager', 'Verifier']
```

Current user can verify because has the *LabManager* role:

```
>>> isTransitionAllowed(blank, "verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(blank, "verify")
True
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(blank, "verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(blank, "verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(blank, "verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(blank, "verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.81 Reference Analysis (Controls) assign guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisControlAssign

4.81.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
```

(continues on next page)

(continued from previous page)

```
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     return ar
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
```

(continues on next page)

(continued from previous page)

```
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> ref_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",␣
↪title="control definition", control=True)
>>> ref_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...             {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...             {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> ref_def.setReferenceResults(ref_refs)
>>> ref_sample = api.create(supplier, "ReferenceSample", title="control",
...                         ReferenceDefinition=ref_def,
...                         control=True, ExpiryDate=date_future,
...                         ReferenceResults=ref_refs)
```

4.81.2 Assign transition and guard basic constraints

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add a control:

```
>>> ref_analyses = worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au])
>>> len(ref_analyses)
3
```

The status of the reference analyses is *assigned*:

```
>>> ref_analyses = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, ref_analyses)
['assigned', 'assigned', 'assigned']
```

All them are controls:

```
>>> map(lambda ref: ref.getReferenceType(), ref_analyses)
['c', 'c', 'c']
```

And are associated to the worksheet:

```
>>> wuid = list(set(map(lambda ref: ref.getWorksheetUID(), ref_analyses)))
>>> len(wuid)
1
>>> wuid[0] == api.get_uid(worksheet)
True
```

controls do not have an Analyst assigned, though:

```
>>> list(set(map(lambda ref: ref.getAnalyst(), ref_analyses)))
['']
```

If I assign a user to the Worksheet, same user will be assigned to analyses:

```
>>> worksheet.setAnalyst(TEST_USER_ID)
>>> worksheet.getAnalyst() == TEST_USER_ID
True
```

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, analyses)
[]
```

And to the controls as well:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, ref_analyses)
[]
```

I can remove one of the controls from the Worksheet:

```
>>> ref = ref_analyses[0]
>>> ref_uid = api.get_uid(ref)
>>> worksheet.removeAnalysis(ref)
>>> len(worksheet.getReferenceAnalyses())
2
```

And the removed control no longer exists:

```
>>> api.get_object_by_uid(ref_uid, None) is None
True
```

From *assigned* state I can do submit:

```
>>> ref_analyses = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, ref_analyses)
['assigned', 'assigned']
>>> ref_analyses[0].setResult(20)
>>> try_transition(ref_analyses[0], "submit", "to_be_verified")
True
```

And controls transition to *to_be_verified*:

```
>>> map(api.get_workflow_status_of, ref_analyses)
['to_be_verified', 'assigned']
```

While keeping the Analyst that was assigned to the worksheet:

```
>>> filter(lambda an: an.getAnalyst() != TEST_USER_ID, ref_analyses)
[]
```

And since there is still regular analyses in the Worksheet not yet submitted, the Worksheet remains in *open* state:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

I submit the results for the rest of analyses:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(10)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

And since there is a control that has not been yet submitted, the Worksheet remains in *open* state:

```
>>> ref = worksheet.getReferenceAnalyses()[1]
>>> api.get_workflow_status_of(ref)
'assigned'
>>> api.get_workflow_status_of(worksheet)
'open'
```

But if I remove the control that has not been yet submitted, the status of the Worksheet is promoted to *to_be_verified*, cause all the rest are in *to_be_verified* state:

```
>>> ref_uid = api.get_uid(ref)
>>> worksheet.removeAnalysis(ref)
>>> len(worksheet.getReferenceAnalyses())
1
>>> api.get_object_by_uid(ref_uid, None) is None
True
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And the control itself no longer exists in the system:

```
>>> api.get_object_by_uid(ref_uid, None) == None
True
```

And now, I cannot add controls anymore:

```
>>> worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au])
[]
>>> len(worksheet.getReferenceAnalyses())
1
```

4.81.3 Check permissions for Assign transition

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> transitioned = do_action_for(ar, "receive")
>>> analyses = ar.getAnalyses(full_objects=True)
```

Create a Worksheet and add the analyses:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> for analysis in analyses:
...     worksheet.addAnalysis(analysis)
```

Add control analyses:

```
>>> len(worksheet.addReferenceAnalyses(ref_sample, [Cu, Fe, Au]))
3
```

Since a reference analysis can only live inside a Worksheet, the initial state of the control is *assigned* by default:

```
>>> duplicates = worksheet.getReferenceAnalyses()
>>> map(api.get_workflow_status_of, duplicates)
['assigned', 'assigned', 'assigned']
```

4.82 Reference Analysis (Control) multi-verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisControlMultiVerify

4.82.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import getReviewHistory
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
```

(continues on next page)

(continued from previous page)

```
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> control_sample = api.create(supplier, "ReferenceSample", title="Control",
...                       ReferenceDefinition=control_def,
...                       Blank=False, ExpiryDate=date_future,
...                       ReferenceResults=control_refs)
```

4.82.2 Multiverify not allowed if multi-verification is not enabled

Enable self verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the control and submit:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(0)
>>> try_transition(control, "submit", "to_be_verified")
True
```

The status of control and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot multi verify the control because multi-verification is not set:

```
>>> isTransitionAllowed(control, "multi_verify")
False
>>> try_transition(control, "multi_verify", "to_be_verified")
False
```

(continues on next page)

(continued from previous page)

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

But I can verify:

```
>>> isTransitionAllowed(control, "verify")
True
>>> try_transition(control, "verify", "verified")
True
```

And the status of the control is now *verified*:

```
>>> api.get_workflow_status_of(control)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.82.3 Multiverify transition with multi-verification enabled

The system allows to set multiple verifiers, both at Setup or Analysis Service level. If set, the control will transition to verified when the total number of verifications equals to the value set in multiple-verifiers.

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Not allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_disabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the control and submit:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(12)
>>> try_transition(control, "submit", "to_be_verified")
True
```

The status of control and others is *to_be_verified*:

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

I cannot *verify*:

```
>>> isTransitionAllowed(control, "verify")
False
>>> try_transition(control, "verify", "verified")
False
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

Because multi-verification is enabled:

```
>>> bikasetup.getNumberOfRequiredVerifications()
3
```

And there are 3 verifications remaining:

```
>>> control.getNumberOfRemainingVerifications()
3
```

But I can multi-verify:

```
>>> isTransitionAllowed(control, "multi_verify")
True
>>> try_transition(control, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

And my user id is recorded as such:

```
>>> action = getReviewHistory(control)[0]
>>> action['actor'] == TEST_USER_ID
True
```

And now, there are two verifications remaining:

```
>>> control.getNumberOfRemainingVerifications()
2
```

So, I cannot verify yet:

```
>>> isTransitionAllowed(control, "verify")
False
>>> try_transition(control, "verify", "verified")
False
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

But I cannot multi-verify neither, cause I am the same user who did the last multi-verification:

```
>>> isTransitionAllowed(control, "multi_verify")
False
>>> try_transition(control, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

And the system is configured to not allow same user to verify multiple times:

```
>>> bikasetup.getTypeOfmultiVerification()
'self_multi_disabled'
```

But I can multi-verify if I change the type of multi-verification:

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
>>> isTransitionAllowed(control, "multi_verify")
True
>>> try_transition(control, "multi_verify", "to_be_verified")
True
```

The status remains to *to_be_verified*:

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

Since there is only one verification remaining, I cannot multi-verify again:

```
>>> control.getNumberOfRemainingVerifications()
1
>>> isTransitionAllowed(control, "multi_verify")
False
>>> try_transition(control, "multi_verify", "to_be_verified")
False
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

But now, I can verify:

```
>>> isTransitionAllowed(control, "verify")
True
>>> try_transition(control, "verify", "verified")
True
```

There is no verifications remaining:

```
>>> control.getNumberOfRemainingVerifications()
0
```

And the status of the control is now *verified*:

```
>>> api.get_workflow_status_of(control)
'verified'
```

While the rest remain in *to_be_verified* state because the regular analysis hasn't been verified yet:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

If we multi-verify the regular analysis (2+1 times):

```
>>> analysis = ar.getAnalyses(full_objects=True)[0]
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "multi_verify", "to_be_verified")
True
>>> try_transition(analysis, "verify", "verified")
True
```

The rest transition to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'verified'
>>> api.get_workflow_status_of(worksheet)
'verified'
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.82.4 Check permissions for Multi verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Set the number of required verifications to 3:

```
>>> bikasetup.setNumberOfRequiredVerifications(3)
```

Set the multi-verification to "Allow same user to verify multiple times":

```
>>> bikasetup.setTypeOfmultiVerification('self_multi_enabled')
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the control and submit:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(12)
>>> try_transition(control, "submit", "to_be_verified")
True
```

Exactly these roles can multi_verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", control)
['LabManager', 'Manager', 'Verifier']
```

Current user can multi_verify because has the *LabManager* role:

```
>>> isTransitionAllowed(control, "multi_verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(control, "multi_verify")
True
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(control, "multi_verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(control, "multi_verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(control, "multi_verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(control, "multi_verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.83 Reference Analysis (Controls) submission guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisControlSubmit

4.83.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
```

(continues on next page)

(continued from previous page)

```
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> control_sample = api.create(supplier, "ReferenceSample", title="Control",
...                     ReferenceDefinition=control_def,
...                     control=False, ExpiryDate=date_future,
...                     ReferenceResults=control_refs)
```

4.83.2 control submission basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get control analyses:

```
>>> controls = worksheet.getReferenceAnalyses()
>>> control_1 = controls[0]
>>> control_2 = controls[1]
>>> control_3 = controls[2]
```

Cannot submit a control without a result:

```
>>> try_transition(control_1, "submit", "to_be_verified")
False
```

Even if we try with an empty or None result:

```
>>> control_1.setResult('')
>>> try_transition(control_1, "submit", "to_be_verified")
False
```

```
>>> control_1.setResult(None)
>>> try_transition(control_1, "submit", "to_be_verified")
False
```

But will work if we try with a result of 0:

```
>>> control_1.setResult(0)
>>> try_transition(control_1, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(control_1)
'to_be_verified'
```

And we cannot re-submit a control that have been submitted already:

```
>>> try_transition(control_1, "submit", "to_be_verified")
False
```

4.83.3 Auto submission of a Worksheets when all its analyses are submitted

Create a Worksheet:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
```

Set results and submit all analyses from the worksheet except controls:

```
>>> for analysis in worksheet.getRegularAnalyses():
...     analysis.setResult(13)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getRegularAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

While the Analysis Request has been transitioned to *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The worksheet has not been transitioned:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

Because controls are still in *assigned* state:

```
>>> map(api.get_workflow_status_of, worksheet.getReferenceAnalyses())
['assigned', 'assigned', 'assigned']
```

If we set results and submit controls:

```
>>> for analysis in worksheet.getReferenceAnalyses():
...     analysis.setResult(0)
...     transitioned = do_action_for(analysis, "submit")
>>> map(api.get_workflow_status_of, worksheet.getReferenceAnalyses())
['to_be_verified', 'to_be_verified', 'to_be_verified']
```

The worksheet will automatically be submitted too:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

4.83.4 Submission of controls with interim fields set

Set interims to the analysis *Au*:

```
>>> Au.setInterimFields([
...     {"keyword": "interim_1", "title": "Interim 1",},
...     {"keyword": "interim_2", "title": "Interim 2",}])
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get control analyses:

```
>>> control = worksheet.getReferenceAnalyses()[0]
```

Cannot submit if no result is set:

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

But even if we set a result, we cannot submit because interims are missing:

```
>>> control.setResult(12)
>>> control.getResult()
'12'
```

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

So, if the control has interims defined, all them are required too:

```
>>> control.setInterimValue("interim_1", 15)
>>> control.getInterimValue("interim_1")
'15'
```

```
>>> control.getInterimValue("interim_2")
''
```

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

Even if we set a non-valid (None, empty) value to an interim:

```
>>> control.setInterimValue("interim_2", None)
>>> control.getInterimValue("interim_2")
''
```

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

```
>>> control.setInterimValue("interim_2", '')
>>> control.getInterimValue("interim_2")
''
```

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

But it will work if the value is 0:

```
>>> control.setInterimValue("interim_2", 0)
>>> control.getInterimValue("interim_2")
'0'
```

```
>>> try_transition(control, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

Might happen the other way round. We set interims but not a result:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

(continues on next page)

(continued from previous page)

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setInterimValue("interim_1", 10)
>>> control.setInterimValue("interim_2", 20)
>>> try_transition(control, "submit", "to_be_verified")
False
```

Still, the result is required:

```
>>> control.setResult(12)
>>> try_transition(control, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

4.83.5 Submission of control analysis with interim calculation

If a control analysis have a calculation assigned, the result will be calculated automatically based on the calculation. If the calculation have interims set, only those that do not have a default value set will be required.

Prepare the calculation and set the calculation to analysis *Au*:

```
>>> Au.setInterimFields([])
>>> calc = api.create(bikasetup.bika_calculations, 'Calculation', title='Test␣
↪Calculation')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1', 'value': 10}
>>> interim_2 = {'keyword': 'IT2', 'title': 'Interim 2', 'value': 2}
>>> interim_3 = {'keyword': 'IT3', 'title': 'Interim 3', 'value': ''}
>>> interim_4 = {'keyword': 'IT4', 'title': 'Interim 4', 'value': None}
>>> interim_5 = {'keyword': 'IT5', 'title': 'Interim 5'}
>>> interims = [interim_1, interim_2, interim_3, interim_4, interim_5]
>>> calc.setInterimFields(interims)
>>> calc.setFormula("[IT1]+[IT2]+[IT3]+[IT4]+[IT5]")
>>> Au.setCalculation(calc)
```

Create a Worksheet with control:

```
>>> ar = new_ar([Au])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
```

Cannot submit if no result is set

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> try_transition(control, "submit", "to_be_verified")
False
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to ‘api.analysis.calculate‘:

```
>>> control.setResult(34)
```

Set a value for interim IT5:

```
>>> control.setInterimValue("IT5", 5)
```

Cannot transition because IT3 and IT4 have None/empty values as default:

```
>>> try_transition(control, "submit", "to_be_verified")
False
```

Let's set a value for those interims:

```
>>> control.setInterimValue("IT3", 3)
>>> try_transition(control, "submit", "to_be_verified")
False
```

```
>>> control.setInterimValue("IT4", 4)
```

Since interims IT1 and IT2 have default values set, the analysis will submit:

```
>>> try_transition(control, "submit", "to_be_verified")
True
```

```
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

4.83.6 Submission of controls with dependencies

controls with dependencies are not allowed. controls can only be created from analyses without dependents.

TODO Might we consider to allow the creation of controls with dependencies?

Reset the interim fields for analysis *Au*:

```
>>> Au.setInterimFields([])
```

Prepare a calculation that depends on *Cu* and assign it to *Fe* analysis:

```
>>> calc_fe = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Fe')
>>> calc_fe.setFormula("[Cu]*10")
>>> Fe.setCalculation(calc_fe)
```

Prepare a calculation that depends on *Fe* and assign it to *Au* analysis:

```
>>> calc_au = api.create(bikasetup.bika_calculations, 'Calculation', title='Calc for␣
↪Au')
>>> interim_1 = {'keyword': 'IT1', 'title': 'Interim 1'}
>>> calc_au.setInterimFields([interim_1])
>>> calc_au.setFormula("([IT1]+[Fe])/2")
>>> Au.setCalculation(calc_au)
```

Create an Analysis Request:

```
>>> ar = new_ar([Cu, Fe, Au])
```

Create a Worksheet with control:

```
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> analyses = worksheet.getRegularAnalyses()
```

Only one control created for *Cu*, cause is the only analysis that does not have dependents:

```
>>> controls = worksheet.getReferenceAnalyses()
>>> len(controls) == 1
True
```

```
>>> control = controls[0]
>>> control.getKeyword()
'Cu'
```

TODO This should not be like this, but the calculation is performed by *ajaxCalculateAnalysisEntry*. The calculation logic must be moved to 'api.analysis.calculate':

```
>>> control.setResult(0)
```

Cannot submit routine *Fe* cause there is no result for routine analysis *Cu* and the control of *Cu* cannot be used as a dependent:

```
>>> fe_analysis = filter(lambda an: an.getKeyword()=="Fe", analyses)[0]
>>> try_transition(fe_analysis, "submit", "to_be_verified")
False
```

4.83.7 Check permissions for Submit transition

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Set a result:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(23)
```

Exactly these roles can submit:

```
>>> get_roles_for_permission("senaite.core: Edit Results", control)
['Analyst', 'LabManager', 'Manager']
```

And these roles can view results:

```
>>> get_roles_for_permission("senaite.core: View Results", control)
['Analyst', 'LabClerk', 'LabManager', 'Manager', 'RegulatoryInspector']
```

Current user can submit because has the *LabManager* role:

```
>>> isTransitionAllowed(control, "submit")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Authenticated', 'LabClerk', 'RegulatoryInspector
↪', 'Sampler'])
>>> isTransitionAllowed(control, "submit")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(control, "submit")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(control, "submit")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.84 Reference Analysis (Control) verification guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisControlVerify

4.84.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
```

(continues on next page)

(continued from previous page)

```
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
```

(continues on next page)

(continued from previous page)

```
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> control_sample = api.create(supplier, "ReferenceSample", title="Control",
...                       ReferenceDefinition=control_def,
...                       Blank=False, ExpiryDate=date_future,
...                       ReferenceResults=control_refs)
```

4.84.2 Control verification basic constraints

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the control and submit:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(0)
>>> try_transition(control, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

I cannot verify the control because I am the same user who submitted:

```
>>> try_transition(control, "verify", "verified")
False
>>> api.get_workflow_status_of(control)
'to_be_verified'
```

And I cannot verify the Worksheet, because it can only be verified once all analyses it contains are verified (and this is done automatically):

```
>>> try_transition(worksheet, "verify", "verified")
False
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

But if I enable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Then, I can verify my own result:

```
>>> try_transition(control, "verify", "verified")
True
```

And the worksheet transitions to *verified*:

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And we cannot re-verify a control that has been verified already:

```
>>> try_transition(control, "verify", "verified")
False
```

To ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.84.3 Check permissions for Verify transition

Enable self verification of results:

```
>>> bikasetup.setSelfVerificationEnabled(True)
>>> bikasetup.getSelfVerificationEnabled()
True
```

Create a Worksheet and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the control and submit:

```
>>> control = worksheet.getReferenceAnalyses()[0]
>>> control.setResult(12)
>>> try_transition(control, "submit", "to_be_verified")
True
```

Exactly these roles can verify:

```
>>> get_roles_for_permission("senaite.core: Transition: Verify", control)
['LabManager', 'Manager', 'Verifier']
```

Current user can verify because has the *LabManager* role:

```
>>> isTransitionAllowed(control, "verify")
True
```

Also if the user has the roles *Manager* or *Verifier*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(control, "verify")
True
```

(continues on next page)

(continued from previous page)

```
>>> setRoles(portal, TEST_USER_ID, ['Verifier',])
>>> isTransitionAllowed(control, "verify")
True
```

But cannot for other roles:

```
>>> setRoles(portal, TEST_USER_ID, ['Analyst', 'Authenticated', 'LabClerk'])
>>> isTransitionAllowed(control, "verify")
False
```

Even if is *Owner*

```
>>> setRoles(portal, TEST_USER_ID, ['Owner'])
>>> isTransitionAllowed(control, "verify")
False
```

And Clients cannot neither:

```
>>> setRoles(portal, TEST_USER_ID, ['Client'])
>>> isTransitionAllowed(control, "verify")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

And to ensure consistency amongst tests, we disable self-verification:

```
>>> bikasetup.setSelfVerificationEnabled(False)
>>> bikasetup.getSelfVerificationEnabled()
False
```

4.85 Reference Analysis retract guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowReferenceAnalysisRetract

4.85.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def to_new_worksheet_with_reference(ar, reference):
...     worksheet = api.create(portal.worksheets, "Worksheet")
...     service_uids = list()
...     for analysis in ar.getAnalyses(full_objects=True):
...         worksheet.addAnalysis(analysis)
...         service_uids.append(analysis.getServiceUID())
...     worksheet.addReferenceAnalyses(reference, service_uids)
...     return worksheet
```

```
>>> def submit_regular_analyses(worksheet):
...     for analysis in worksheet.getRegularAnalyses():
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def try_transition(object, transition_id, target_state_id):
...      success = do_action_for(object, transition_id)[0]
...      state = api.get_workflow_status_of(object)
...      return success and state == target_state_id
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> control_sample = api.create(supplier, "ReferenceSample", title="Control",
...                       ReferenceDefinition=control_def,
...                       Blank=False, ExpiryDate=date_future,
...                       ReferenceResults=control_refs)
```

4.85.2 Retract transition and guard basic constraints

Create an Analysis Request and submit regular analyses:

```
>>> ar = new_ar([Cu])
>>> worksheet = to_new_worksheet_with_reference(ar, control_sample)
>>> submit_regular_analyses(worksheet)
```

Get the reference and submit:

```
>>> reference = worksheet.getReferenceAnalyses()[0]
>>> reference.setResult(12)
>>> try_transition(reference, "submit", "to_be_verified")
True
>>> api.get_workflow_status_of(reference)
'to_be_verified'
```

(continues on next page)

(continued from previous page)

```
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

Retract the reference:

```
>>> try_transition(reference, "retract", "retracted")
True
>>> api.get_workflow_status_of(reference)
'retracted'
```

And one new additional reference has been added in *assigned* state:

```
>>> references = worksheet.getReferenceAnalyses()
>>> sorted(map(api.get_workflow_status_of, references))
['assigned', 'retracted']
```

And the Worksheet has been transitioned to *open*:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

While the Analysis Request is still in *to_be_verified*:

```
>>> api.get_workflow_status_of(ar)
'to_be_verified'
```

The new analysis is a copy of retracted one:

```
>>> retest = filter(lambda an: api.get_workflow_status_of(an) == "assigned",␣
↪references)[0]
>>> retest.getKeyword() == reference.getKeyword()
True
>>> retest.getReferenceAnalysesGroupID() == reference.getReferenceAnalysesGroupID()
True
>>> retest.getRetestOf() == reference
True
>>> reference.getRetest() == retest
True
>>> retest.getAnalysisService() == reference.getAnalysisService()
True
```

And keeps the same results as the retracted one:

```
>>> retest.getResult() == reference.getResult()
True
```

And is located in the same slot as well:

```
>>> worksheet.get_slot_position_for(reference) == worksheet.get_slot_position_
↪for(retest)
True
```

If I submit the result for the new reference:

```
>>> try_transition(retest, "submit", "to_be_verified")
True
```

The status of both the reference and the Worksheet is "to_be_verified":

```
>>> api.get_workflow_status_of(retest)
'to_be_verified'
>>> api.get_workflow_status_of(worksheet)
'to_be_verified'
```

And I can even retract the retest:

```
>>> try_transition(retest, "retract", "retracted")
True
>>> api.get_workflow_status_of(retest)
'retracted'
```

And one new additional reference has been added in *assigned* state:

```
>>> references = worksheet.getReferenceAnalyses()
>>> sorted(map(api.get_workflow_status_of, references))
['assigned', 'retracted', 'retracted']
```

And the Worksheet has been transitioned to *open*:

```
>>> api.get_workflow_status_of(worksheet)
'open'
```

4.86 Retract transition when reference analyses from same Reference Sample are added

When analyses from same Reference Sample are added in a worksheet, the worksheet allocates different slots for them, although each of the slots keeps the container the analysis belongs to (in this case the same Reference Sample). Hence, when retracting a reference analysis, the retest must be added in the same position as the original, regardless of how many reference analyses from same reference sample exist. Further information: https://github.com/senaite/senaite.core/pull/1179

Create an Analysis Request:

```
>>> ar = new_ar([Cu])
>>> worksheet = api.create(portal.worksheets, "Worksheet")
... for analysis in ar.getAnalyses(full_objects=True):
...     worksheet.addAnalysis(analysis)
```

Add same reference sample twice:

```
>>> ref_1 = worksheet.addReferenceAnalyses(control_sample, [api.get_uid(Cu)])[0]
>>> ref_2 = worksheet.addReferenceAnalyses(control_sample, [api.get_uid(Cu)])[0]
>>> ref_1 != ref_2
True
```

Get the reference analyses positions:

```
>>> ref_1_pos = worksheet.get_slot_position_for(ref_1)
>>> ref_1_pos
1
>>> ref_2_pos = worksheet.get_slot_position_for(ref_2)
```

(continues on next page)

(continued from previous page)

```
>>> ref_2_pos
2
```

Submit both:

```
>>> ref_1.setResult(12)
>>> ref_2.setResult(13)
>>> try_transition(ref_1, "submit", "to_be_verified")
True
>>> try_transition(ref_2, "submit", "to_be_verified")
True
```

Retract the first reference analysis. The retest has been added in same slot:

```
>>> try_transition(ref_1, "retract", "retracted")
True
>>> retest_1 = ref_1.getRetest()
>>> worksheet.get_slot_position_for(retest_1)
1
```

And the same if we retract the second reference analysis:

```
>>> try_transition(ref_2, "retract", "retracted")
True
>>> retest_2 = ref_2.getRetest()
>>> worksheet.get_slot_position_for(retest_2)
2
```

4.87 Worksheet auto-transitions

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowWorksheetAutotransitions

4.87.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from DateTime import DateTime
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
```

(continues on next page)

(continued from previous page)

```
...         'Contact': contact.UID(),
...         'DateSampled': DateTime(),
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     do_action_for(ar, "receive")
...     return ar
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.87.2 Retract transition and guard basic constraints

Create a Worksheet:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
```

The status of the worksheet is “open”:

```
>>> api.get_workflow_status_of(ws)
'open'
```

If we submit all analyses from the Worksheet except 1:

```
>>> analyses = ws.getAnalyses()
>>> for analysis in analyses[1:]:
...     analysis.setResult(12)
...     success = do_action_for(analysis, "submit")
```

The Worksheet remains in "open" status:

```
>>> api.get_workflow_status_of(ws)
'open'
```

If now we remove the remaining analysis:

```
>>> ws.removeAnalysis(analyses[0])
```

The Worksheet is submitted automatically because all analyses it contains have been submitted already:

```
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

If we add the analysis again:

```
>>> ws.addAnalysis(analyses[0])
```

The worksheet is rolled-back to open again:

```
>>> api.get_workflow_status_of(ws)
'open'
```

If we remove again the analysis and verify the rest:

```
>>> ws.removeAnalysis(analyses[0])
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

```
>>> setup.setSelfVerificationEnabled(True)
>>> for analysis in analyses[1:]:
...     success = do_action_for(analysis, "verify")
>>> setup.setSelfVerificationEnabled(False)
```

The worksheet is verified automatically too:

```
>>> api.get_workflow_status_of(ws)
'verified'
```

And we cannot add analyses anymore:

```
>>> ws.addAnalysis(analyses[0])
>>> api.get_workflow_status_of(ws)
'verified'
```

```
>>> not analyses[0].getWorksheet()
True
```

```
>>> analyses[0] in ws.getAnalyses()
False
```

4.88 Worksheet remove guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowWorksheetRemove

4.88.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from DateTime import DateTime
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': DateTime(),
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
...     do_action_for(ar, "receive")
...     return ar
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> setup = portal.bika_setup
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(setup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(setup.bika_labcontacts, "LabContact", Firstname="Lab",␣
↪Lastname="Manager")
>>> department = api.create(setup.bika_departments, "Department", title="Chemistry",␣
↪Manager=labcontact)
>>> category = api.create(setup.bika_analysiscategories, "AnalysisCategory", title=
↪"Metals", Department=department)
```

(continues on next page)

(continued from previous page)

```
>>> Cu = api.create(setup.bika_analysisservices, "AnalysisService", title="Copper",␣
↪Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(setup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(setup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

4.88.2 Retract transition and guard basic constraints

Create a Worksheet:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
```

The status of the worksheet is "open":

```
>>> api.get_workflow_status_of(ws)
'open'
```

And is not possible to remove unless empty:

```
>>> isTransitionAllowed(ws, "remove")
False
```

```
>>> for analysis in ws.getAnalyses():
...     success = do_action_for(analysis, "unassign")
>>> isTransitionAllowed(ws, "remove")
True
```

If we do "remove", the Worksheet object is deleted:

```
>>> container = ws.aq_parent
>>> len(container.objectValues("Worksheet"))
1
>>> success = do_action_for(ws, "remove")
>>> len(container.objectValues("Worksheet"))
0
```

Try now for all possible statuses:

```
>>> analyses = ar.getAnalyses(full_objects=True)
>>> cu = filter(lambda an: an.getKeyword() == "Cu", analyses)[0]
>>> fe = filter(lambda an: an.getKeyword() == "Fe", analyses)[0]
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> ws.addAnalysis(cu)
>>> cu.setResult(12)
>>> success = do_action_for(cu, "submit")
```

For *to_be_verified* status:

```
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

(continues on next page)

(continued from previous page)

```
>>> isTransitionAllowed(ws, "remove")
False
```

For *rejected* status:

```
>>> success = do_action_for(ws, "reject")
>>> api.get_workflow_status_of(ws)
'rejected'
>>> isTransitionAllowed(ws, "remove")
False
```

For *verified* status:

```
>>> setup.setSelfVerificationEnabled(True)
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> ws.addAnalysis(fe)
>>> fe.setResult(12)
>>> success = do_action_for(fe, "submit")
>>> verified = do_action_for(fe, "verify")
>>> api.get_workflow_status_of(ws)
'verified'
>>> isTransitionAllowed(ws, "remove")
False
>>> setup.setSelfVerificationEnabled(False)
```

4.88.3 Check permissions for Remove transition

Create an empty Worksheet:

```
>>> ws = api.create(portal.worksheets, "Worksheet")
```

The status of the Worksheet is *open*:

```
>>> api.get_workflow_status_of(ws)
'open'
```

Exactly these roles can remove:

```
>>> get_roles_for_permission("senaite.core: Transition: Remove Worksheet", ws)
['LabManager', 'Manager']
```

Current user can remove because has the *LabManager* role:

```
>>> isTransitionAllowed(ws, "remove")
True
```

Also if the user has the role *Manager*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(ws, "remove")
True
```

But cannot for other roles:

```
>>> other_roles = ['Analyst', 'Authenticated', 'LabClerk', 'Verifier']
>>> setRoles(portal, TEST_USER_ID, other_roles)
>>> isTransitionAllowed(ws, "remove")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.89 Worksheet retract guard and event

Running this test from the buildout directory:

bin/test test_textual_doctests -t WorkflowWorksheetRetract

4.89.1 Test Setup

Needed Imports:

```
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
>>> from bika.lims.workflow import doActionFor as do_action_for
>>> from bika.lims.workflow import isTransitionAllowed
>>> from DateTime import DateTime
>>> from plone.app.testing import setRoles
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def timestamp(format="%Y-%m-%d"):
...     return DateTime().strftime(format)
```

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

```
>>> def new_ar(services):
...     values = {
...         'Client': client.UID(),
...         'Contact': contact.UID(),
...         'DateSampled': date_now,
...         'SampleType': sampletype.UID()}
...     service_uids = map(api.get_uid, services)
...     ar = create_analysisrequest(client, request, values, service_uids)
```

(continues on next page)

(continued from previous page)

```
...     transitioned = do_action_for(ar, "receive")
...     return ar
```

```
>>> def submit_analyses(ar):
...     for analysis in ar.getAnalyses(full_objects=True):
...         analysis.setResult(13)
...         do_action_for(analysis, "submit")
```

```
>>> def get_roles_for_permission(permission, context):
...     allowed = set(rolesForPermissionOn(permission, context))
...     return sorted(allowed)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> blank_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",
↪ title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': api.get_uid(Cu), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Fe), 'result': '0', 'min': '0', 'max': '0'},
...               {'uid': api.get_uid(Au), 'result': '0', 'min': '0', 'max': '0'},]
>>> blank_def.setReferenceResults(blank_refs)
>>> control_def = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': api.get_uid(Cu), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Fe), 'result': '10', 'min': '0', 'max': '0'},
...                 {'uid': api.get_uid(Au), 'result': '15', 'min': '14.5', 'max':
↪'15.5'},]
>>> control_def.setReferenceResults(control_refs)
>>> blank = api.create(supplier, "ReferenceSample", title="Blank",
...                    ReferenceDefinition=blank_def,
```

(continues on next page)

(continued from previous page)

```
...                     Blank=True, ExpiryDate=date_future,
...                     ReferenceResults=blank_refs)
>>> control = api.create(supplier, "ReferenceSample", title="Control",
...                      ReferenceDefinition=control_def,
...                      Blank=False, ExpiryDate=date_future,
...                      ReferenceResults=control_refs)
```

4.89.2 Retract transition and guard basic constraints

Create a Worksheet:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
```

The status of the worksheet is "open":

```
>>> api.get_workflow_status_of(ws)
'open'
```

And is not possible to retract when status is "open":

```
>>> isTransitionAllowed(ws, "retract")
False
```

But is possible to retract if the status is "to_be_verified":

```
>>> submit_analyses(ar)
>>> list(set(map(api.get_workflow_status_of, ws.getAnalyses())))
['to_be_verified']
>>> api.get_workflow_status_of(ws)
'to_be_verified'
>>> isTransitionAllowed(ws, "retract")
True
```

The retraction of the worksheet causes all its analyses to be retracted:

```
>>> do_action_for(ws, "retract")
(True, '')
>>> analyses = ws.getAnalyses()
>>> len(analyses)
6
>>> sorted(map(api.get_workflow_status_of, analyses))
['assigned', 'assigned', 'assigned', 'retracted', 'retracted', 'retracted']
```

And the Worksheet transitions to "open":

```
>>> api.get_workflow_status_of(ws)
'open'
```

With duplicates and reference analyses, the system behaves the same way:

```
>>> dups = ws.addDuplicateAnalyses(1)
>>> blanks = ws.addReferenceAnalyses(blank, [Cu.UID(), Fe.UID(), Au.UID()])
>>> controls = ws.addReferenceAnalyses(control, [Cu.UID(), Fe.UID(), Au.UID()])
>>> len(ws.getAnalyses())
15
>>> for analysis in ws.getAnalyses():
...     analysis.setResult(10)
...     success = do_action_for(analysis, "submit")
>>> analyses = ws.getAnalyses()
>>> sorted(set(map(api.get_workflow_status_of, analyses)))
['retracted', 'to_be_verified']
```

Since all non-retracted analyses have been submitted, the worksheet status is *to_be_verified*:

```
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

The Worksheet can be retracted:

```
>>> isTransitionAllowed(ws, "retract")
True
>>> do_action_for(ws, "retract")
(True, '')
>>> analyses = ws.getAnalyses()
>>> len(analyses)
27
>>> statuses = map(api.get_workflow_status_of, analyses)
>>> len(filter(lambda st: st == "assigned", statuses))
12
>>> len(filter(lambda st: st == "retracted", statuses))
15
```

And the worksheet transitions to “open”:

```
>>> api.get_workflow_status_of(ws)
'open'
```

4.89.3 Check permissions for Retract transition

Create a Worksheet and submit results:

```
>>> ar = new_ar([Cu, Fe, Au])
>>> ws = api.create(portal.worksheets, "Worksheet")
>>> for analysis in ar.getAnalyses(full_objects=True):
...     ws.addAnalysis(analysis)
>>> submit_analyses(ar)
```

The status of the Worksheet and its analyses is *to_be_verified*:

```
>>> api.get_workflow_status_of(ws)
'to_be_verified'
```

```
>>> analyses = ws.getAnalyses()
>>> list(set(map(api.get_workflow_status_of, analyses)))
['to_be_verified']
```

Exactly these roles can retract:

```
>>> get_roles_for_permission("senaite.core: Transition: Retract", ws)
['LabManager', 'Manager']
```

Current user can verify because has the *LabManager* role:

```
>>> isTransitionAllowed(ws, "retract")
True
```

Also if the user has the role *Manager*:

```
>>> setRoles(portal, TEST_USER_ID, ['Manager',])
>>> isTransitionAllowed(ws, "retract")
True
```

But cannot for other roles:

```
>>> other_roles = ['Analyst', 'Authenticated', 'LabClerk', 'Verifier']
>>> setRoles(portal, TEST_USER_ID, other_roles)
>>> isTransitionAllowed(ws, "retract")
False
```

Reset the roles for current user:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
```

4.90 Worksheet - Apply Worksheet Template

Worksheets are the main artifact for planning tests in the laboratory. They are also used to add reference samples (controls and blancs), duplicates and aggregate related tests from different Analysis Requests to be processed in a single run.

Although worksheets can be created manually by the labmanager each time is required, a better approach is to create them by using Worksheet Templates. In a Worksheet Template, the labman defines the layout, the number of slots and the type of analyses (reference or routine) to be placed in each slot, as well as the Method and Instrument to be assigned. Thus, Worksheet Templates are used for the semi-automated creation of Worksheets.

This doctest will validate the consistency between the Worksheet and the Worksheet Template used for its creation. It will also test the correctness of the worksheet when applying a Worksheet Template in a manually created Worksheet.

4.91 Test Setup

Running this test from the buildout directory:

bin/test -t WorksheetApplyTemplate

Needed Imports:

```
>>> import re
>>> from AccessControl.PermissionRole import rolesForPermissionOn
>>> from bika.lims import api
>>> from bika.lims.content.analysisrequest import AnalysisRequest
>>> from bika.lims.utils.analysisrequest import create_analysisrequest
```

(continues on next page)

(continued from previous page)

```
>>> from bika.lims.utils import tmpID
>>> from bika.lims.workflow import doActionFor
>>> from bika.lims.workflow import getCurrentState
>>> from bika.lims.workflow import getAllowedTransitions
>>> from DateTime import DateTime
>>> from plone.app.testing import TEST_USER_ID
>>> from plone.app.testing import TEST_USER_PASSWORD
>>> from plone.app.testing import setRoles
```

Functional Helpers:

```
>>> def start_server():
...     from Testing.ZopeTestCase.utils import startZServer
...     ip, port = startZServer()
...     return "http://{}:{}/{}".format(ip, port, portal.id)
```

Variables:

```
>>> portal = self.portal
>>> request = self.request
>>> bikasetup = portal.bika_setup
>>> date_now = DateTime().strftime("%Y-%m-%d")
>>> date_future = (DateTime() + 5).strftime("%Y-%m-%d")
```

We need to create some basic objects for the test:

```
>>> setRoles(portal, TEST_USER_ID, ['LabManager',])
>>> client = api.create(portal.clients, "Client", Name="Happy Hills", ClientID="HH",␣
↪MemberDiscountApplies=True)
>>> contact = api.create(client, "Contact", Firstname="Rita", Lastname="Mohale")
>>> sampletype = api.create(bikasetup.bika_sampletypes, "SampleType", title="Water",␣
↪Prefix="W")
>>> labcontact = api.create(bikasetup.bika_labcontacts, "LabContact", Firstname="Lab",
↪ Lastname="Manager")
>>> department = api.create(bikasetup.bika_departments, "Department", title="Chemistry
↪", Manager=labcontact)
>>> category = api.create(bikasetup.bika_analysiscategories, "AnalysisCategory",␣
↪title="Metals", Department=department)
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", Name="Naralabs")
>>> Cu = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Copper
↪", Keyword="Cu", Price="15", Category=category.UID(), Accredited=True)
>>> Fe = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Iron",␣
↪Keyword="Fe", Price="10", Category=category.UID())
>>> Au = api.create(bikasetup.bika_analysisservices, "AnalysisService", title="Gold",␣
↪Keyword="Au", Price="20", Category=category.UID())
```

Create some Analysis Requests, so we can use them as sources for Worksheet cration:

```
>>> values = {
...     'Client': client.UID(),
...     'Contact': contact.UID(),
...     'DateSampled': date_now,
...     'SampleType': sampletype.UID()}
>>> service_uids = [Cu.UID(), Fe.UID(), Au.UID()]
>>> ar0 = create_analysisrequest(client, request, values, service_uids)
>>> ar1 = create_analysisrequest(client, request, values, service_uids)
```

(continues on next page)

(continued from previous page)

```
>>> ar2 = create_analysisrequest(client, request, values, service_uids)
>>> ar3 = create_analysisrequest(client, request, values, service_uids)
>>> ar4 = create_analysisrequest(client, request, values, service_uids)
>>> ar5 = create_analysisrequest(client, request, values, service_uids)
>>> ar6 = create_analysisrequest(client, request, values, service_uids)
>>> ar7 = create_analysisrequest(client, request, values, service_uids)
>>> ar8 = create_analysisrequest(client, request, values, service_uids)
>>> ar9 = create_analysisrequest(client, request, values, service_uids)
```

4.91.1 Worksheet Template creation

Create a Worksheet Template, but for *Cu* and *Fe* analyses, with the following layout with 7 slots:

- Routine analyses in slots 1, 2, 4
- Duplicate analysis from slot 1 in slot 3
- Duplicate analysis from slot 4 in slot 5
- Control analysis in slot 6
- Blank analysis in slot 7

```
>>> service_uids = [Cu.UID(), Fe.UID()]
>>> layout = [
...     {'pos': '1', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '2', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '3', 'type': 'd',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': '1'},
...     {'pos': '4', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '5', 'type': 'd',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': '4'},
...     {'pos': '6', 'type': 'c',
...      'blank_ref': '',
...      'control_ref': 'jajsjas',
...      'dup': ''},
...     {'pos': '7', 'type': 'b',
...      'blank_ref': 'asasasa',
...      'control_ref': '',
...      'dup': ''},
... ]
>>> template = api.create(bikasetup.bika_worksheettemplates, "WorksheetTemplate",␣
↪title="WS Template Test", Layout=layout, Service=service_uids)
```

4.92 Apply Worksheet Template to a Worksheet

Create a new Worksheet by using this worksheet template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(template)
```

Since we haven't received any analysis requests, this worksheet remains empty:

```
>>> worksheet.getAnalyses()
[]
>>> worksheet.getLayout()
[]
```

Receive the Analysis Requests and apply again the Worksheet Template:

```
>>> performed = doActionFor(ar0, 'receive')
>>> performed = doActionFor(ar1, 'receive')
>>> performed = doActionFor(ar2, 'receive')
>>> performed = doActionFor(ar3, 'receive')
>>> performed = doActionFor(ar4, 'receive')
>>> performed = doActionFor(ar5, 'receive')
>>> performed = doActionFor(ar6, 'receive')
>>> performed = doActionFor(ar7, 'receive')
>>> performed = doActionFor(ar8, 'receive')
>>> performed = doActionFor(ar9, 'receive')
>>> worksheet.applyWorksheetTemplate(template)
```

Slots 1, 2 and 4 are filled with routine analyses:

```
>>> worksheet.get_slot_positions(type='a')
[1, 2, 4]
```

Each slot occupied by routine analyses is assigned to an Analysis Request, so each time we add an analysis, it will be added into it's corresponding slot:

```
>>> container = worksheet.get_container_at(1)
>>> container.UID() == ar0.UID()
True
```

```
>>> slot1_analyses = worksheet.get_analyses_at(1)
>>> an_ar = list(set([an.getRequestUID() for an in slot1_analyses]))
```

```
>>> len(an_ar) == 1
True
```

```
>>> an_ar[0] == ar0.UID()
True
```

```
>>> [an.getKeyword() for an in slot1_analyses]
['Cu', 'Fe']
```

Slots 3 and 5 are filled with duplicate analyses:

```
>>> worksheet.get_slot_positions(type='d')
[3, 5]
```

```
>>> dup1 = worksheet.get_analyses_at(3)
>>> len(dup1) == 2
True
```

```
>>> list(set([dup.portal_type for dup in dup1]))
['DuplicateAnalysis']
```

The first duplicate analysis located at slot 3 is a duplicate of the first analysis from slot 1:

```
>>> dup_an = dup1[0].getAnalysis()
>>> slot1_analyses[0].UID() == dup_an.UID()
True
```

But since we haven't created any reference analysis (neither blank or control), slots reserved for blank and controls are not occupied:

```
>>> worksheet.get_slot_positions(type='c')
[]
>>> worksheet.get_slot_positions(type='b')
[]
```

4.93 Remove analyses and Apply Worksheet Template again

Remove analyses located at position 2:

```
>>> to_del = worksheet.get_analyses_at(2)
>>> worksheet.removeAnalysis(to_del[0])
>>> worksheet.removeAnalysis(to_del[1])
```

Only slots 1, 4 are filled with routine analyses now:

```
>>> worksheet.get_slot_positions(type='a')
[1, 4]
```

Modify the Worksheet Template to allow *Au* analysis and apply the template to the same Worksheet again:

```
>>> service_uids = [Cu.UID(), Fe.UID(), Au.UID()]
>>> template.setService(service_uids)
>>> worksheet.applyWorksheetTemplate(template)
```

Now, slot 2 is filled again:

```
>>> worksheet.get_slot_positions(type='a')
[1, 2, 4]
```

And each slot contains the additional analysis *Au*:

```
>>> slot1_analyses = worksheet.get_analyses_at(1)
>>> len(slot1_analyses) == 3
True
```

```
>>> an_ar = list(set([an.getRequestUID() for an in slot1_analyses]))
>>> an_ar[0] == ar0.UID()
True
```

```
>>> [an.getKeyword() for an in slot1_analyses]
['Cu', 'Fe', 'Au']
```

As well as in duplicate analyses:

```
>>> dup1 = worksheet.get_analyses_at(3)
>>> len(dup1) == 3
True
```

```
>>> slot3_analyses = worksheet.get_analyses_at(3)
>>> [an.getKeyword() for an in slot3_analyses]
['Cu', 'Fe', 'Au']
```

4.94 Remove a duplicate and add it manually

Remove all duplicate analyses from slot 5:

```
>>> dup5 = worksheet.get_analyses_at(5)
>>> len(dup5) == 3
True
```

```
>>> worksheet.removeAnalysis(dup5[0])
>>> worksheet.removeAnalysis(dup5[1])
>>> worksheet.removeAnalysis(dup5[2])
>>> dup5 = worksheet.get_analyses_at(5)
>>> len(dup5) == 0
True
```

Add duplicates using the same source routine analysis, located at slot 4, but manually instead of applying the Worksheet Template:

```
>>> dups = worksheet.addDuplicateAnalyses(4)
```

Three duplicate have been added to the worksheet:

```
>>> [dup.getKeyword() for dup in dups]
['Cu', 'Fe', 'Au']
```

And these duplicates have been added in the slot number 5, cause this slot is where this duplicate fits better in accordance with the layout defined in the worksheet template associated to this worksheet:

```
>>> dup5 = worksheet.get_analyses_at(5)
>>> [dup.getKeyword() for dup in dup5]
['Cu', 'Fe', 'Au']
```

```
>>> dups_uids = [dup.UID() for dup in dups]
>>> dup5_uids = [dup.UID() for dup in dup5]
>>> [dup for dup in dup5_uids if dup not in dups_uids]
[]
```

But if we remove only one duplicate analysis from slot number 5:

```
>>> worksheet.removeAnalysis(dup5[0])
>>> dup5 = worksheet.get_analyses_at(5)
>>> [dup.getKeyword() for dup in dup5]
['Fe', 'Au']
```

And we manually add duplicates for analysis in position 4, a new slot will be added at the end of the worksheet (slot number 8), cause the slot number 5 is already occupied and slots 6 and 7, although empty, are reserved for blank and control:

```
>>> worksheet.get_analyses_at(8)
[]
```

```
>>> dups = worksheet.addDuplicateAnalyses(4)
>>> [dup.getKeyword() for dup in dups]
['Cu', 'Fe', 'Au']
```

```
>>> dup8 = worksheet.get_analyses_at(8)
>>> [dup.getKeyword() for dup in dup8]
['Cu', 'Fe', 'Au']
```

```
>>> dups_uids = [dup.UID() for dup in dups]
>>> dup8_uids = [dup.UID() for dup in dup8]
>>> [dup for dup in dup8_uids if dup not in dups_uids]
[]
```

4.95 Control and blanks with Worksheet Template

First, create a Reference Definition for blank:

```
>>> blankdef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition",␣
↪title="Blank definition", Blank=True)
>>> blank_refs = [{'uid': Cu.UID(), 'result': '0', 'min': '0', 'max': '0', 'error': '0
↪'},
...               {'uid': Fe.UID(), 'result': '0', 'min': '0', 'max': '0', 'error': '0
↪'},]
>>> blankdef.setReferenceResults(blank_refs)
```

And for control:

```
>>> controldef = api.create(bikasetup.bika_referencedefinitions, "ReferenceDefinition
↪", title="Control definition")
>>> control_refs = [{'uid': Cu.UID(), 'result': '10', 'min': '0.9', 'max': '10.1',
↪'error': '0.1'},
...                 {'uid': Fe.UID(), 'result': '10', 'min': '0.9', 'max': '10.1',
↪'error': '0.1'},]
>>> controldef.setReferenceResults(control_refs)
```

Then, we create the associated Reference Samples:

```
>>> blank = api.create(supplier, "ReferenceSample", title="Blank",
...                    ReferenceDefinition=blankdef,
...                    Blank=True, ExpiryDate=date_future,
...                    ReferenceResults=blank_refs)
```

(continues on next page)

(continued from previous page)

```
>>> control = api.create(supplier, "ReferenceSample", title="Control",
...                      ReferenceDefinition=controldef,
...                      Blank=False, ExpiryDate=date_future,
...                      ReferenceResults=control_refs)
```

Apply the blank and control to the Worksheet Template layout:

```
>>> layout = template.getLayout()
>>> layout[5] = {'pos': '6', 'type': 'c',
...              'blank_ref': '',
...              'control_ref': controldef.UID(),
...              'dup': ''}
>>> layout[6] = {'pos': '7', 'type': 'b',
...              'blank_ref': blankdef.UID(),
...              'control_ref': '',
...              'dup': ''}
>>> template.setLayout(layout)
```

Apply the worksheet template again:

```
>>> worksheet.applyWorksheetTemplate(template)
```

Blank analyses at slot number 7, but note the reference definition is only for analyses *Cu* and *Fe*:

```
>>> ans = worksheet.get_analyses_at(7)
>>> [an.getKeyword() for an in ans]
['Cu', 'Fe']
>>> list(set([an.getReferenceType() for an in ans]))
['b']
```

Control analyses at slot number 6:

```
>>> ans = worksheet.get_analyses_at(6)
>>> [an.getKeyword() for an in ans]
['Cu', 'Fe']
>>> list(set([an.getReferenceType() for an in ans]))
['c']
```

4.96 Remove Reference Analyses and add them manually

Remove all controls from slot 6:

```
>>> ans6 = worksheet.get_analyses_at(6)
>>> len(ans6)
2
```

```
>>> worksheet.removeAnalysis(ans6[0])
>>> worksheet.removeAnalysis(ans6[1])
>>> worksheet.get_analyses_at(6)
[]
```

Add a reference analysis, but manually:

```
>>> ref_ans = worksheet.addReferenceAnalyses(control, [Fe.UID(), Cu.UID()])
>>> [ref.getKeyword() for ref in ref_ans]
['Cu', 'Fe']
```

These reference analyses have been added in the slot number 6, cause this slot is where these reference analyses fit better in accordance with the layout defined in the worksheet template associated to this worksheet:

```
>>> ref6 = worksheet.get_analyses_at(6)
>>> [ref.getKeyword() for ref in ref6]
['Cu', 'Fe']
```

```
>>> refs_uids = [ref.UID() for ref in ref_ans]
>>> ref6_uids = [ref.UID() for ref in ref6]
>>> [ref for ref in ref6_uids if ref not in refs_uids]
[]
```

But if we remove only one reference analysis from slot number 6:

```
>>> worksheet.removeAnalysis(ref6[0])
>>> ref6 = worksheet.get_analyses_at(6)
>>> [ref.getKeyword() for ref in ref6]
['Fe']
```

And we manually add references, a new slot will be added at the end of the worksheet (slot number 8), cause the slot number 6 is already occupied, as well as the rest of the slots:

```
>>> worksheet.get_analyses_at(9)
[]
```

```
>>> ref_ans = worksheet.addReferenceAnalyses(control, [Fe.UID(), Cu.UID()])
>>> [ref.getKeyword() for ref in ref_ans]
['Cu', 'Fe']
```

```
>>> ref9 = worksheet.get_analyses_at(9)
>>> [ref.getKeyword() for ref in ref9]
['Cu', 'Fe']
```

```
>>> refs_uids = [ref.UID() for ref in ref_ans]
>>> ref9_uids = [ref.UID() for ref in ref9]
>>> [ref for ref in ref9_uids if ref not in refs_uids]
[]
```

Reject any remaining analyses awaiting for assignment:

```
>>> query = {"portal_type": "Analysis", "review_state": "unassigned"}
>>> objs = map(api.get_object, api.search(query, "bika_analysis_catalog"))
>>> sucess = map(lambda obj: doActionFor(obj, "reject"), objs)
```

4.97 WorksheetTemplate assignment to a non-empty Worksheet

Worksheet Template can also be used when the worksheet is not empty. The template has slots available for routine analyses in positions 1, 2 and 4:

```
>>> layout = template.getLayout()
>>> slots = filter(lambda p: p["type"] == "a", layout)
>>> sorted(map(lambda p: int(p.get("pos")), slots))
[1, 2, 4]
```

Create 3 samples with 'Cu' analyses:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(3))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Create a worksheet and apply the template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(template)
```

The Sample from first slot contains 1 analysis only (Cu):

```
>>> first = worksheet.get_container_at(1)
>>> first_analyses = worksheet.get_analyses_at(1)
>>> len(first_analyses)
1
```

```
>>> first_analyses[0].getKeyword()
'Cu'
```

```
>>> first_analyses[0].getRequest() == first
True
```

Add "Fe" analysis to the sample from first slot and re-assign the worksheet:

```
>>> cu = first.getAnalyses(full_objects=True)[0]
>>> first.setAnalyses([cu, Fe])
>>> worksheet.applyWorksheetTemplate(template)
```

The first slot, booked for the first Sample, contains now 'Fe':

```
>>> first_analyses = worksheet.get_analyses_at(1)
>>> len(first_analyses)
2
```

```
>>> map(lambda a: a.getKeyword(), first_analyses)
['Cu', 'Fe']
```

```
>>> map(lambda a: a.getRequest() == first, first_analyses)
[True, True]
```

Add "Fe" analysis to the third Sample (slot #4) and re-assign the worksheet:

```
>>> third = worksheet.get_container_at(4)
>>> cu = third.getAnalyses(full_objects=True)[0]
>>> third.setAnalyses([cu, Fe])
>>> worksheet.applyWorksheetTemplate(template)
```

The fourth slot contains now 'Fe' too:

```
>>> third_analyses = worksheet.get_analyses_at(4)
>>> len(third_analyses)
2
```

```
>>> map(lambda a: a.getKeyword(), third_analyses)
['Cu', 'Fe']
```

```
>>> map(lambda a: a.getRequest() == third, third_analyses)
[True, True]
```

Create now 5 more samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(3))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

And reassign the template to the worksheet:

```
>>> worksheet.applyWorksheetTemplate(template)
```

None of these new samples have been added:

```
>>> new_samp_uids = map(api.get_uid, samples)
>>> container_uids = map(lambda l: l["container_uid"], worksheet.getLayout())
>>> [u for u in new_samp_uids if u in container_uids]
[]
```

Add "Fe" analysis to the second Sample and re-assign the worksheet:

```
>>> second = worksheet.get_container_at(2)
>>> cu = second.getAnalyses(full_objects=True)[0]
>>> second.setAnalyses([cu, Fe])
>>> worksheet.applyWorksheetTemplate(template)
```

The second slot contains now 'Fe' too:

```
>>> second_analyses = worksheet.get_analyses_at(2)
>>> len(second_analyses)
2
```

```
>>> map(lambda a: a.getKeyword(), second_analyses)
['Cu', 'Fe']
```

```
>>> map(lambda a: a.getRequest() == second, second_analyses)
[True, True]
```

While none of the analyses from new samples have been added:

```
>>> container_uids = map(lambda l: l["container_uid"], worksheet.getLayout())
>>> [u for u in new_samp_uids if u in container_uids]
[]
```

Reject any remaining analyses awaiting for assignment:

```
>>> query = {"portal_type": "Analysis", "review_state": "unassigned"}
>>> objs = map(api.get_object, api.search(query, "bika_analysis_catalog"))
>>> sucess = map(lambda obj: doActionFor(obj, "reject"), objs)
```

4.98 WorksheetTemplate assignment keeps Sample natural order

Analyses are grabbed by using their priority sort key, but samples are sorted in natural order in the slots.

Create and receive 3 samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(3))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Create a worksheet and apply the template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(template)
```

Slots follows the natural order of the samples:

```
>>> map(lambda s: worksheet.get_slot_position(s), samples)
[1, 2, 4]
```

4.99 Assignment of a WorksheetTemplate with no services

Create a Worksheet Template without services assigned:

```
>>> service_uids = []
>>> layout = [
...     {'pos': '1', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '2', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
... ]
>>> empty_template = api.create(bikasetup.bika_worksheettemplates, "WorksheetTemplate
↪", title="WS Template Empty Test", Layout=layout, Service=service_uids)
```

Create and receive 2 samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(2))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Create a Worksheet and assign the template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(empty_template)
```

Worksheet remains empty:

```
>>> worksheet.getAnalyses()
[]
```

4.100 Assignment of Worksheet Template with Instrument

When a Worksheet Template has an instrument assigned, only analyses that can be performed with that same instrument are added in the worksheet.

Create a new Instrument:

```
>>> instr_type = api.create(bikasetup.bika_instrumenttypes, "InstrumentType", title=
↪"Temp instrument type")
>>> manufacturer = api.create(bikasetup.bika_manufacturers, "Manufacturer", title=
↪"Temp manufacturer")
>>> supplier = api.create(bikasetup.bika_suppliers, "Supplier", title="Temp supplier")
>>> instrument = api.create(bikasetup.bika_instruments,
...                         "Instrument",
...                         title="Temp Instrument",
...                         Manufacturer=manufacturer,
...                         Supplier=supplier,
...                         InstrumentType=instr_type)
```

Create a Worksheet Template and assign the instrument:

```
>>> service_uids = [Cu]
>>> layout = [
...     {'pos': '1', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '2', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
... ]
>>> instr_template = api.create(bikasetup.bika_worksheettemplates,
...                             "WorksheetTemplate",
...                             title="WS Template with instrument",
...                             Layout=layout,
...                             Instrument=instrument,
...                             Service=service_uids)
```

Create and receive 2 samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(2))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Create a Worksheet and assign the template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(instr_template)
```

Worksheet remains empty because the instrument is not allowed for *Cu* service:

```
>>> worksheet.getAnalyses()
[]
```

Assign the Instrument to the *Cu* service:

```
>>> Cu.setInstrumentEntryOfResults(True)
>>> Cu.setInstruments([instrument,])
```

Re-assign the worksheet template:

```
>>> worksheet.applyWorksheetTemplate(instr_template)
```

Worksheet still remains empty, because the analyses were created before the assignment of Instrument to the the *Cu* service:

```
>>> worksheet.getAnalyses()
[]
```

Create a 2 more samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(2))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Re-assign the worksheet template and the worksheet now contains the two analyses from the new samples we've created:

```
>>> worksheet.applyWorksheetTemplate(instr_template)
>>> ws_analyses = worksheet.getAnalyses()
>>> len(ws_analyses)
2
```

```
>>> all(map(lambda a: a.getRequest() in samples, ws_analyses))
True
```

Unassign instrument from *Cu* service:

```
>>> Cu.setInstrumentEntryOfResults(False)
>>> Cu.setInstruments([])
```

Reject any remaining analyses awaiting for assignment:

```
>>> query = {"portal_type": "Analysis", "review_state": "unassigned"}
>>> objs = map(api.get_object, api.search(query, "bika_analysis_catalog"))
>>> success = map(lambda obj: doActionFor(obj, "reject"), objs)
```

4.101 Assignment of Worksheet Template with Method

When a Worksheet Template has a method assigned, only analyses that can be performed with that same method are added in the worksheet.

Create a new Method:

```
>>> method = api.create(portal.methods, "Method", title="Temp method")
```

Create a Worksheet Template and assign the method:

```
>>> service_uids = [Cu]
>>> layout = [
...     {'pos': '1', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
...     {'pos': '2', 'type': 'a',
...      'blank_ref': '',
...      'control_ref': '',
...      'dup': ''},
... ]
>>> method_template = api.create(bikasetup.bika_worksheettemplates,
...                              "WorksheetTemplate",
...                              title="WS Template with instrument",
...                              Layout=layout,
...                              RestrictToMethod=method,
...                              Service=service_uids)
```

Create and receive 2 samples:

```
>>> service_uids = [Cu]
>>> samples = map(lambda i: create_analysisrequest(client, request, values, service_
↪uids), range(2))
>>> success = map(lambda s: doActionFor(s, "receive"), samples)
```

Create a Worksheet and assign the template:

```
>>> worksheet = api.create(portal.worksheets, "Worksheet")
>>> worksheet.applyWorksheetTemplate(method_template)
```

Worksheet remains empty because the method is not allowed for *Cu* service:

```
>>> worksheet.getAnalyses()
[]
```

Assign the Method to the *Cu* service:

```
>>> Cu.setMethods([method, ])
```

Re-assign the worksheet template:

```
>>> worksheet.applyWorksheetTemplate(method_template)
```

The worksheet now contains the two analyses:

```
>>> worksheet.applyWorksheetTemplate(method_template)
>>> ws_analyses = worksheet.getAnalyses()
>>> len(ws_analyses)
2
```

```
>>> all(map(lambda a: a.getRequest() in samples, ws_analyses))
True
```

Unassign method from *Cu* service:

```
>>> Cu.setMethods([])
```

Reject any remaining analyses awaiting for assignment:

```
>>> query = {"portal_type": "Analysis", "review_state": "unassigned"}
>>> objs = map(api.get_object, api.search(query, "bika_analysis_catalog"))
>>> success = map(lambda obj: doActionFor(obj, "reject"), objs)
```

CHAPTER 5

Release notes

5.1 Update from 1.3.x to 2.0.0rc1

IMPORTANT: This is a release candidate. Do not use for production

!!! This update requires at least Plone 5.2.1 and Python 2.x. !!!

Please update your *buildout.cfg* configuration file to this version and re-run the *buildout* script.

You can use the Plone unified installer to install Plone 5.x first: https://github.com/plone/Installers-UnifiedInstaller

Please make sure to pin *senaite.lims* to version *2.0.0rc1* to get the correct versions of all dependent SENAITE packages.

Run first the Plone upgrade before you continue to update SENAITE to version 2.x.

5.2 Update from 1.3.0 to 1.3.1

IMPORTANT: Plan the upgrade with enough time

This update might take long depending on the number of objects registered in the system:

- Stale Sample and Partition objects have been removed from ZODB https://github.com/senaite/senaite.core/pull/1351
- Full Audit log has been added to Senaite https://github.com/senaite/senaite.core/pull/1324

If you have your own add-on, please review the changes to check beforehand if some parts of your add-on require modifications. Worth to mention that the following tips are strongly recommended before proceeding with the upgrade:

- Do a zeopack
- Do a backup of both your code and database
- Try to have as much analyses in verified/published statuses as possible
- Stop unnecessary applications and services that may consume RAM
- Start with a clean log file

5.3 Update from 1.2.9 to 1.3.0

IMPORTANT: Plan the upgrade with enough time

Version 1.3.0 is not a hotfix release, it rather comes with a lot of changes that require additional care when planning the update. If you have your own add-on, please review the changes to check beforehand if some parts of your add-on require modifications.

This update will take long (up to 5h for instances with large amounts of data). Therefore, is recommended to plan and allocate enough resources for the process to complete beforehand. For big databases, RAM is a critical factor to be considered before upgrading. Worth to mention that the following tips are strongly recommended before proceeding with the upgrade:

- Do a zeopack
- Do a backup of both your code and database
- Try to have as much analyses in verified/published statuses as possible
- Stop unnecessary applications and services that may consume RAM
- Start with a clean log file

Most of the base code has been refactored keeping in mind the following objectives:

- Less complexity: less code, better code
- High test coverage: lower chance of undetected bugs
- Boost performance: better experience, with no delays
- Improve security: rely on Zope's security policies
- Code responsibility: focus on core functionalities and let other add-ons to deal with the rest (*senaite.lims*, *senaite.core.listing*, etc.)

Besides of this refactoring, this version also comes with a myriad of new functionalities and enhancements: full-fledged sample partitions, reinvented listings and results entry, new adapters for extensibility, etc.

Version 1.3 is the result of hard, but exciting work at same time. Four months of walking through valleys of tears and fighting hydras. Four exciting months to be proud of.

5.4 Update from 1.2.8 to 1.2.9

IMPORTANT: Plan the upgrade with enough time This update might take long depending on the number of Analyses, Analysis Requests and Samples registered in the system:

- Role mappings updated for Analaysis Requests and Samples (rejection) https://github.com/senaite/senaite.core/pull/1041
- Recatalog of invalidated/retest Analysis Requests (invalidation) https://github.com/senaite/senaite.core/pull/1027
- Reindex and recatalog of getDueDate for Analysis Requests https://github.com/senaite/senaite.core/pull/1051
- Reindex of getDueDate for Analyses: https://github.com/senaite/senaite.core/pull/1032
- Workflow: *retract_ar* transition has been renamed to *invalidate* https://github.com/senaite/senaite.core/pull/1027

5.5 Update from 1.2.7 to 1.2.8

- Operators for min and max values have been added. For specifications already present in the system, the result ranges are considered as bounded and closed: *[min,max] = {result | min <= result <= max}*. https://github.com/senaite/senaite.core/pull/965

5.6 Update from 1.2.4 to 1.2.5

- This update requires the execution of *bin/buildout*, because Products.TextIndexNG3 has been added. It will help to search by wildcards in TextIndexNG3 indexes instead of looking for the keyword inside wildcards. For now, it is used only in AR listing catalog. https://pypi.python.org/pypi/Products.TextIndexNG3/
- This update might take long depending on the number of Analyses registered in the system, because the upgrade step will walk through all analyses in order to update those that do not have a valid (non-floatable) duplicate variation value (see #768).

5.7 Update from 1.2.3 to 1.2.4

- This update requires the execution of *bin/buildout*, because WeasyPrint has been updated to version 0.42.2: http://weasyprint.readthedocs.io/en/stable/changelog.html#version-0-42-2

5.8 Update from 1.2.2 to 1.2.3

- IMPORTANT: Plan the upgrade with enough time This update might take long depending on the number of Analysis Requests registered in the system because a new index and column 'assigned_state' has been added in Analysis Requests catalog, that require the catalog to be reindexed (see #637).

5.9 Update from 1.2.1 to 1.2.2

- IMPORTANT: Plan the upgrade with enough time This update might take long depending on the number of Batches registered in the system, because an index from their catalog needs to be reindexed (#574). Also, a new index that affects the Worksheets that have a Worksheet Template assigned has been added and needs to be indexed.

5.10 Update from 1.2.0 to 1.2.1

- This update requires the execution of *bin/buildout*, because a new dependency has been added: Plone Subrequest
- With this update, Analyses Services that are inactive, but have active dependent services, will be automatically transitioned to *active* state. This procedure fixes eventual inconsistencies amongst the statuses of Analyses Services. See #555

CHAPTER 6

Changelog

6.1 2.0.0rc3 (unreleased)

- #1666 Added adapter to extend listing_searchable_text index
- #1665 Display Auditlog listing icon
- #1664 Display correct icons in listings
- #1662 Custom view/edit forms for dexterity types
- #1660 Cleanup unused ajax endpoints for reports and js
- #1659 Fix language in datepicker widgets

6.2 2.0.0rc2 (2020-10-13)

- #1657 Allow to edit Profiles in Samples for pre verified/published states
- #1655 Rename service's "Result Options" and "Additional Values"
- #1655 Move service's "Additional values" to "Result Options" tab
- #1654 Fix Text of interim choices is not displayed correctly on readonly mode
- #1653 Fix Maximum length for Choices field from Interim fields is set to 40
- #1650 Fix Error when invalidating a sample with contained retests
- #1646 Allow multi-select in results entry
- #1645 Allow translation of path bar items
- #1643 Setup View Filter
- #1642 Allow multi-choice in results entry
- #1640 Fix AttributeError on Worksheet Template assignment

- #1638 Fix "Published results" tab is not displayed to Client contacts
- #1637 Fix "Page not Found" Error for migrated SENAITE Contents with File/Image Fields
- #1635 Sidebar toggle
- #1632 Reorganize JS/CSS modules
- #1626 Fix assignment of analyses via worksheet template when Worksheet is full
- #1620 Add Results Interpretation Templates
- #1621 Fix instrument import for analyses with result options
- #1618 Better style for DX form based field errors
- #1616 Fix writing instrument methods on read when reindexing services
- #1613 Compatibility with Plone 5.2.2

6.3 2.0.0rc1 (2020-07-24)

- Compatibility with *Plone 5.x* on *Python 2.x*
- User Interface updated to *Bootstrap 4.x*
- Integrated *Webpack* for resource management
- Added *senaite.core* namespace package
- Added global *SenaiteTheme* view
- Integrated SVG icons
- New install screens

6.4 1.3.4 (2020-08-11)

Added

- #1609 Support result options entry for interim values
- #1598 Added "modified" index in Sample's (AnalysisRequest) catalog
- #1596 Allow to hide actions menu by using new marker interface IHideActionsMenu
- #1588 Dynamic Analysis Specs: Lookup dynamic spec only when the specification is set
- #1586 Allow to configure the variables for IDServer with an Adapter
- #1584 Date (yymmdd) support in IDs generation
- #1582 Allow to retest analyses without the need of retraction
- #1573 Append the type name of the current record in breadcrumbs (Client)
- #1573 Add link "My Organization" under top-right user selection list

Changed

- #1607 Allow to set instruments from method edit view
- #1588 Dynamic Analysis Specs: Hide compliance viewlets
- #1579 Remove classic mode in folderitems

- #1577 Do not force available workflow transitions in batches listing
- #1573 Do not display top-level "Clients" folder to non-lab users

Fixed

- #1606 Fix Traceback on Dynamic Analysis Specs Edit
- #1605 Fix Retests are not displayed in Worksheet's print view
- #1604 Fix Analyses from partitions do not show up when using Worksheet Template
- #1602 Fix Report "Analysis per Service" is always creating the same PDF file
- #1601 Fix Wrong url in client's sample templates listing
- #1594 Fix System does not validate values from Results Options to be different
- #1596 Fix Reports page shows the Display/State/Add menu
- #1595 Fix Wrong url in client's analyses profiles listing
- #1593 Fix Out-of-range alert icon is shown to users w/o "View Results" privileges
- #1592 Fix Publisher user cannot publish samples
- #1591 Fix User can assign a contact from another client while creating a Sample
- #1585 Fix wrong label and description for *ShowPartitions* setting from setup
- #1583 Fix traceback in services listing in ARTemplate view
- #1581 Fix Some values are not properly rendered in services listing
- #1580 Fix Analysts are not displayed once created in worksheets listing
- #1575 Fix Uncertainties are displayed although result is below Detection Limit
- #1572 Fix Unable to get the previous status when duplicated in review history
- #1570 Fix Date time picker does not translates well to current language
- #1571 Fix Cannot reject Sample when contact has no email set
- #1568 Fix Traceback when rendering sticker *Code_39_2ix1i*
- #1567 Fix missing CCContact after adding a new Sample
- #1566 Fix column sorting in Worksheet listing
- #1563 Fix Client Contacts can create Samples without Contact

6.5 1.3.3.1 (2020-03-04)

Fixed

- #1560 Fix missing Add Dynamic Analysis Specifications Button for Lab Managers

6.6 1.3.3 (2020-03-03)

Added

- #1553 Allow to modify the email template for rejection notification
- #1549 Added registry profile for jQuery UI settings

- #1544 Progress indicator for Batch listing
- #1536 Integrated Setup and Profiles from senaite.lims
- #1534 Integrate browser resources from senaite.lims
- #1529 Moved contentmenu provider into core
- #1523 Moved Installation Screens into core
- #1520 JavaScripts/CSS Integration and Cleanup
- #1517 Integrate senaite.core.spotlight
- #1516 Consider analyses with result options or string in duplicate valid range
- #1515 Moved Setup View into Core
- #1506 Specification non-compliant viewlet in Sample
- #1506 Sample results ranges out-of-date viewlet in Sample
- #1506 Warn icon in analyses when range is not compliant with Specification
- #1492 Dynamic Analysis Specifications
- #1507 Support for semi-colon character separator in CCEmails field
- #1499 Moved navigation portlet into core
- #1498 Moved all viewlets from senaite.lims to senaite.core
- #1505 Display partition link in analyses listing
- #1491 Enable Audit-logging for Dexterity Contents
- #1489 Support Multiple Catalogs for Dexterity Contents
- #1481 Filter Templates field when Sample Type is selected in Sample Add form
- #1483 Added Accredited symbol in Analyses listings
- #1466 Support for “readonly” and “hidden” visibility modes in ReferenceWidget

Changed

- #1555 List all multi-reports for samples, where the current sample is contained
- #1543 Sort navigation child-nodes alphabetically
- #1539 Avoid unnecessary Price recalculations in Sample Add Form
- #1532 Updated jQuery Barcode to version 2.2.0
- #1513 Better Ajax Loader for Sample Add Form
- #1508 Do not try to render InstrumentQCFailuresViewlet to non-lab personnel
- #1495 Better Remarks handling and display
- #1502 Improved DateTime Widget
- #1490 Support Dexterity Behavior Fields in API
- #1488 Support Dexterity Contents in Catalog Indexers
- #1486 Clean-up of indexes and metadata from *setup_catalog*

Removed

- #1531 Remove sampling rounds from core

- #1551 Removed dependency to plone.app.iterate
- #1530 Removed ARImport
- #1530 Removed stale type registrations
- #1541 Remove add/edit options of ReferenceWidget
- #1535 Remove *zcatalog* monkey (and *getRequestUID* index)
- #1518 Removed stale indexes from *analysis_catalog*
- #1516 Removed getResultsRange metadata from analysis_catalog
- #1487 Dexterity Compatible Catalog Base Class
- #1482 Remove *senaite.instruments* dependency for instrument import form
- #1478 Remove AcquireFieldDefaults (was used for CCEmails field only)

Fixed

- #1556 Fix TypeError when retracting analyses with ExtendedField
- #1552 Rejection on registration is neither generating rejection pdf nor email
- #1550 Fix Uncaught TypeError in combogrid
- #1542 Fix sporadical errors when contacts do not have a valid email address
- #1540 Fix flushing CCEmail fields in Sample Add Form
- #1533 Fix traceback from log when rendering stickers preview
- #1525 Fix error when creating partitions with analyst user
- #1522 Fix sporadical timeout issue when adding new samples/remarks
- #1506 Changes via manage results don't get applied to partitions
- #1506 Fix recursion error when getting dependencies through Calculation
- #1506 setter from ARAnalysisField does no longer return values
- #1512 QC Analyses listing appears empty in Sample view
- #1510 Error when viewing a Sample w/o Batch as client contact
- #1511 Links to partitions for Internal Use are displayed in partitions viewlet
- #1505 Manage Analyses Form re-applies partitioned Analyses back to the Root
- #1503 Avoid duplicate CSS IDs in multi-column Add form
- #1501 Fix Attribute Error in Reference Sample Popup
- #1493 jsonapi.read omits *include_methods* when a single parameter is used
- #1494 Fix KeyError in Sample Type Listing
- #1477 Sample edit form - some selection widgets empty
- #1478 Clients default CC E-Mails missing in Add Sample
- #1479 Fixed too many redirects error: Labclerks viewing verified worksheets
- #1480 Worksheet removal results in 404
- #1475 User with "Analyst" role cannot submit analyses from worksheet
- #1474 Adding Control Reference to Worksheet causes print fail

- #1473 Hidden settings of analysis services lost on Sample creation
- #1472 Secondary samples - removal of analysis profile not possible
- #1469 Fix Site Properties Generic Setup Export Step
- #1467 Cannot override behavior of Batch folder when using *before_render*

6.7 1.3.2 (2019-10-30)

Added

- #1463 Structure Export/Import Handlers for Generic Setup
- #1462 Allow to extend the behavior of fields from AddSample view with adapters
- #1455 Added support for adapters in guard handler
- #1436 Setting in setup for auto-reception of samples upon creation
- #1433 Added Submitter column in Sample's analyses listing
- #1441 Added Auto ID Behavior for Dexterity Contents
- #1422 Notify user with failing addresses when emailing of results reports
- #1420 Allow to detach a partition from its primary sample
- #1410 Email API

Changed

- #1451 Render Analysis Remarks in Listings as HTML
- #1445 Allow formatted HTML in the other rejection reasons
- #1428 Publish verified partitions
- #1429 Add2: Do not set template values on already filled fields
- #1427 Improved performance of Sample header table rendering
- #1417 Cache allowed transitions for analyses on the request
- #1413 Improved Email Publication

Removed

- #1449 Removed InheritedObjects (Inherited from) field from Batch
- #1430 Removed Identifier Types

Fixed

- #1462 Autofill Client Contact in Sample Add form when current user is a client
- #1461 Allow unassign transition for cancelled/rejected/retracted analyses
- #1449 sort_limit was not considered in ReferenceWidget searches
- #1449 Fix Clients were unable to add batches
- #1453 Fix initial IDs not starting with 1
- #1454 Fix occasional error when labeling samples w/o report as printed
- #1452 Fix missing error percentage calculation for reference samples

- #1447 New Client contact has access to last client's Sample only
- #1446 Parameter *group* in *contact._addUserToGroup* was not considered
- #1444 Fixed Worksheet autofill of wide Iterims
- #1443 Fix non-saving checkbox values for manual Interims in Analysis Services
- #1439 Fix global Auditlog when Analyses/Attachments were removed
- #1426 Render HTML Texts in Info Popups correctly
- #1423 Use the value set for ui_item property when displaying ReferenceWidget
- #1425 Fix adapter priority for widget visibility
- #1421 Fix Search Query for Batches Listing
- #1418 Subscriber adapters not supported in clients listing
- #1419 Mixed permissions for transitions in client workflow
- #1414 Occasional "OSError: [Errno 24] Too many open files" in frontpage

6.8 1.3.1 (2019-07-01)

Added

- #1401 Allow capture of text results
- #1391 Samples for internal use (lab personnel) only
- #1384 Added missing Html Field to ARReport
- #1369 Add getter to access the title of the sample condition directly
- #1347 Consider laboratory workdays only for the late analyses calculation
- #1324 Audit Log

Changed

- #1392 Hide partitions to clients when "Show Partitions" is not selected
- #1371 Allow sample publication without sending Email
- #1355 Make api.getId to also consider id metadata column (not only getId)
- #1352 Make timeit to not display args by default
- #1330 Make guards to not rely on review history
- #1339 Make Forumula a required field on Calculation

Removed

- #1359 Remove stale Sample/Partition objects from ZODB
- #1362 Remove unnecessary code from worksheet listing (bad performance)
- #1346 Remove Searchable Text Overrides
- #1328 Remove transition filtering in Worksheet listings

Fixed

- #1404 Avoid conflict errors during email publication

- #1403 Also consider the detached states as cancellable
- #1397 Fix Worksheet does not show the contained analyses
- #1395 Make Action Handler Pool Thread-Safe
- #1389 Analysts and Labclerks cannot create worksheets
- #1386 No auto-rejection of Sample when rejection reasons are set in Add form
- #1382 Fix double publication of the same sample when using multi-reports
- #1368 Fix WF state propagation on partition verification
- #1367 Clients can see interim values of analyses not yet verified
- #1361 Fix leap sample ID sequence after secondary sample
- #1344 Handle inline images in Results Interpretation
- #1336 Fix result capture date inconsistency
- #1334 Number of analyses are not updated after modifying analyses in a Sample
- #1319 Make api.get_review_history to always return a list
- #1317 Fix Analysis Service URL in Info Popup
- #1316 Barcodes view does not render all labels once Samples are registered
- #1341 Moved Agilent instruments from core to senaite.instruments
- #1356 Fixed selection on Analysis Spec on AR
- #1353 Fixed saving of PublicationSpecification on AR
- #1376 Fixed ft120.py to properly import winescan ft120 CSV files

6.9 1.3.0 (2019-03-30)

Added

- #1310 Support for radio copy in Sample Add view
- #1309 Added Samples rejection view
- #1291 "Remove" transition for empty Worksheets
- #1259 Added Facscalibur instrument import interface
- #1244 Added "Body for Sample Invalidation email" field in setup
- #1231 Add Client ID Column in Batch Listing
- #1230 Add Client ID Column in Sample Listing
- #1222 Added User and Security API
- #1217 Added filtering buttons in Analyses listings (Valid, Invalid, All)
- #1193 Added viewlets for partition and primary ARs
- #1180 Analysis Request field-specific permissions managed in *ar_workflow*
- #1154 Default to "Active" Worksheets in listing
- #1153 Progress bar in Worksheet listing

- #1120 Listing: Confirm before transition
- #1077 Creation of retests for blanks and controls via retraction
- #1077 Creation of retests for duplicates via retraction
- #1077 Auto-retraction of dependents on retract transitions
- #1077 The removal of a routine analysis causes the removal of its duplicates
- #1077 Added *rollback_to_received* transition in *ar_workflow*
- #1077 Added *rollback_to_open* transition in *worksheet_workflow*
- #1077 Battery of doctests for *referenceanalysis_workflow*
- #1077 Battery of doctests for *duplicateanalysis_workflow*
- #1077 Battery of doctests for *analysis_workflow*
- #1066 Enhanced partitioning system (partition magic)

Changed

- #873 Generalize the assignment of values into fields for Setup on import
- #1257 Fix Traceback for MultiSelectionWidgets in View Mode for UIDReferenceFields
- #1249 Render attachments in report in worksheets too
- #1243 ID Server Suffix Support for Retested ARs
- #1240 Support action-specific *workflow_action* requests with named adapters
- #1215 Do not copy CaptureDate and Result in retest analyses when created
- #1215 Do not modify the ID of analysis on retraction
- #1207 Make use of adapters for instrument auto-import
- #1206 Make use of adapters for instrument import/export interfaces
- #1203 Remove explicit definition of transitions in AR listing
- #1192 Integrate Container and Preservation in Partition Magic
- #1180 Analysis Request default ID Format becomes {sampleType}-{seq:04d}
- #1180 *visibility* attr behavior (AR fields) in favour of field-specific perms
- #1180 Sanitized *ar_workflow* regarding to guards, transitions and permissions
- #1180 Sample content type discarded in favour of Analysis Request
- #1182 Allow open min/max values in analysis specifications
- #1000 Refactor service calculation dependency/dependants functionality to API
- #1176 Unbind *cancellation_workflow* from AnalysisRequest content type
- #1173 Improve Resultsinterpretation Form
- #1161 Listing: Transposed worksheet improvements
- #1150 Completeness of not yet published Analysis Requests is not 100%
- #1147 Set empty option selected by default in result options
- #1148 Add “All” filter in Analysis Requests listings
- #1148 Make “Active” filter to display ongoing Analysis Requests only

- #1136 Skip objects w/o transitions in allowed transitions calculation
- #1135 Listing: Separate Remarks Toggle-Handle
- #1128 Listing: Removed non-conform handling of disabled fields
- #1123 Listing: Handle visibility of selected rows
- #1117 Removed *attachment_due* state and transition from analysis workflow
- #1114 Listing integration for Worksheet Templates
- #1109 Unassignment of an analysis causes the removal of its duplicates
- #1077 Rejection of an analysis causes the removal of its duplicates
- #1077 Don't allow to cancel Analysis Requests with assigned/submitted analyses
- #1077 Decouple *cancellation_workflow* from Analysis content type
- #1077 Refactored *referenceanalysis_workflow* + after transitions and guards
- #1077 Refactored *duplicateanalysis_workflow* + after transitions and guards
- #1077 Refactored *analysis_workflow* + after transitions and guards
- #1095 New worksheet results listing
- #1091 New Worksheet blank/control/duplicate listings
- #1093 Listing integration for Analysis Specification Widget
- #1092 Listing integration for Profile Analyses Widget
- #1081 API functions improved
- #1076 Instrument QC Viewlet Availability
- #1071 Reinvented Listing Tables
- #1066 Set default page size for listings to 50
- #1063 Permission for ar_add changed to "AddAnalysisRequest"
- #1064 Python 2.x is not supported by WeasyPrint v43. Pinned version: 0.42.3

Removed

- #1308 Remove install screen
- #1224 Replace publication engine with SENAITE IMPRESS
- #1207 Remove results auto-import interval from Setup: no limitations to cron
- #1180 Remove AdHoc field from Analysis Request
- #1180 Remove support for "sampleId" and "sample" keywords in ID Formatting
- #1180 Remove Sample views and accesses to Sample content types
- #1180 Remove Sample Partitions classic functionality
- #1167 Remove filtering by department
- #1149 Remove "Unpublished" filter from Analysis Requests listing
- #1132 Remove "Submitted by current user" icon from AR listing (performance)
- #1125 Remove Sample views, listings and links to Sample(s) from everywhere
- #1118 Removed all legacy Bika Listing / Advanced Filtering from Codebase

- #1077 Remove Sample-specific states from analysis workflow
- #1077 Remove *worksheetanalysis_workflow*
- #1059 Remove updates alert viewlet
- #1060 Remove classic portlets
- #1058 Remove gpw dependency
- #1058 Remove broken Quality Control reports
- #1057 Remove z3c.unconfigure dependency
- #1056 Remove collective.taskqueue dependency
- #808 Remove old AR Add code

Fixed

- #1109 Linking LabContacts to LDAP-user not possible
- #1283 Retracting a calculated Analysis leads to an inconsistent state
- #1281 Adding Analyses to an existing Worksheet fails
- #1269 Render analysis remarks conditionally
- #1277 Traceback in Manage Analyses
- #1245 Not all clients are shown in clients drop menu for Productivity Reports
- #1239 Fix and Improve Stickers
- #1214 Disallow entry of analysis results if the sample is not yet received
- #1213 Fix instrument notification display in Manage Results View
- #1212 Fix typo in SamplingFieldsVisibility
- #1191 Some worksheets pre-1.3 with published analyses remain in open/to_be_verified state
- #1190 Fixed evolution chart for reference analyses
- #1183 Fix results calculation of dependent calculations
- #1175 Fixed Version Display of SENAITE CORE Add-on in the Quickinstaller Tool
- #1142 Fix instrument QC Analyses Table
- #1137 Fixed and refactored log view
- #1124 Traceback when invalidating an Analysis Request with retracted analyses
- #1090 Primary AR does not recognize created Partitions
- #1089 Deepcopy Service Interims to Analyses
- #1082 Worksheet folder listing fixtures for direct analyst assignment
- #1080 Improve searchability of Client and Multifile fields
- #1072 Calculations with dependents do not work after 1.2.9 update
- #1070 Cannot get the allowed transitions (guard_sample_prep_transition)
- #1065 Creation of reflex rules does not work with senaite.lims add-on

Security

- #896 Users without privileges can create reports

- #1258 Fix widget permissions for Specs/Profiles/Templates Widgets
- #1237 Global Permission and Role Mappings refactoring
- #1077 Transitions and states strongly bound to DC Workflow + guards security

6.10 1.2.9 (2018-10-08)

Added

- #1051 Show the Due date in late's image tooltip in Analysis Requests listings
- #1048 Allow to set the pagesize in listings and show total number of results
- #1031 Added profiling and timing decorators
- #1001 Option to show Interim fields on results reports
- #1024 Function to get the Verifiers from an Analysis Request
- #1019 Support for min and max warns in range charts
- #1003 Alphanumeric numbering in sequential IDs generator

Changed

- #1050 Added Late filter button to analysisrequests listing
- #1046 Show "Date Registered" instead of "Date Created" in Analysis Requests listings
- #1044 State of analyses in retests is set to *received* by default (was *to_be_verified*)
- #1042 Function api.get_object() supports UID as input param
- #1036 Manage Analyses: Check permission of the AR to decide if it is frozen
- #764 Code cleanup and redux of 2-Dimensional-CSV instrument interface
- #1032 Refactored and fixed inconsistencies with Analysis TAT logic
- #1027 Refactored relationship between invalidated ARs and retests
- #1027 Rename *retract_ar* transition to *invalidate*
- #1012 Refactored Contacts listing
- #1010 Increased max length of Results options to 255 chars (was 40)
- #899 Sample's Date Received editable only when *received* analyses exist

Removed

- #1232 Remove *uniquefieldvalidator* for Client Names
- #1026 Removed auto-digest of results reports on verify transitions
- #1005 Removed databasesanitize package
- #992 Removed "Attach" report option for Attachments

Fixed

- #1216 Allow manual entry (if granted) of results if instrument is invalid
- #1051 Analyses Requests w/o submitted results always appear as not late
- #1047 Fix translate utility function

- #1049 Secondary Analysis Request changes received date of Sample
- #1041 Reject transition is available to Client once AR/Sample is received
- #1043 Invalid AR Retested informative message is not prominent enough
- #1039 Detection limit criteria from retracted analysis is preserved
- #1037 Display supplier view instead of reference samples per default
- #1030 Earliness of analysis is not expressed as minutes
- #1029 TAT in Analysis TAT over time report does not display days
- #1029 TAT in Analysis TAT over time report with decimals
- #1029 Need to always choose an analyst in productivity reports
- #1034 Attachments assigned to Analyses break and get orphaned when the referenced Analysis was removed
- #1028 Numbers for productivity report "Analyses by client" are all zero
- #1022 Date Received saved as UTC time
- #1018 Fix AR Add cleanup after template removal
- #1014 ReferenceWidget does not handle searches with null/None
- #1008 Previous results from same batch are always displayed in reports
- #1013 ARs and Samples from other clients are listed when logged in as contact
- #991 New client contacts do not have access to their own AR Templates
- #996 Hide checkbox labels on category expansion
- #990 Fix client analysisspecs view
- #888 Order of Interim Fields not maintained on ARs

6.11 1.2.8 (2018-08-11)

Added

- #965 Added operators for max and min values in Specifications
- #947 Instrument import interface: Cobas Integra 400plus
- #924 Added ExtProxyField for its use in SchemaExtender

Changed

- #971 Refactored Client's Analysis Requests, Samples and Batches listings
- #945 Show AR Remarks in Worksheet ManageResults views
- #953 Refactored Analysis Categories Listing
- #956 Refactored LabContacts Listing
- #955 Refactored Departments Listing
- #954 Refactored Attachment Types Listing
- #944 Remarks style in Manage Results/Analyses
- #943 AnalysisRequest View Remarks Field Style

- #938 Refactored Analysis Profiles Widget
- #937 Refactored Analysis Specifications Widget
- #936 Refactored AR Templates Listing
- #933 Refactored SampleConditions Listing
- #932 Refactored Calculation Listing
- #931 Refactored AnalysisSpecs Listing
- #935 Refactored SamplingDeviations Listing
- #926 Refactored Analysis Services Listing
- #916 Refactored Instruments Listing
- #919 Refactored Profiles Listing
- #915 Refactored SamplePoints Listing
- #914 Refactored Sampletypes Listing
- #913 Refactored Methods Listing View

Removed

- #972 Remove "Linked Sample" from Sample
- #912 Remove "Default AR Specifications" Selection from Setup
- #901 Remove explicit permission settings for clients
- #900 Removed basic handling of custom Sample Preparation Workflows

Fixed

- #983 Traceback in Client's Analysis Specs view
- #986 Result input fields are not read-only for analyst after submission
- #985 Do not display content actions in listings from inside Client
- #966 Traceback in Analyses listings when analysis unit is a numeric value
- #959 Time not displayed for Date Created in Analysis Requests listings
- #949 Retain AR Spec if Analyses were added/removed
- #948 Inactive Sample Types shown in Analysis Specifications
- #940 Label "Date Received" appears twice in Analysis Request view
- #917 Localization of date and time strings in listings
- #902 Attribute error when updating QC results using an import interface
- #456 Date Published appears two times on the header table of AR view
- #898 Cannot view/edit Supplier. Tabs for different views now visible.
- #905 Users created through LabContact's Login Details view are added to "Clients" group
- #906 DateTime Widget does not display the Time
- #909 List of clients cannot sort by Client ID
- #921 Missing interim fields in worksheet/analyses_transposed view
- #920 Refactored Remarks and created RemarksField and RemarksWidget

- #958 Traceback on batch book view
- #960 Traceback on AnalysisSpec Log
- #962 Calculated results not marked for submission if zero
- #964 Dormant Analysis Services displayed in AR Templates
- #967 Avoid deepcopy, "Can't pickle acquisition wrappers"

6.12 1.2.7 (2018-07-10)

Added

- #836 Allow (Multi-)Attachment upload available in AR Add
- #846 Transifex integration
- #848 Show icon on the Supply Order View
- #844 Missing interface for AR Report added
- #858 Only Lab Managers sees rejected analysis requests

Changed

- #891 Better default styles for listing tables
- #887 New icon set
- #879 Upgrade lxml version from 2.3.6 to 3.6.0 and Plone from 4.3.15 to 4.3.17
- #873 Sample Type field editable in AR and Sample edit views before receive
- #868 AR Add Form: Refactoring and Styling
- #817 Make warn message clearer if transition rejection is due to missing sampler

Fixed

- #892 Display only active Analyses for new Profiles
- #889 Fix override order of message catalogs
- #864 Sort order in setup of analysis services wrong
- #881 Fixed JS i18n catalog names
- #880 Fix message factory
- #878 Fix AR Header Table Styles and Ajax Failures
- #877 Worksheet's attachments column is empty after results import from file
- #857 "other" reasons are not listed on AR rejection notifications (e-mail and attached pdf)
- #875 Fix Batch AR View
- #872 Date format appears wrong in Users history administrative report
- #855 Dashboard is displayed to Lab clerks after login only
- #871 Fix OpenTagError for i18ndude
- #865 AR VAT Amount when using Profiles is not calculated correctly
- #851 Fix worksheet verification with retracted results

6.13 1.2.6 (2018-06-08)

Changed

- #838 Unpinned WeasyPrint to allow Addon to use newer version
- #820 Always allow interim fields to be added to Analysis Services
- #826 Display signatures of verifiers instead of dept managers in results report
- #814 Change naming from Bika LIMS Configuration to LIMS Configuration in the Site Setup page
- #814 Change naming from Bika Setup to Setup in the LIMS Configuration section found in the Site Setup page

Fixed

- #842 Re-compiled Coffee-Scripts with version 1.12.7
- #824 Instrument Listing Views Fixes and Refactoring
- #840 Fix date range filter for "Data entry day book" report
- #828 Traceback when removing a retracted analysis through Manage Analyses view
- #832 Set new calculation Interims to dependant services
- #833 Fix sort order of interims in Calculations and Analysis Services
- #834 Fix Duplication Action for Analysis Services
- #835 List only published ARs when "Published" Filter is active
- #825 Error when exporting Worksheets list with *senaite.exporter*
- #821 Cannot retract single analysis services

6.14 1.2.5 (2018-05-05)

Added

- #777 Delay option for searches in reference widget combos
- #806 Include Client ID when setting up ARReport on the IDServer

Changed

- #815 Change description and title of the invalidation notification option

Removed

- #811 Remove senaite.api import in printform
- #807 Remove "queued_ars" JS call to avoid 404
- #800 Remove Dry Matter from tests
- #779 Remove Dry Matter functionality

Fixed

- #813 Saving AR results gives TypeError: can only compare to a set
- #799 On AR Listing, edit for Date Sampled not working when Sampler has a value
- #776 Analyses submission in Worksheet is slow
- #726 404 Error raised when clicking Print Samples Sheets from within a client

- #802 Remove Dry Matter remainders
- #781 Delete Permission on ARs for Lab Managers
- #784 Fix workflow state filter not kept when expanding categories in AS listing
- #786 Fix inactive services listed in AR "Manage Analyses" forms
- #775 Analyses on Analysis Requests are hyperlinked to their Worksheets
- #769 Traceback when submitting duplicate when Duplicate Variation is not set
- #771 Slow Searches in Listing Views
- #774 When retracting an Analysis Requests its analyses are also retracted
- #772 Improved UID check in API

6.15 1.2.4 (2018-04-06)

Added

- #741 CSV Importer for 'Cobas Taqman 48' Instrument Interface
- #737 Added Instrument: Metler Toledo DL55
- #730 Added Instrument: LaChat QuickChem FIA
- #729 Added Instrument: Varian Vista-PRO ICP
- #694 Added "Warn Min" and "Warn Max" subfields in Analysis Specifications
- #710 Added more builtin functions for Calculations

Changed

- #765 Department Filtering Improvements
- #746 StringField to UIDReferenceField for Default Department of Lab Contact
- #744 Updated WeasyPrint to 0.42.2
- #694 Out of range/shoulders logic redux, ported to *api.analysis*
- #694 Make getResultRange functions from Analysis-types consistent
- #694 Out of range/shoulders icons are rendered in AnalysesView

Removed

- #694 Removal of "Permitted % Error" subfield in Analysis Specifications
- #694 Removal of Adapters for out of range icons

Fixed

- #763 Datetime conversion error in CSV Importer of Taqman 48
- #761 Dormant Reference Definitions were listed for selection on WS Templates
- #735 Interim fields not created for QC Analyses on WSs
- #752 Published Date field of Analyses is never set
- #760 Default to empty the Title field when creating a new Analysis Specification
- #759 Date error in invoice batch creation although End date is after Start date

- #743 Traceback when accessing the view of a Statement
- #734 Chameleon parse error in productivity reports
- #750 Wrong redirect after Batch Label edit or creation
- #721 Fix filter functionality of Worksheets after sort/pagination
- #738 Traceback when Invalidating Analysis Requests
- #694 Bad calculation of min and max in ReferenceResults on negative result
- #694 Instrument validity not updated in accordance with latest QC tests
- #694 Result range shoulders computed badly on full/partial negative specs
- #725 Late Analyses are not filtered by Department
- #723 Solve Traceback on Print Samples Sheet
- #724 Department Filtering Portlet appears only for the manager with 'admin' username
- #720 Make automatic sticker printing work with 'registered' option activated
- #719 Fix interim fields result and calculation updating on the 2-Dimensional-CSV instrument
- #716 Samples from inside Batch are not filtered correctly
- #707 AR Add: Set default contact on client change
- #700 Fix filtering by review state in aggregated list of analyses
- #715 AR Rejection Fails when e-mail Notification is enabled
- #709 Fix removal not possible of last non-verified Analysis in Manage Analysis View
- #706 Filtering by Department is not working
- #712 Dates in date picker are visible again
- #703 Containers of Duplicated Analyses are not found
- #698 Fix Publish Actions for Batches
- #696 Worksheet count in dashboard is wrong in when filtered by department

6.16 1.2.3 (2018-02-23)

Added

- #666 Added Unpublished filter in Analysis Requests list

Changed

- #690 Refactored and improved client folder listing view
- #689 Allow to hide Analyses in any AR state
- #685 Display the stacked bars in evo charts sorted by number of occurrences
- #685 Small changes in colours palette for evo charts from Dashboard
- #684 Aggregated lists of analyses set to read-only mode
- #674 Dashboard with slightly better performance
- #621 AnalysesView code refactoring

- #668 AR Add: Debounce expensive XHR calls
- #660 Better style for reference widgets
- #627 Unassigned filter on Analysis Requests view does not work
- #659 Display the Unit in Profile Analyses Listing
- #636 Do not display "Advanced..." item in object's workflow actions menu
- #652 Added Sample Type, Partition ID and Date Sampled in Code_128_1x48mm sticker
- #655 Updated German Translations
- #647 Refactored bika.lims.bikalisting.js + several functional fixtures
- #637 Deassociate Analysis Request portal type from *worksheetanalysis_workflow*

Fixed

- #688 A traceback was appearing when navigating to rejected samples
- #686 Balloon button for adding Remarks is displayed while disabled in Setup
- #681 Invalidated Analysis Requests do not appear on Dashboard's evo chart
- #680 Fix Traceback with periodicity in DashboardView
- #679 Analysis could not set to "Hidden" in results view
- #677 Fix category toggling when the category name contains spaces
- #672 Traceback on automatic sticker printing in batch context
- #673 QC Analyses and Samples not totaled correctly in Worksheets list
- #670 Listings: Fix sort_on change on Show More click
- #653 Points in QC Charts are not displayed in accordance with capture date
- #662 Viewing Cancelled AR's fails
- #550 Wrong Listings of Analyses called from Dashboard
- #666 "Rejected" filter is displayed in AR lists regardless of Setup setting
- #666 "To be preserved" filter is displayed in AR lists regardless of Setup setting
- #666 "Scheduled sampling" is displayed in AR lists regardless of Setup setting
- #666 "To be sampled" filter is displayed in AR lists regardless of Setup setting
- #664 Improved async transition loading and workflow button rendering in listing tables
- #658 Worksheet listing view shows old- invalid Analysts
- #663 AR Report Listing shows all Reports
- #654 Default's Multi Analysis Request report gives a Traceback
- #649 Specification fields decimal-mark validator not working for new opened categories
- #637 Analysis Requests are never transitioned to assigned/unassigned
- #641 Broken Analyses list on ReferenceSample in Supplier
- #640 Broken Reference Sample Results view

6.17 1.2.2 (2018-02-09)

Added

- #594 Add button in Sample View for the createion of Analysis Requests
- #607 Ability to choose sticker template based on sample type
- #480 Sample panel in dashboard
- #617 Instrument import interface: 2-Dimensional-CSV
- #617 Instrument import interface: Agilent Masshunter
- #617 Instrument import interface: Shimadzu GCMS-QP2010 SE
- #617 Instrument import interface: Shimadzu GCMS-TQ8030 GC/MS/MS
- #617 Instrument import interface: Shimadzu ICPE-9000 Multitype
- #617 Instrument import interface: Shimadzu HPLC-PDA Nexera-I
- #617 Instrument import interface: Shimadzu LC MS/MS Nexera X2
- #537 Instrument import interface: Sysmex XT-4000i
- #536 Instrument import interface: Sysmex XT-1800i
- #607 Barcode and labelling depending on Sample Type
- #618 When previewing stickers the number of copies to print for each sticker can be modified.
- #618 The default number of sticker copies can be set and edited in the setup Sticker's tab.

Changed

- #619 Changed listing tables search logic to operate on catalog metadata
- #621 Change Errors to Warnings when importing instrument results

Fixed

- #639 Analysis Requests from inside Batch are not filtered correctly
- #591 Fixed workflow publish recursion error that reached max depth
- #634 Fix undefined Symbols in Sample Transition Guards
- #616 Fix character encodings in analysisservice duplication
- #624 TypeError: "Can't pickle objects in acquisition wrappers" (WorksheetTemplate)
- #530 Calculated results do not get updated when importing instrument results
- #614 Fix accreditation category titles
- #611 Advanced filter bar: filter Analysis Requests by Service name not working
- #622 (Re-)Installation always adds another identifier type
- #620 Client batch list is not filtered by state
- #628 Hide Department on lab contact inherited from Person
- #631 Traceback on stickers display

6.18 1.2.1 (2018-01-26)

Added

- #555 Don't allow the deactivation of Analysis Services with active dependencies
- #555 Don't allow the activation of Analysis Services with inactive dependents

Changed

- #569 Minimalistic dashboard indicators

Fixed

- #606 Handle unicode queries in Client ReferenceWidgetVocabulary
- #603 Out of range Icons are not displayed through all Analysis states
- #598 BadRequest error when changing Calculation on Analysis Service
- #593 Price/Spec/Interim not set in AR Manage Analyses
- #585 Empty value for Analysis Request column in aggregated list of analyses
- #578 Fix translation for review state titles in listings
- #580 Fix calculations using built-ins
- #563 Deactivated Analyses are added in new ARs when using Analysis Profiles/Template
- #562 Client Batch lists are empty
- #561 Sampler field is not displayed in Analysis Request Add form
- #559 Fix numeric field event handler in bika.lims.site.js
- #553 Fixed that images and barcodes were not printed in reports
- #551 Traceback in Worksheet Templates list when there are Instruments assigned
- #571 Added try/except around id-template format function to log key errors in ID generation

6.19 1.2.0 (2018-01-03)

Added

- #498 Added getPriorityText method to Analysis Request

Changed

- #519 #527 #528 bika.lims to senaite.core distribution

Fixed

- #522 Worksheets: Analyses listing does not show attached Analyses
- #514 Site Error when listing Dormant Worksheet Templates
- #517 Expired Reference Samples are displayed in Add Blank/Add Control views
- #517 Inactive services displayed for selection in Add Blank/Add Control views
- #516 List of Analyses Services is not properly filtered by state
- #516 Activate and Deactivate buttons do not appear in Analysis Services list
- #512 Duplicates transition to "Attachment due" after submit

- #499 Wrong slots when adding analyses manually in Worksheet with a WST assigned
- #499 When a Worksheet Template is used, slot positions are not applied correctly
- #499 Applying a WS template which references a Duplicate raises an Error
- #513 ShowPrices doctest is failing
- #488 JS Errors in bika.lims.analysisrequest.js

6.20 1.1.8 (2017-12-23)

Added

- #440 ITopLeft, ITopRight and ITopWide hooks (placeholders) in bikalisting
- #472 Dashboard panels visibility by roles
- #467 All/Mine filters in Dashboard panels
- #423 Instrument import interface for Abbott's m2000 Real Time

Changed

- #469 Remove unique field validator for Batch titles
- #459 PR-1942 Feature/instrument certification interval refactoring
- #431 Make ARAnalysesField setter to accept Analysis/Service objects

Fixed

- #494 Rejection reasons widget does not appear on rejection
- #492 Fix AR Add Form: CC Contacts not set on Contact Change
- #489 Worksheet Templates selection list is empty in Worksheets view
- #490 Fix AR Add Form: No specifications found if a sample type was set
- #475 Assigning Analyses to a WS raises AttributeError
- #466 UnicodeDecodeError if unicode characters are entered into the title field
- #453 Sample points do not show the referenced sample types in view
- #470 Sort order of Analyses in WS print view wrong
- #457 Calculation referring to additional python module not triggered
- #459 Traceback in Instruments list after adding a calibration certificate
- #454 Click on some analyses pops up a new page instead of object log
- #452 Traceback error when deleting attachment from Analysis Request
- #450 Traceback after clicking "Manage Results" in a WS w/o Analyses assigned
- #445 Fix AR Add Form: No sample points are found if a sample type was set

6.21 1.1.7 (2017-12-01)

Added

- #377 XML importer in Instrument Interface of Nuclisense EasyQ

Removed

- #417 Remove calls to deprecated function getService (from AbstractAnalysis)

Fixed

- #439 Cannot verify calculated analyses when retracted dependencies
- #432 Wrong indentation of services in Worksheet
- #436 Auto Import View has an Add Button displayed, but shouldn't
- #436 Clicking on the Add Button of Instrument Certifications opens an arbitrary Add form
- #433 Analyses not sorted by sortkey in Analysis Request' manage analyses view
- #428 AR Publication from Client Listing does not work
- #425 AR Listing View: Analysis profiles rendering error
- #429 Fix worksheet switch to transposed layout raises an Error
- #420 Searches by term with custom indexes do not work in clients folder view
- #410 Unable to select or deselect columns to be displayed in lists
- #409 In Add Analyses view, analyses id are displayed instead of Analysis Request IDs
- #378 Fix GeneXpert interface does not import results for multiple analyses
- #416 Fix inconsistencies with sorting criterias in lists
- #418 LabClerks don't have access to AR view after received and before verified
- #415 Referencefield JS UID check: Don't remove Profile UIDs
- #411 Analyses don't get selected when copying an Analysis Request without profiles

6.22 1.1.6 (2017-11-24)

Changed

- #390 Remove log verbosity of UIDReference.get when value is None or empty

Fixed

- #403 Calculations not triggered in manage results view
- #402 Sort Analysis Services correctly based on their Sortkey + Title (Again)
- #398 PR-2315 ID Server does not find the next correct sequence after flushing the number generator
- #399 PR-2318 AR Add fails silently if e.g. the ID of the AR was already taken
- #400 PR-2319 AR Add fails if an Analysis Category was disabled
- #401 PR-2321 AR Add Copy of multiple ARs from different clients raises a Traceback in the background
- #397 Fix Issue-396: AttributeError: uid_catalog on AR publication

6.23 1.1.5 (2017-11-20)

Added

- #386 PR-2297 Added seeding function to IDServer
- #372 Added build system to project root
- #345 'SearchableText' field and adapter in Batches
- #344 PR-2294 Allow year in any portal type's ID format string
- #344 PR-2210 ID Server and bika setup updates along with migation step
- #321 PR-2158 Multiple stickers printing in lists
- #319 PR-2112 Laboratory Supervisor
- #317 Enable backreferences associated to UIDReference fields
- #315 PR-1942 Instrument Certification Interval
- #292 PR-2125 Added descriptions for Analysis Requests
- #291 PR-1972 Landscape Layout for Reports
- #286 Added Github Issue/PR Template
- #281 PR-2269 Show the Unit in Manage Analyses View
- #279 Allow external Python library functions to be used in Calculation Formulas
- #279 Calculation formula test widgets
- #279 PR-2154 New ar add form

Changed

- #385 PR-2309 Unnecessary loops were done in instrument listing views
- #369 Let DateTimeField setter accept datetime.datetime objects and convert them
- #362 Add "Methods" column and hide unused columns in Analysis Services list
- #353 Remove deprecation warnings
- #338 Preserve Analysis Request order when adding into Worksheet
- #338 Analyses sorted by priority in Add Analyses view
- #333 Display analyses sorted by sortkey in results report
- #331 Sort analyses lists by sortkey as default
- #321 Sticker's autoprint generates PDF instead of browser's print dialog
- #312 Worksheet: "Print" does not display/print partial results
- #306 PR-2077 Better usability of Clients lists for sites with many users
- #298 PR-2246 Implemented ProxyField to fix data duplication between ARs and Samples

Fixed

- #419 'getLastVerificator' function of Abstract Analyses fails when there is no Verificator.
- #388 Unable to get the portal object when digesting/creating results report
- #387 ClientWorkflowAction object has no attribute 'portal_url' when publishing multiple ARs

- #386 PR-2313 UniqueFieldValidator: Encode value to utf-8 before passing it to the catalog
- #386 PR-2312 IDServer: Fixed default split length value
- #386 PR-2311 Fix ID Server to handle a flushed storage or existing IDs with the same prefix
- #385 PR-2309 Some objects were missed in instrument listing views
- #384 PR-2306 Do not use localized dates for control chart as it breaks the controlchart.js datetime parser
- #382 PR-2305 TypeError in Analysis Specification category expansion
- #380 PR-2303 UnicodeDecodeError if title field validator
- #379 Missing "Instrument-Import-Interface" relationship
- #375 Dependencies error in Manage Analyses view
- #371 Reflex rules don't have 'inactive_state' values set
- #365 LIMS installation fails during setting client permissions in bika_setup
- #364 Error on Manage Results view while adding new Analyses from different Category
- #363 PR-2293 Add CCEmails to recipients for Analysis Request publication reports
- #352 Traceback on listings where objects follow the bika_inactive_workflow
- #323 Allow IDServer to correctly allocate IDs for new attachments (add Attachment to portal_catalog)
- #344 PR-2273. Ensure no counters in the number generator before initialising id server
- #343 PR-2281 Fix publication preferences for CC Contacts
- #340 TypeError: "Can't pickle objects in acquisition wrappers" (Calculation)
- #339 Index not found warnings in bika listing
- #337 Error when adding reference analysis in a Worksheet
- #336 Accreditation Portlet renders an error message for anonymous users
- #335 The Lab Name is always set to "Laboratory" after reinstallation
- #334 TypeError (setRequestId, unexpected keyword argument) on AR Creation
- #330 Show action buttons when sorting by column in listings
- #318 PR-2205 Conditional Email Notification on Analysis Request retract
- #316 Small fixes related with i18n domain in Worksheet's print fixtures
- #314 'SamplingDate' and 'DateSampled' fields of AR and Sample objects don't behave properly
- #313 The PDF generated for stickers doesn't have the right page dimensions
- #311 PR-1931 Fixed Link User to Contact: LDAP Users not found
- #309 PR-2233 Infinite Recursion on Report Publication.
- #309 PR-2130 Copied ARs are created in random order.
- #308 Analysis Service' interim fields not shown
- #307 Fix sorting of Analysis Services list and disable manual sorting
- #304 PR-2081 Fixed multiple partition creation from ARTemplate
- #304 PR-2080 Batch Book raises an Error if the Batch inherits from 2 ARs
- #304 PR-2053 Computed Sample Field "SampleTypeUID" does not check if a SampleType is set

- #304 PR-2017 Fixed BatchID getter
- #304 PR-1946 Showing Verified Worksheets under all
- #299 PR-1931 Fixed Link User to Contact: LDAP Users not found
- #298 PR-1932 AttributeError: 'bika_setup' on login on a new Plone site w/o bika.lims installed
- #297 PR-2102 Inline rendered attachments are not displayed in rendered PDF
- #296 PR-2093 Sort order in Bika Setup Listings
- #294 PR-2016 Convert UDL and LDL values to string before copy
- #293 Fix analysis_workflow permissions for Field Analysis Results
- #284 PR-1917 Solved WF Translation issues and fixed WF Action Buttons in Bika Listings
- #283 PR-2252 Traceback if the title contains braces on content creation
- #282 PR-2266 Instrument Calibration Table fixes
- #280 PR-2271 Setting 2 or more CCContacts in AR view produces a Traceback on Save

6.24 1.0.0 (2017-10-13)

Added

- #269 Added IFrontPageAdapter, to make front page custom-redirections easier
- #250 Sanitize tool to fix wrong creation dates for old analyses

Fixed

- #272 Unknown sort_on index (getCategoryTitle) in Worksheet's Add Analyses view
- #270 ParseError in Reference Widget Search. Query contains only common words
- #266 Worksheet column appears blank in Aggregated List of Analyses
- #265 ValueError in productivity report
- #264 Fix permissions error on site install
- #262 DateSampled does not appear to users other than labman or administrator
- #261 Checking async processes fails due to Privileges of Client Contact
- #259 Error when saving and Analysis Request via the Save button
- #258 Sorting Analysis Requests by progress column does not work
- #257 AttributeError (getRequestUID) when submitting duplicate analyses
- #255 Client contacts cannot see Analysis Requests if department filtering is enabled
- #249 Unable to reinstate cancelled Analysis Requests

Security

- #256 Restrict the linkeage of client contacts to Plone users with Client role only
- #254 Anonymous users have access to restricted objects

6.25 3.2.0.1709-a900fe5 (2017-09-06)

Added

- #244 Asynchronous creation of Analysis Requests
- #242 Visibility of automatically created analyses because of reflex rule actions
- #241 Fine-grained visibility of analyses in results reports and client views
- #237 Performance optimizations in Analysis Request creation
- #236 Progress bar column in Analysis Requests list and Analyses number
- #233 Background color change on mouse over for fields table from ARAdd view
- #232 Display Priority in Analyses Add View from Worksheet and allow to sort
- #229 Highlight rows in bikalisting on mouse over
- #157 Catalog for productivity/management reports to make them faster

Changed

- #218 Render barcodes as bitmap images by default
- #212 Allow direct verification of analyses with dependencies in manage results view
- #213 Sampling Date and Date Sampled fields refactoring to avoid confusions
- #228 Translations updated
- #224 Remove warnings and unuseful elses in Analysis Request setters
- #193 Render transition buttons only if 'show_workflow_action' in view is true
- #191 Code sanitize to make Analysis Specifications folder to load faster

Fixed

- #248 Search using Client not working in Add Analyses (Worksheet)
- #247 Sample Type missing in analysis view for rejected samples
- #246 ZeroDivisionError when calculating progress
- #245 Missing Lab Contacts tab in Departments View
- #240 Unable to modify Sample point field in Analysis Request view
- #235 Fix Jsi18n adapter conflict
- #239 Sort on column or index is not valid
- #231 Partition inconsistences on secondary Analysis Requests
- #230 Priority not showing on Analysis Request listing
- #227 Malformed messages and/or html make i18ndude to fail
- #226 Action buttons are not translated
- #225 State inconsistencies when adding an analysis into a previous Analysis Request
- #223 TypeError when Analysis Service's exponential format precision is None
- #221 Filters by Service, Category and Client do not work when adding Analyses into a Worksheet
- #220 Not all departments are displayed when creating a new Lab Contact

- #219 When a Sample Point is modified in AR view, it does not get printed in report
- #217 Setupdata import fixes
- #216 Results reports appear truncated
- #215 All Samples are displayed in Analysis Request Add form, regardless of client
- #214 Status inconsistences in Analyses in secondary Analysis Requests
- #211 Sorting by columns in batches is not working
- #210 In some cases, the sampler displayed in results reports is wrong
- #209 AttributeError: 'NoneType' object has no attribute 'getPrefix' in Analysis Request add view
- #208 Rendering of plone.abovecontent in bika.lims.instrument_qc_failures_viewlet fails
- #206 Unknown sort_on index (getClientTitle) in Add Analyses view from Worksheet
- #202 Once a result is set, the checkbox is automatically checked, but action buttons do not appear
- #201 Results interpretation field not updated after verification or prepublish
- #200 Dependent analyses don't get selected when analysis with dependents is choosen in AR Add view
- #199 AttributeError when adding a Blank in a Worksheet because of Service without category
- #198 The assignment of a Calculation to a Method doesn't get saved apparently, but does
- #196 Error invalidating a published test report (retract_ar action)
- #195 List of Analysis Request Templates appears empty after adding a Sampling Round Template
- #192 Date Sampled is not displayed in Analysis Request View
- #190 Bad time formatting on Analysis Request creation within a Sampling Round
- #189 Bad time formatting when creating a secondary Analysis Request
- #187 After verification, department managers are not updated in results report anymore
- #185 Analysis services list not sorted by name
- #183 Decimals rounding is not working as expected when uncertainties are set
- #181 Client contact fields are not populated in Sampling Round add form
- #179 Wrong values for "Sampling for" and "Sampler for scheduled sampling" fields after AR creation
- #178 Sampler information is wrong in results reports
- #175 Changes in "Manage Analyses" from "Analysis Request" have no effect
- #173 NameError (global name 'safe_unicode' is not defined) in Analysis Request Add view
- #171 Error printing contact address
- #170 Index error while creating an Analysis Request due to empty Profile
- #169 ValueError (Unterminated string) in Analysis Request Add view
- #168 AttributeError 'getBatch' after generating barcode
- #166 Analyses don't get saved when creating an Analysis Request Template
- #165 AttributeError in Bika Setup while getting Analysis Services vocabulary
- #164 AttributeError on Data Import: 'NoneType' object has no attribute 'Import'
- #161 TypeError from HistoryAwareReferenceField while displaying error message

- #159 Date published is missing on data pulled through API
- #158 Date of collection greater than date received on Sample rejection report
- #156 Calculation selection list in Analysis Service edit view doesn't get displayed
- #155 Error while rejecting an Analysis Request. Unsuccessful AJAX call

6.26 3.2.0.1706-315362b (2017-06-30)

Added

- #146 Stickers to PDF and new sticker 2"x1" (50.8mm x 25.4mm) with barcode 3of9
- #152 Caching to make productivity/management reports to load faster

Changed

- #150 Dynamic loading of allowed transitions in lists
- #145 Workflow refactoring: prepublish
- #144 Workflow refactoring: publish

Fixed

- #154 AttributeError on upgrade step v1705: getDepartmentUID
- #151 State titles not displayed in listings
- #149 Decimal point not visible after edition
- #143 Fix AttributeError 'getProvince' and 'getDistrict' in Analysis Requests view
- #142 AttributeError on publish: 'getDigest'
- #141 AttributeError on upgrade.v3_2_0_1705: 'NoneType' object has no attribute 'aq_parent'

6.27 3.2.0.1706-baed368 (2017-06-21)

Added

- #133 Multiple use of instrument control in Worksheets

Fixed

- #139 Reference migration fails in 1705 upgrade
- #138 Error on publishing when contact's full name is empty
- #137 IndexError while notifying rejection: list index out of range
- #136 Worksheets number not working in Dashboard
- #135 Fix string formatting error in UIDReferenceField
- #132 ValueError in worksheets list. No JSON object could be decoded
- #131 "Show more" is missing on verified worksheets listing
- #129 Unsupported operand type in Samples view

6.28 3.2.0.1706-afc4725 (2017-06-12)

Fixed

- #128 TypeError in Analysis Request' manage results view: object of type 'Missing.Value' has no len()
- #127 AttributeError while copying Service: 'float' object has no attribute 'split'
- #126 AttributeError during results publish: getObject
- #123 Analysis Request state inconsistences after upgrade step v3.2.0.1705
- #122 ValueError on results file import

6.29 3.2.0.1706-f32494f (2017-06-08)

Added

- #120 Add a field in Bika Setup to set the default Number of ARs to add
- #88 GeneXpert Results import interface
- #85 Sticker for batch
- #84 Sticker for worksheet
- #83 Adapter to make the generation of custom IDs easier
- #82 Added a method the get always the client in stickers
- #75 Wildcards on searching lists

Changed

- #106 Predigest publish data
- #103 Prevent the creation of multiple attachment objects on results import
- #101 Performance improvement. Remove Html Field from AR Report
- #100 Performance improvement. Replacement of FileField by BlobField
- #97 Performance improvement. Removal of versionable content types
- #95 Performance improvement. Analysis structure and relationship with Analysis Service refactored
- #58 Defaulting client contact in Analysis Request Add view

Fixed

- #118 Results import throwing an error
- #117 Results publishing not working
- #113 Biohazard symbol blocks the sticker making it impossible to be read
- #111 Fix error while submitting reference analyses
- #109 Remarks in analyses (manage results) are not displayed
- #105 System doesn't save AR when selected analyses are from a department to which current user has no privileges
- #104 ReferenceException while creating Analysis Request: invalid target UID
- #99 Instrument's getReferenceAnalyses. bika.lims.instrument_qc_failures_viewlet fails

- #94 Site Search no longer searching Analysis Requests
- #93 Analyses did not get reindexed after recalculating results during import
- #92 Analyses disappearing on sorting by date verified
- #91 KeyError on Samples view: 'getSamplingDate'
- #90 AttributeError on Analysis Request submission: 'NoneType' object has no attribute 'getDepartment'
- #89 Analysis to be verified not showing results
- #87 AttributeError in analyses list: 'getNumberOfVerifications'
- #82 JS error while checking for rejection reasons in client view
- #80 CatalogError: Unknown sort_on index (Priority)
- #79 ValueError in Bika's DateTimeWidget
- #78 CatalogError in Batch View. Unknown sort_on index (BatchID)
- #77 ValueError in AR Add: time data '2016-05-10' does not match format '%Y-%m-%d %H:%M'
- #76 AttributeError in Client ARs view: bika_catalog
- #74 AttributeError: 'NoneType' object has no attribute 'getCalculation'
- #73 Analyses disappearing on sorting by date verified
- #72 Cancelled analyses appearing in aggregated list of analyses
- #71 AttributeError on publish: 'getRequestID'
- #70 The number of pending verifications displayed in analyses list is wrong
- #69 Selecting a sticker template in AR's sticker preview does nothing
- #68 Error while listing analyses in Analysis Request details view
- #67 Show more button is not working in Analysis Services list
- #66 TypeError in Worksheets view. TypeError: 'list' object is not callable
- #65 Fix error when an object has no status defined while listing in WS
- #64 AttributeError: 'NoneType' object has no attribute 'getInstrumentEntryOfResults
- #63 If login failed, setDepartmentCookies throws an IndexError
- #61 Show more button is not working in Worksheet's Add Analyses view
- #60 Index Error in Analysis Request Add view
- #59 AttributeError (NoneType) in service.getInstruments()
- #57 Select all departments option is not working
- #56 Client and District not sortable in Analysis Requests listing
- #52 System throwing error on opening "Verified" folder

6.30 3.2.0.1703-0f28b48 (2017-03-30)

Added

- #39 Performance improvement. Make use of brains in Worksheets lists

- #32 Performance improvement. Catalog for analyses and make use of brains

Fixed

- #48 Error on AR publish. Global name 'traceback' is not defined (getServiceUsingQuery)
- #47 Error in CloneAR during retraction. AttributeError: setRequestID
- #46 Error rejecting an Analysis Request
- #45 CatalogError in Dashboard. Unknown sort_on index (created) in view.get_sections()
- #44 AttributeError in worksheets view
- #43 Sort not working on all lists
- #41 No Service found for UID None
- #40 Client Sample ID is missing in Analysis Request Add view

6.31 3.2.0.1703-1c2913e (2017-03-20)

Added

- #33 New Analysis Request Add form outside client

Fixed

- #37 Publish results throwing an error
- #36 System is not printing labels automatically
- #35 Equipment interface is not working
- #34 Results import submission error

6.32 3.2.0.1703-e596f2d (2017-03-08)

Added

- #25 Instrument import without user intervention
- #22 Date Tested range filter on lists
- #20 Added filter bar in Aggregated list of analyses
- HEALTH-364: Added country/province/district columns to client listings
- Add buttons to export lists to csv and xml formats
- Additional "printed" workflow for analysis requests once published

Changed

- #12 Multi-method assignment and Virtual-Real Instrument correspondence
- #11 Restrictions in manual instrument import - Instruments and interfaces
- #10 Performance improvement. Catalog for Analysis Requests and use of brains

Fixed

- #26 Publishing bug due to SMTP Authentication
- #24 Condition rule being affected on duplicate samples

- #23 Date of Birth: crash if date is before 1900
- #21 Rejection option does not appear if only one column in AR Add form
- #19 Inconsistent status of Analysis in WS after AR rejection
- #13 Number of verifications no longer taking effect
- HEALTH-568: TaqMan 96 interface not working well
- HEALTH-567: Nuclisens interface not working well

6.33 3.2.0.1701-26f2c4b (2017-01-17)

- LIMS-2477: Reference Analysis has no dependencies; remove guard that assumes it does
- LIMS-2465: Not possible to translate Bika Listing Table Workflow Action Buttons
- LIMS-1391: Add configurable identifier types (CAS# for AnalysisService)
- LIMS-2466: Central Instrument Location Management
- LIMS-2357: Custom Landing Page and Link to switch between the Front Page and Dashboard
- LIMS-2341: Cleanup and format default Multi-AR COA
- LIMS-2455: Contact/Login Linkage Behavior
- LIMS-2456: Restrict duplicate slots in worksheet templates to routine analyses only.
- LIMS-2447: getDatePublished index not indexed correctly at time of AR publication
- LIMS-2404: AR list in batches permitted sampling without Sampler and Sampling date provided
- LIMS-2380: ARs are created in correct order (order of columns in ar-create form)
- LIMS-2394: Calculation failure in worksheets. TDS Calc misfires again.
- LIMS-2391: Use source analysis's sample ID in duplicate analysis IDs
- LIMS-2351: Field analyses without results do not prevent Samples from being received
- LIMS-2366: Workflow. AR stays in Received state with all Analyses in To be Verifie
- LIMS-2384: ARImport: Workflow state of imported ARs and their Analyses not synchronised.
- LIMS-2369: Workflow. Sampler and Date Sampled should be compulsory
- LIMS-2355: Unable to view dormant/active filters in some bika_setup pages
- LIMS-2344: Fix some UI javascript failures when viewing ARs
- LIMS-2319: AR Add: Deleting a selected CC Contact corrupts the UID of reference widgets
- LIMS-2325: Allow SampleTypes to be linked with Client Sample Points
- LIMS-2324: WS export to the LaChat Quick Chem FIA
- LIMS-2298: Add filter in Clients list
- LIMS-2299: Add ui for editing ar_count in all analysisrequest lists
- LIMS-2268: Instrument Interface. Vista Pro Simultaneous ICP, bi-directional CSV
- LIMS-2261: Cannot create analysis request
- LIMS-1562: Using a Sample Round. Basic form and printed form

- LIMS-2266: Crating partitions through Add form, doesn't create partitions.
- HEALTH-394: Sample sticker layout. 2 new sticker layouts, 2 stickers per row
- LIMS-2032: AS Methods initialise with 1st available Instrument (loading setup data)
- LIMS-2014: I can only select a Default Method for an AS if Manual results capture is on
- LIMS-2181: An analysis is not stopped from using an invalid instrument
- HEALTH-310: Implemented Nuclisens EasyQ instrument importer
- HEALTH-319: Instrument. Roche Cobas Taqman 96
- LIMS-2091: Table Column Display options Everywhere
- LIMS-2207: Indentation in analysisrequests.py
- LIMS-2208: WinescanCSVParser class instance variable misspelling
- LIMS-1832: New Results Template, COA. Multiple ARs in columns
- LIMS-2148: Unable to sort Bika Listing tables
- LIMS-1774: Shiny graphs for result ranges
- Replacement of pagination by 'Show more' in tables makes the app faster
- Add Bika LIMS TAL report reference in reports preview
- Simplify instrument interface creation for basic CSV files
- Scheduled sampling functionality added
- LIMS-2257: Scheduled sampling
- LIMS-2255: Switch to Chameleon (five.pt) for rendering TAL templates
- System-wide filter by department
- Allow to assign a lab contact to more than one department
- Multi-verification of analyses, with different verification types
- Add option to allow multi-approval (multi-verification) of results
- Added Analyses section in the Dashboard
- Add option to allow labman to self-verify analysis results
- Replacement of pagination by 'Show more' in tables makes the app faster
- Add Bika LIMS TAL report reference in reports preview
- Simplify instrument interface creation for basic CSV files

6.34 3.1.13 (2016-12-28)

- LIMS-2299: Add ui for editing ar_count in all analysisrequest lists
- Removed commented HTML that was causing Chameleon to choke when adding ARs.

6.35 3.1.12 (2016-12-15)

- HEALTH-569 Bar code printing not working on sample registration
- Pinned CairoSVG to 1.0.20 (support for Python 2 removed in later versions)

6.36 3.1.11 (2016-04-22)

- LIMS-2252: Partitions not submitted when creating AR if the form is submitted before partitions are calculated
- LIMS-2223: Saving a recordswidget as hidden fails
- LIMS-2225: Formatted results not displayed properly in Worksheet's transposed layout
- LIMS-2001: Duplicate for one analysis only
- LIMS-1809: Typos. Perdiod an missing spaces
- LIMS-2221: Decimal mark doesn't work in Sci Notation
- LIMS-2219: Using a SciNotation diferent from 'aE+b / aE-b' throws an error
- LIMS-2220: Raw display of exponential notations in results manage views
- LIMS-2216: Results below LDL are not displayed in reports
- LIMS-2217: Specifications are not set in analyses on Analysis Request creation
- LIMS-2218: Result is replaced by min or max specs when "<Min" or ">Max" fields are used
- LIMS-2215: Decimal mark not working
- LIMS-2203: 'Comma' as decimal mark doesnt work
- LIMS-2212: Sampling round- Sampling round templates show all system analysis request templates
- LIMS-2209: error in manage analyises
- LIMS-1917: Inconsistencies related to significant digits in uncertainties
- LIMS-2015: Column spacing on Client look-up
- LIMS-1807: Validation for Start Date - End date relationship while creating invoices and price lists
- LIMS-1991: Sort Order for Analysis Categories and Services
- LIMS-1521: Date verified column for AR lists
- LIMS-2194: Error when submitting a result
- LIMS-2169: Cannot start instance
- WINE-125: Client users receive unauthorized when viewing some published ARs

6.37 3.1.10 (2016-01-13)

- Updated Plone to 4.3.7
- Dashboard: replace multi-bar charts by stacked-bar charts
- LIMS-2177: template_set error when no template has been selected
- HEALTH-410: AR Create. Auto-complete Contact field if only 1

- LIMS-2175: “NaN” is shown automatically for result fields that have AS with “LDL” enabled and then an error is shown after submitting a result
- LIMS-1917: Inconsistencies related to significant digits in uncertainties
- LIMS-2143: Statements vs Invoices
- LIMS-1989: Retracting a published AR fails if one or more ASs has been retracted before publishing
- LIMS-2071: Can’t generate Invoice Batch/Monthly Statements
- WINE-71: Instrument. BBK WS export to FIA fails
- WINE-72: Instrument. BBK WineScan Auto Import fails
- WINE-58: Instrument. BBK FIAStar import fails
- WINE-76: WineScan FT120 Import warnings incorrect?
- LIMS-1906: Spaces should be stripped out of the keywords coming from the Instrument
- LIMS-2117: Analysis Categories don’t expand on Analysis Specification creation
- LIMS-1933: Regression: Selecting secondary AR in client batches, fails.
- LIMS-2075: Ensure hiding of pricing information when disabled in site-setup
- LIMS-2081: AR Batch Import WorkflowException after edit
- LIMS-2106: Attribute error when creating AR inside batch with no client.
- LIMS-2080: Correctly interpret default (empty) values in ARImport CSV file
- LIMS-2115: Error rises when saving a Calculation
- LIMS-2116: JSONAPI throws an UnicodeDecodeError
- LIMS-2114: AR Import with Profiles, no Analyses are created
- LIMS-2132: Reference Analyses got the same ID
- LIMS-2133: Once in a while, specs var is going empty in results reports
- LIMS-2136: Site Error on AR Verification
- LIMS-2121: Fix possible Horiba ICP csv handling errors
- LIMS-2042: Improving Horiba ICP to avoid Element Symbols as keywords
- LIMS-2123: Analysis Categories don’t expand in Worksheet Templates
- LIMS-1993: Existing Sample look-up for AR Create in Batch does not work
- LIMS-2124: QR missing on sticker preview
- LIMS-2147: Add ARImport schema fields when creating ARs
- LIMS-409: ShowPrices setting was getting ignored in some contexts
- LIMS-2062: Cancelled ARs no longer appear in analysisrequest folder listings
- LIMS-2076: Cancelled batches appear in listing views
- LIMS-2154: Hide inactive ARs from BatchBook view
- LIMS-2134: Inactive services appear in AR Create
- LIMS-2139: WS Blank and Control Selection renderes whole page
- LIMS-2156: Ignore blank index values when calculating ReferenceAnalysesGroupID

- LIMS-2157: Cancelled ARs appear in AR listing inside Batches
- LIMS-2042: Horiba ICP: Missing 'DefaultResult' for imported rows
- LIMS-2030: Assign ARs in alphabetical ID order to WS
- LIMS-2167: Cannot assign a QC analysis to an invalid instrument
- LIMS-2067: Prevent initial method/instrument query for each analysis
- WINE-82: Ignore invalid entry in Sample field during AR creation
- LIMS-1717: Workflow transitions in edit context do not take effect
- WINE-111: Do not attempt formatting of 'nan' analysis result values
- WINE-114: Some users cannot view published ARs (unauthorised)
- WINE-122: Transposed worksheet layout failed while rendering empty slots
- LIMS-2149: Missing analyses can cause error accessing worksheet
- LIMS-1521: Date verified column for AR lists
- LIMS-2015: Column spacing on Client look-up
- LIMS-1807: Validation for Start Date - End Date relationship

6.38 3.1.9 (2015-10-8)

- LIMS-2068: LIMS-2068 Urgent. Analysis Catgories don't expand
- LIMS-1875: Able to deactivate instruments and reference samples without logging in
- LIMS-2049: Displaying lists doesn't work as expected in 319
- LIMS-1908: Navigation tree order
- LIMS-1543: Add "Security Seal Intact Y/N" checkbox for partition container
- LIMS-1544: Add "File attachment" field on Sample Point
- LIMS-1949: Enviromental conditions
- LIMS-1549: Sampling Round Templates privileges and permissions
- LIMS-1564: Cancelling a Sampling Round
- LIMS-2020: Add Sampling Round - Department not available for selection
- LIMS-1545: Add "Composite Y/N" checkbox on AR Template
- LIMS-1547: AR Templates tab inside Sampling Round Template
- LIMS-1561: Editing a Sampling Round
- LIMS-1558: Creating Sampling Rounds
- LIMS-1965: Modified default navtree order for new installations
- LIMS-1987: AR Invoice tab should not be shown if pricing is toggled off
- LIMS-1523: Site Error when transitioning AR from 'Manage Analyses' or 'Log' tab
- LIMS-1970: Analyses with AR Specifications not displayed properly in AR Add form
- LIMS-1969: AR Add error when "Categorise analysis services" is disabled

- LIMS-1397: Fix Client Title accessor to prevent catalog error when data is imported
- LIMS-1996: On new system with no instrument data is difficult to get going.
- LIMS-2005: Click on Validations tab of Instruments it give error
- LIMS-1806: Instrument Interface. AQ2. Seal Analytical - Error
- LIMS-2002: Error creating Analysis Requests from batch.
- LIMS-1996: On new system with no instrument data it is difficult to get going. The warnings could be confusing
- LIMS-1312: Transposed Worksheet view, ARs in columns
- LIMS-1760: Customised AR Import spreadsheets (refactored, support importing to Batch)
- LIMS-1548: Client-specific Sampling Round Templates
- LIMS-1546: Sampling Round Template Creation and Edit view
- LIMS-1944: Prevent concurrent form submissions from clobbering each other's results
- LIMS-1930: AssertionError: Having an orphan size, higher than batch size is undefined
- LIMS-1959: Not possible to create an AR
- LIMS-1956: Error upgrading to 319
- LIMS-1934: Hyperlinks in invoices
- LIMS-1943: Stickers preview and custom stickers templates support
- LIMS-1855: Small Sticker layout. QR-code capabilities
- LIMS-1627: Pricing per Analysis Profile
- HEALTH-279: AS IDs to be near top of page. Columns in AS list
- LIMS-1625: Instrument tab titles and headers do not correspond
- LIMS-1924: Instrument tab very miss-titled. Internal Calibration Tests
- LIMS-1922: Instrument out of date typo and improvement
- HEALTH-175: Supplier does not resolve on Instrument view page
- LIMS-1887: uniquefield validator doesn't work properly
- LIMS-1869: Not possible to create an Analysis Request
- LIMS-1867: Auto-header, auto-footer and auto-pagination in results reports
- LIMS-1743: Reports: ISO (A4) or ANSI (letter) pdf report size
- LIMS-1695: Invoice export function missing
- LIMS-1812: Use asynchronous requests for expanding categories in listings
- LIMS-1811: Refactor AR Add form Javascript, and related code.
- LIMS-1818: Instrument Interface. Eltra CS-2000
- LIMS-1817: Instrument Interface. Rigaku Supermini XRF
- New System Dashboard for LabManagers and Admins

6.39 3.1.8.3 (2015-10-01)

- LIMS-1755: PDF writer should be using a world-writeable tmp location
- LIMS-2041: Resolve ${analysis_keyword) in instrument import alert.
- LIMS-2041: Resolve translation syntax error in instrument import alert
- LIMS-1933: Secondary Sample selection in Client Batches does not locate samples

6.40 3.1.8.2 (2015-09-27)

- LIMS-1996: On new system with no instrument data is difficult to get going.
- LIMS-1760: Customised AR Import spreadsheets (refactored, support importing to Batch)
- LIMS-1930: AssertionError: Having an orphan size, higher than batch size is undefined
- LIMS-1818: Instrument Interface. Eltra CS-2000
- LIMS-1817: Instrument Interface. Rigaku Supermini XRF
- LIMS-2037: Gracefully anticipate missing analysis workflow history
- LIMS-2035: Prevent Weasyprint flooding due to asyncronous publish

6.41 3.1.8.1 (2015-06-23)

- LIMS-1806: Instrument Interface. AQ2. Seal Analytical - Error
- LIMS-1760: Customised AR Import spreadsheets (refactored, support importing to Batch)
- Fix portlets.xml for Plone 4.3.6 compatibility

6.42 3.1.8 (2015-06-03)

- LIMS-1923: Typo InstrumentCalibration
- HEALTH-287: Hyperlink in Instrument messages
- LIMS-1929: Translation error on Instrument Document page
- LIMS-1928 Asset Number on Instruments' Certificate tab should use Instrument's default
- LIMS-1929: Translation error on Instrument Document page
- LIMS-1773: Instrument. Thermo Fisher ELISA Spectrophotometer
- LIMS-1697: Error updating bika.lims 317 to 318 via quickinstaller
- LIMS-1820: QC Graphs DateTime's X-Axis not well sorted
- LIMS-280 : System IDs starting from a specific value
- LIMS-1819: Bika LIMS in footer, not Bika Lab Systems
- LIMS-1808: Uncertainty calculation on DL
- LIMS-1522: Site Error adding display columns to sorted AR list

- LIMS-1705: Invoices. Currency unit overcooked
- LIMS-1806: Instrument Interface. AQ2. Seal Analytical
- LIMS-1770: FIAStar import 'no header'
- LIMS-1771: Instrument. Scil Vet abc Plus
- LIMS-1772: Instrument. VetScan VS2
- LIMS-1507: Bika must notify why is not possible to publish an AR
- LIMS-1805: Instrument Interface. Horiba JY ICP
- LIMS-1710: UnicodeEncode error while creating an Invoice from AR view
- WINE-44: Sample stickers uses Partition ID only if ShowPartitions option is enabled
- LIMS-1634: AR Import fields (ClientRef, ClientSid) not importing correctly
- LIMS-1474: Disposed date is not shown in Sample View
- LIMS-1779: Results report new fields and improvements
- LIMS-1775: Allow to select LDL or UDL defaults in results with readonly mode
- LIMS-1769: Allow to use LDL and UDL in calculations.
- LIMS-1700: Lower and Upper Detection Limits (LDL/UDL). Allow manual input
- LIMS-1379: Allow manual uncertainty value input
- LIMS-1324: Allow to hide analyses in results reports
- LIMS-1754: Easy install for LIMS' add-ons was not possible
- LIMS-1741: Fixed unwanted overlay when trying to save supply order
- LIMS-1748: Error in adding supply order when a product has no price
- LIMS-1745: Retracted analyses in duplicates
- LIMS-1629: Pdf reports should split analysis results in different pages according to the lab department
- Some new ID Generator's features, as the possibility of select the separator type
- LIMS-1738: Regression. 'NoneType' object has no attribute 'getResultsRangeDict'
- LIMS-1739: Error with results interpretation field of an AR lacking departments
- LIMS-1740: Error when trying to view any Sample
- LIMS-1724: Fixed missing start and end dates on reports
- LIMS-1628: There should be a results interpretation field per lab department
- LIMS-1737: Error when adding pricelists of lab products with no volume and unit
- LIMS-1696: Decimal mark conversion is not working with "<0,002" results type
- LIMS-1729: Analysis Specification Not applying to Sample when Selected
- LIMS-1507: Do not cause exception on SMTPServerDisconnect when publishing AR results.

6.43 3.1.7 (2015-02-26)

- LIMS-1693: Error trying to save a new AR
- LIMS-1570: Instrument interface: Roche Cobas Taqman 48
- LIMS-1520: Allow to invalidate verified ARs
- LIMS-1690: Typo. Instrument page
- LIMS-1688: After AR invalidation, ARs list throws an error
- LIMS-1569: Instrument interface: Beckman Coulter Access 2
- LIMS-1689: Error while creating a new invoice batch
- LIMS-1266: Sampling date format error
- LIMS-1365: Batch search parameters on Work sheets/Work sheets insides Batches
- LIMS-1428: After receiving a sample with Sampling Workflow enable is not possible to input results
- LIMS-1540: When accent characters are used in a "Sample Type" name, it is not possible to create a new AR
- LIMS-1617: Error with bin/test
- LIMS-1571: Instrument interface: Sysmex XS-1000i
- LIMS-1574: Fixed AR and Analysis attachments
- LIMS-1670: Fixed windows incompatibility in TAL (referencewidget.pt)
- LIMS-1594: Added option to select landing page for clients in configuration registry
- LIMS-1594: Re-ordered tabs on Client home page
- LIMS-1520: Allow to invalidate verified ARs
- LIMS-1539: Printable Worksheets. In both AR by row or column orientations
- LIMS-1199: Worksheet totals in WS lists
- LIMS-257: Set Blank and Warning icons in Reference Sample main view
- LIMS-1636: Batch Sample View crash
- LIMS-1524: Invalidate email does not have variables populated
- LIMS-1572: Instrument interface: Sysmex XS-500i
- LIMS-1575: Thermo Arena 20XT
- LIMS-1423: Save details when AR workflow action kicked off
- LIMS-1624: Import default test.xlsx fails
- LIMS-1614: Error when selecting Analysis Administration Tab after receiving a sample with Sampling Workflow enabled
- LIMS-1605: Tescan TIMA interface
- LIMS-1604: BioDrop uLite interface
- LIMS-1603: Life Technologies Qubit interface
- LIMS-1517: Storage field tag untranslated?
- LIMS-1518: Storage Location table
- LIMS-1527: CC Contact on AR view (edit) offers all contacts in system

- LIMS-1536: Add button [Add], to alow quickly addings in referencewidget
- LIMS-1587: Better support for extension of custom sample labels
- LIMS-1622: Version Check does not correctly check cache
- LIMS-1623: Implement bika-frontpage as a BrowserView

6.44 3.1.6 (2014-12-17)

- LIMS-1530: Scrambled Analysis Category order in Published Results
- LIMS-1529: Error while inserting an AR with container-based partitioning is required
- LIMS-1460: Additional field in AR for comments or results interpretation
- LIMS-1441: An error message related to partitions unit is shown when selecting analysis during AR creation
- LIMS-1470: AS Setup. File attachment field tag is missing
- LIMS-1422: Results doesn't display yes/no once verified but 1 or 0
- LIMS-1486: Typos in instrument messages
- LIMS-1498: Published Results not Showing for Logged Clients
- LIMS-1445: Scientific names should be written in italics in published reports
- LIMS-1389: Units in results publishing should allow super(sub)script format, for example in cm2 or m3
- LIMS-1500: Alere Pima's Instrument Interfice
- LIMS-1457: Exponential notation in published AR pdf should be formatted like a×10^b instead of ae^+b
- LIMS-1334: Calculate result precision from Uncertainty value
- LIMS-1446: After retracting a published AR the Sample gets cancelled
- LIMS-1390: More workflow for Batches
- LIMS-1378: Bulking up Batches
- LIMS-1479: new-version and upgrade-steps should be python viewlets
- LIMS-1362: File attachment uploads to Batches
- LIMS-1404: New Batch attributes (and their integration with existing ones on Batch views)
- LIMS-1467: Sample Point Lookup doesn't work on AR modify
- LIMS-1363: Batches per Client
- LIMS-1405: New Sample and AR attributes
- LIMS-1085: Allow Clients to add Attachments to ARs
- LIMS-1444: In AR published report accredited analysis services are not marked as accredited
- LIMS-1443: In published reports the publishing date is not shown in the pdf
- LIMS-1420: Status filter is not kept after moving to next page
- LIMS-1442: Sample Type is not filtred by Sample Point
- LIMS-1448: Reports: when you click on "Analysis turnaround time" displays others
- LIMS-1440: Error when trying to publish with analysis from different categories

- LIMS-1459: Error when checking instrument validity in manage_results
- LIMS-1430: Create an AR from batch allows you to introduce a non existent Client and Contacts don't work properly
- After modifying analysis Category, reindex category name and UID for all subordinate analyses
- Setup data import improvements and fixes
- Simplify installation with a custom Plone overview and add site

6.45 3.1.5 (2014-10-06)

- LIMS-1082: Report Barcode. Was images for pdf/print reports etc
- LIMS-1159: reapply fix for samplepoint visibility
- LIMS-1325: WSTemplate loading incompatible reference analyses
- LIMS-1333: Batch label replace with standard Plone keyword widget
- LIMS-1335: Reference Definitions don't sort alphabetically on WS Template lay-outs
- LIMS-1345: Analysis profiles don't sort
- LIMS-1347: Analysis/AR background colour to be different to for Receive and To be Sampled
- LIMS-1360: Number of analyses in ARs folder view
- LIMS-1374: Auto label printing does not happen for an AR drop-down receive
- LIMS-1377: Error when trying to publish after updating branch hotfix/next or develop
- LIMS-1378: Add AR/Sample default fields to Batch
- LIMS-1395: front page issue tracker url
- LIMS-1402: If no date is chosen, it will never expire." not been accomplished
- LIMS-1416: If a sample point has a default sample type the field is not pulled automatically during AR template creation
- LIMS-1425: Verify Workflow (bika_listing) recursion
- added 'getusers' method to JSON API
- Added 'remove' method to JSON API
- Added AR 'Copy to new' action in more contexts
- Added basic handling of custom Sample Preparation Workflows
- Added decimal mark configuration for result reports
- Added help info regards to new templates creation
- Added IAcquireFieldDefaults - acquire field defaults through acquisition
- Added IATWidgetVisibility - runtime show/hide of AT edit/view widgets
- Added watermark on invalid reports
- Added watermark on provisional reports
- Alert panel when upgrades are available
- All relevant specification ranges are persisted when copying ARs or adding analyses

- Allow comma entry in numbers for e.g. German users
- Bika LIMS javascripts refactoring and optimization
- Fix ZeroDivisionError in variation calculation for DuplicateAnalysis
- Fixed spreadsheet load errors in Windows.
- Fixed template rendering errors in Windows
- JSONAPI update: always use field mutator if available
- JSONAPI: Added 'remove' and 'getusers' methods.
- Refactored ARSpecs, and added ResultsRange field to the AR

6.46 3.1.4.1 (2014-07-24)

- 3.1.4 release was broken, simple ARs could not be created.
- LIMS-1339: Published reports should use "±" symbol instead of "+/-"
- LIMS-1327: Instrument from worksheet
- LIMS-1328: Instrument calibration test graphs do not work on multiple samples
- LIMS-1347: Analysis/AR background colour to be different to for Receive and To be Sampled
- LIMS-1353: Analyses don't sort in Attachment look-up
- Preview for Results reports
- Single/Multi-AR preview
- Allows to cancel the pre-publish/publish process
- Results reports. Allows to make visible/invisible the QC analyses
- Results reports. Allows to add new custom-made templates
- Results reports. JS machinery allowed for pdf reporting

6.47 3.1.4 (2014-07-23)

- LIMS-113: Allow percentage value for AS uncertainty
- LIMS-1087: Prevent listing of empty categories
- LIMS-1203: Fix Batch-AnalysisRequests query
- LIMS-1207: LIMS-113 Allow percentage value for AS uncertainty
- LIMS-1221: use folder icon for ARImports in nav
- LIMS-1240: fix permissions for "Copy To New" in AR lists
- LIMS-1330: handle duplicate of reference analysis
- LIMS-1340: soft-cache validator results
- LIMS-1343: Prevent sudden death if no version information is available
- LIMS-1352: SamplingWorkflow not saved to sample
- LIMS-334: Add Service/ExponentialFormatPrecision

- LIMS-334: Added ExponentialFormatThreshold setting
- LIMS-334: Allow exponential notation entry in numeric fields
- LIMS-334: Exponent Format used for analysis Result
- LIMS-334: Remove duplicate getFormattedResult code
- LIMS-83: Update Method->calculation reference version when Calculation changes
- Formula statements can be written on multiple lines for clarity.
- Replace kss-bbb ajax-spinner with a quieter one
- bika.lims.utils.log logs location url correctly

6.48 3.1.3 (2014-07-17)

- Missing fixes from 3.1.2
- LIMS-671: Preferred/Restricted client categories
- LIMS-1251: Supply order permission error
- LIMS-1272: Currency in Price Lists
- LIMS-1310: Broken AnalysisProfile selector in AR Add form.

6.49 3.1.2 (2014-07-15)

- LIMS-1292: UI fix Retracted ARs workfow: Warning msg on “full” retract.
- LIMS.1287: UI fix Report parameter formatting
- LIMS-1230: UI fix Livesearch’s box
- LIMS-1257: UI fix Long titles in Analysis Profiles, Sample Points, etc.
- LIMS-1214: UI fix More columns
- LIMS-1199: UI fix Worksheet listing: better columns
- LIMS-1303: jsi18n strings must be added to bika-manual.pot. i18ndude cannot find.
- LIMS-1310: Filter SamplePoints by client in AR Template Edit View
- LIMS-1256: Client objects included in AR-Add filters for Sample Point etc.
- LIMS-1290: Allows Analyst to retract analyses, without giving extra permissions.
- LIMS-1218: Slightly nicer monkey patch for translating content object ID’s and titles.
- LIMS-1070: Accreditation text can be customised in bika_setup
- LIMS-1245: off-by-one in part indicators in ar_add
- LIMS-1240: Hide “copy to new” from Analyst users
- LIMS-1059: Added worksheet rejection workflow
- RejectAnalysis (Analysis subclass (has IAnalysis!)) workflow transition.
- Does not retract individual Analysis objects
- Sets attributes on src and dst worksheets:

- WS instance rejected worksheet attribute: .replaced_by = UID
- WS instance replacement worksheet attribute: .replaces_rejected_worksheet:UID
- Fixed some i18n and encoding snags, and updated translations.

6.50 3.1.1 (2014-06-29)

- Some bugs which only appear while running Windows, have been fixed.
- LIMS-1281: Fix Restricted and Default categories in ar_add
- LIMS-1275: Fix lax Aalyst permissions
- LIMS-1301: jsonapi can set ReferenceField=”“
- LIMS-1221: Icon for ARImports folder in Navigation
- LIMS-1252: AR Published Results Signature Block formatting
- LIMS-1297: Update frontpage

6.51 3.1 (2014-06-23)

- #oduct and Analysis specifications per AR
- Incorrect published results invalidation workflow
- Improved re-testing workflow
- Adjustment factors on worksheets
- Using ‘< n’ and ‘> n’ results values
- Sample Storage locations
- Sample Categories
- Analysis Prioritisation
- Bulk AR creation from file
- Results reports inclusion of relevant QC results
- Supply Inventory and Orders
- JSON interface
- Management Reports export to CSV
- Enhancements to AR Batching
- Enhancements to Results Reports
- Instrument management module
- Calibration certificates, maintenance, Instrument QC
- Method, Instrument and Analysis integrity
- Instrument import interface: Agilent MS ‘Masshunter Quant’
- Instrument import interface: Thermo Gallery
- Instrument import interface: Foss Winescan FT 120, Auto

- Invoices per AR, Analysis per Invoice line.
- Invoices per Supply Order, inventory item per Invoice line
- Invoices by email
- Invoice 'batches' for selected time period, ARs aand Orders per Invoice line
- Invoice batch export to accounts systems
- Price lists. Analysis Services and Supplies

6.52 3.1.3036 (2014-05-30)

- Added two checboxes in BikaSetup > Security:
- Allow access to worksheets only to assigned analysts (Y/N)
- Only lab managers can create and amange new worksheets (Y/N)

** IMPORTANT NOTES **

The 3036 upgrade sets the above options to true by default, so after being upgraded, only the labmanagers will be able to manage WS and the analysts will only have access to the worksheets to which they are assigned. These defaults can be changed in BikaSetup > Security.

6.53 3.0 (2014-03-15)

- Fix some out-dated dependencies that prevented the app from loading.
- Development of the current bika 3.0 code has slowed, and our efforts have been focused on the 3.01a branch for some time.

6.54 3.0rc3.5.1 (2013-10-25)

- Fix CSS AR Publication error
- Fix error displaying client sample views

6.55 3.0rc3.5 (2013-10-24)

- Requires Plone 4.3.
- Fix a serious error saving Analysis results.
- Improve upgrade handling in genericsetup profile
- Fix errors in setupdata loader
- Force UTF-8 encoding of usernames (imported client contacts can now login)
- Removed outdated test setup data
- Handle duplicate request values in bika_listing
- ID server handles changes in ID schemes without error

- Remove folder-full-view from front-page view
- Updated workflow and permissions to prevent some silly errors
- Add robot tests
- Add default robots.txt

6.56 3.0rc3.2 (2013-06-28)

- Fix site-error displaying upgraded instruments
- Fix spinner (KSS is not always enabled)
- Add extra save button in ar_add
- Label Printing: "Return to list" uses browser history
- Bold worksheet position indicators
- Remove version.txt (use only setup.py for version)

6.57 3.0rc3.1 (2013-06-27)

- Fix permission name in upgrade step

6.58 3.0rc3 (2013-06-25)

- Many instrument management improvements! (Merge branch 'imm')
- Removed ReferenceManufacturer (use of generic Manufacturer instead)
- Removed ReferenceSupplier (use Supplier instead)
- Improve service/calculation interim field widgets
- Allows service to include custom fields (without calculation selected)
- Fix services display table categorisation in Analysis Specification views
- Stop focusing the search gadget input when page load completes. (revert)
- Limit access to Import tab (BIKA: Manage Bika)
- New permission: "BIKA: Import Instrument Results"
- New permission: "BIKA: Manage Login Details" - edit contact login details
- Some late changes to better handle the updates to ID creation
- Plone 4.3 compatibility (incomplete)
- Use Collections as a base for Queries (incomplete)
- Many many bugfixes.

6.59 3.0rc2.3 (2013-01-29)

- Fix bad HTML

6.60 3.0rc2.2 (2013-01-28)

- Fix an error during AR Publish

6.61 3.0rc2.1 (2013-01-21)

- Fix bad HTML
- Pin collective.js.jqueryui version to 1.8.16.9

6.62 3.0rc2 (2013-01-21)

- Updated all translations and added Brazilian Portuguese
- RecordsWidget: subfield_types include “date”
- RecordsWidget: Automatic combogrid lookups
- Added all bika types to Search and Live Search
- Transition SamplePartition IDs to new format (SampleType-000?-P?
- Always handle non-ASCII characters: UTF-8 encoding everywhere
- Accept un-floatable (text) results for analyses
- Hidden InterimFields in Calculations
- Added InterimFields on AnalysisServices for overriding Calculation Interimfields.
- Disable KSS inline-validation
- Categorized analyses in AR views
- Added remarks for individual analyses
- Improved Javascript i18n handling
- Improved default permissions
- Added ‘Analysis summary per department’ (merge of ‘Analyses lab department weekly’ and ‘Analyses request summary by date range’
- Added ‘Analyses performed as % of total’ report
- Added Analyses per lab department report
- Added ‘Samples received vs. samples reported’ report
- Added Daily Samples Received report
- Many many bugfixes.

6.63 3.0rc1 (2012-10-01)

- Removed Bika Health data from released egg
- Remove remarks from portal_factory screens
- Add Month/Year selectors to default datetime widget
- ClientFolder default sorting.
- Date formats for jquery datepicker
- Don't overwrite the Title specified in @@plone-addsite
- Bug fixes

6.64 3.0rc1 (2012-09-25)

- Requires Python 2.7 (Plone 4.2)
- Add GNUPlot dependency
- Added client sample points
- Added Sampling Deviation selections
- Added Ad-Hoc sample flag
- Added Sample Matrices (Sampletype categorisation)
- Added custom ResultsFooter field in bika setup
- Added PDF Attachments to published results
- Electronic signature included in Results and Reports
- Login details form to create users for LabContacts
- Sampling workflow is disabled by default
- Methods are versioned by default
- Methods are publicly accessible by default
- Queries WIP
- Reports WIP
- Modified label layouts for easier customisation
- Cleaned print styles
- Use plonelocales for handling Date/Time formats
- SMS and Fax setup items are disabled by default

6.65 2012-06-21

- Partitioning & Preservation automation
- Reports
- Sample point & types relations in UI

- AR template enhancements
- Sample and AR layout improvements
- Labels
- Configuration logs
- Faster indexing
- JavaScript optimisation
- Better IE compatibility
- Set-up worksheet improvements
- Updated translations
- Workflow tweaks
- Tweaks to Icons, Views & Lists

6.66 2012-04-23

- Optional sampling and preservation workflows and roles.
- Sample partitioning.
- AR templates - Sample point & Sample type restrictions.
- Reports - framework only. 'Analysis per service' shows what is planned.
- Improved i18n handling, and updated strings from Transifex.
- Numerous performance enhancements
- Analysis Service & Method associations.
- An improved Analysis Service pop-up window.
- Sample Type and Sample Point relationship.
- Currency selection from zope locales
- Combined AR View and Edit tabs.
- Re-factored AR 'Add/Remove Analyses' screen
- Store the date of capture for analysis results
- Append only remarks fields on more objects.

6.67 2012-01-23

- Made Bika compatible with Plone 4.1
- Sampler and Preserver roles, users and permissions
- Sampling and Preservation workflows
- Inactive and Cancellation Workflows
- #e-preserved Containers
- Automatic versioning for some bika_setup types

- Analyst and Instrument on Worksheet templates
- XLSX setup data loader
- Sample disposal date based on date sampled, not date received.
- Internal ID Server by default
- user defined calculations and interim fields
- Dry Matter results option does not appear until enabled in Site Setup
- Accreditation portlet disabled until enabled in Site Setup
- BikaListingView
- New icons
- (mostly) usable at 800x600
- Column display toggles
- Future dated samples and ARs
- Accreditation template: i18n in locales/manual.pot/accreditation_*
- intermediate workflow state for analyses requiring attachments
- Labmanager has Site Administrator role (not Manager)
- 'Indeterminate' results
- use portal_factory everywhere
- working test suite
- static resource directories
- Merged BikaMembers types
- CoordinateField/Widget
- DurationField/Widget
- CustomRecordsWidget

6.68 2.3.3 Bug fix release

- Inclusion of BikaMembers 0.0.3. No changes to bika code, version bumped to facilitate release of new BikaMembers version.

6.69 2.3

- Analysis categories introduced
- Analysis service result restrictions - specification of possible results
- Allow site and client specification of email and fax subject line content
- Additional instrument/export formats: WinescanFT120, WinescanAuto, FIAStar and Bartelt's data-collector
- Export worksheet analyses to instruments
- PDF as a result output option

- SMS result output option
- Result publication options synchronized and signatures added to emails
- Email batching of query results conforms to result mailing
- IDServer batching of unique id request
- Optmization of worksheet searching on selection criteria
- Extract tab added with extract for analysis services or profiles
- Batch update of analysis service prices
- German translation module added
- Added a light query form which excludes analysis category and service
- Batch size setting in analysis request lists
- BikaMembers replaces UpfrontContacts
- ATSchemaEditor removed
- Significant performance improvements
- Resolve client action conflicts
- Sampled date validation
- Drymatter formatting on output corrected
- Correct default none workflows
- Review portlet optimization
- #icelist prints blank for analysis service with price not defined

6.70 2.2

- Attachments permitted on analysis requests and analyses
- Worksheet resequencing, and sort order for worksheet analysis selection
- Worksheet deletion only available for open worksheets
- Portlet to provide export of analysis services and analysis profiles
- Requirement for unique analysis service names, analysis service keywords,
- instrument import keywords and analysis profile keywords enforced.
- Report headings and formats standardized accross different reports
- AR import alternative layout provided with selection, including profiles
- #ogress bar introduced for long running processes

6.71 2.1.1

- Disposal Date for Samples and Retention Period per Sample Type added.
- Various new search criteria added.
- Standard Manufacturers introduced.

- Labels for Standard Samples introduced.
- "Print" and "Email" facilities introduced for lists of Standard Samples and Standard Stocks.
- "Duplicate" facility for Analysis Services introduced.
- Addresses added to top of emailed query results.
- Labels for Samples and Analysis Requests changed.
- Analysis Services can have multiple Methods.
- Change log introduced for Methods.
- Methods added to left navigation bar.
- List of Methods included in pop-up for Analyses.
- Documents may be uploaded for Methods.

6.72 2.1

- Sample object and workflow introduced
- Results specifications, lab and per client
- Analysis profiles
- Worksheet template engine
- Interface to Bika Calendar
- Import of analysisrequests from csv file
- Export of results to csv file
- #int as publication option
- Lab Departments, lab contacts, and department manager introduced
- Quality Control calculations. Control, blank and duplicate analyses.
- QC graphs, normal distribution, trends and duplicate variation
- Various analysis calculations allowed. Described by Calculation Type
- Dependant Calcs introduced. Where an analysis result is calculated from
- other analyses: e.g. AnalysisX = AnalysisY - Analysis Z
- Dry matter result reporting. Results are reported on sample as received, and also as dry matter result on dried sample
- Re-publication, Pre publication of individual results and per Client
- Many reports including Turn around, analyses repeated and out of spec

6.73 1.2.1

- Removed invoice line item descriptions from core code to allow skin integration
- Create dummy titration values for analyses imported from instrument
- More language translations

6.74 1.2.0

- Statements renamed to Invoices
- Jobcards renamed to Worksheets
- New identification fields added to analysis request
- Client Reference, Sample Type and Sample Point
- Welcome page introduced
- Late analyses list linked from late analyses portlet
- Icon changes
- Accreditation body logo and details added to laboratory info
- Accreditation logo, disclaimers added throughout web site
- Laboratory confidence level value data driven from laboratory info
- Analyses methods provided as pop-up where analyses are listed
- Titration factors and titration volumes added to analyses and worksheets
- Measure of uncertainties introduced per analysis and intercept
- Two new specialist roles created - verifier and publisher
- Sample test data load script - load_sample_data.py
- Implement generic instrument data import tool
- Login portlet added
- Modifications required to support interlab
- Permit analysis parent (sample) to be in 'released' state.
- Reference SampleID on AnalysisRequest-
- 1566324: Logged in page redirected to welcome page.
- 1573299: LiveSearch - Added permissions to InvoiceLineItem.
- 1573083: Status Drop Down - Invoicing
- 1551957: Contacts not visible to other contacts. Correct local owner role
- 1566334: position of 'add new ar' button changed to conform to other forms
- 1532008: query results sort order most recent first
- 1532770: Order default listing correction
- 1558458: Member discount data driven in messages on AR forms
- 1538354: SubTotal and VAT calculation on edit AR
- 1532796: AR edit - allow change of contact

6.75 1.1.3

This is a bug fix release. Migration from older versions has also been improved greatly.

Please note that AnalysisRequest now has a custom mutator that expects the title of the Cultivar, not the UID. This will impact anybode that customised the *analysisrequed_add.cpy* controller script and the *validate_analysisrequest_add_form.vpy* validation script.

- 1423182: IndexError on surfing to LIMS pages without being logged on
- 1423238: Orders - Dispatch date
- 1429992: AR edit tab - Cultivar uneditable
- 1429996: Cultivar names to allow numbers
- 1429999: Late analysis alert - 'More…' URL
- 1430002: Sample due alerts - 'More…' URL
- 1433787: Security - Clients
- 1434100: Search - Index & Attribute errors
- 1418473: Updated start-id-server.bat for Win2K & Win XP

6.76 1.1.2

- 1423205: Show logs to labmanager set-up
- 1291750: Added default ID prefixes for Order and Statement
- 1424589: Late analysis alert to be calulated on date received

6.77 1.1.1

- Updated portlets with Plone 2.1 style definition list markup
- 1423179: Clients must not see JobCard links on Analysis Requests
- 1423182: IndexError on surfing to LIMS pages without being logged on
- 1423188: Site map - Clients should not have access to …
- 1423191: Link rot - 'logged in' page
- 1423193: Groups folder should not be shown
- 1423194: No 'More…' if there are less than 5
- 1423204: AR view - Missing tabs and status drop down
- 1423209: Schema Editor - Drop Down List Issue (Select)
- 1423234: Late Analysis alert shows for anonymous visitors
- 1423363: Report Analysis Totals
- 1423386: Email publication error

6.78 1.1.0

- Made Bika compatibable with Plone 2.1
- Added Spanish translation contributed by Luis Espinoza
- Added Italian translation contributed by Pierpaolo Baldan
- Added Dutch translation contributed by Joris Goudriaan
- Added Portugese translation contributed by Nuno R. Pinhão
- The schemas of Client, Contact, AnalysisRequest and Order can be edited in the through-the-web schema editor, ATSchemaEditorNG.
- The maximum time allowed for the publication of results can now be set per analysis service. The portlet 'skins/bika/portlet_late_analysis.pt' has been added to alert lab users when analyses are late.
- Analyses on an AnalysisRequest have a reference to a Jobcard, rendered as a hyperlink on the AnalysisRequest view.
- A bug has been fixed where 'not_requested' analyses were checked on the AnalysisRequest edit form.
- Enabled 'changed_state' folder button globally and disabled on AnalysisRequest and Jobcard.

6.79 1.0.1

- Updated 'skins/bika/date_components_support.py' with latest version of script in Plone 2.0.5
- Modified access to transitions in workflow scripts, normal attribute access seems to guarded since Zope 2.7.5.
- Added CHANGES.txt and README.txt
- Added windows batch script for ID server (scripts/start-id-server.bat)