Wicket Viewer

Table of Contents

Chapter 1. Wicket Viewer

The Wicket Viewer automatically exposes an Apache Isis domain object model for use by end-users. The viewer is implemented using Apache Wicket.

This user guide discuss end-user features, configuration and customization of the Wicket viewer.

It also discusses how to extend the viewer, and the (non-ASF) Isis Addons wicket components.

1.1. Other Guides

Apache Isis documentation is broken out into a number of user and reference guides.

The user guides available are:

- Fundamentals
- Wicket viewer (this guide)
- Restful Objects viewer
- DataNucleus object store
- Security
- Testing
- Beyond the Basics

The reference guides are:

- Annotations
- Domain Services
- Configuration Properties
- Classes, Methods and Schema
- Apache Isis Maven plugin
- Framework Internal Services

The remaining guides are:

- Developers' Guide (how to set up a development environment for Apache Isis and contribute back to the project)
- Committers' Guide (release procedures and related practices)

Chapter 2. Features/end-user usage

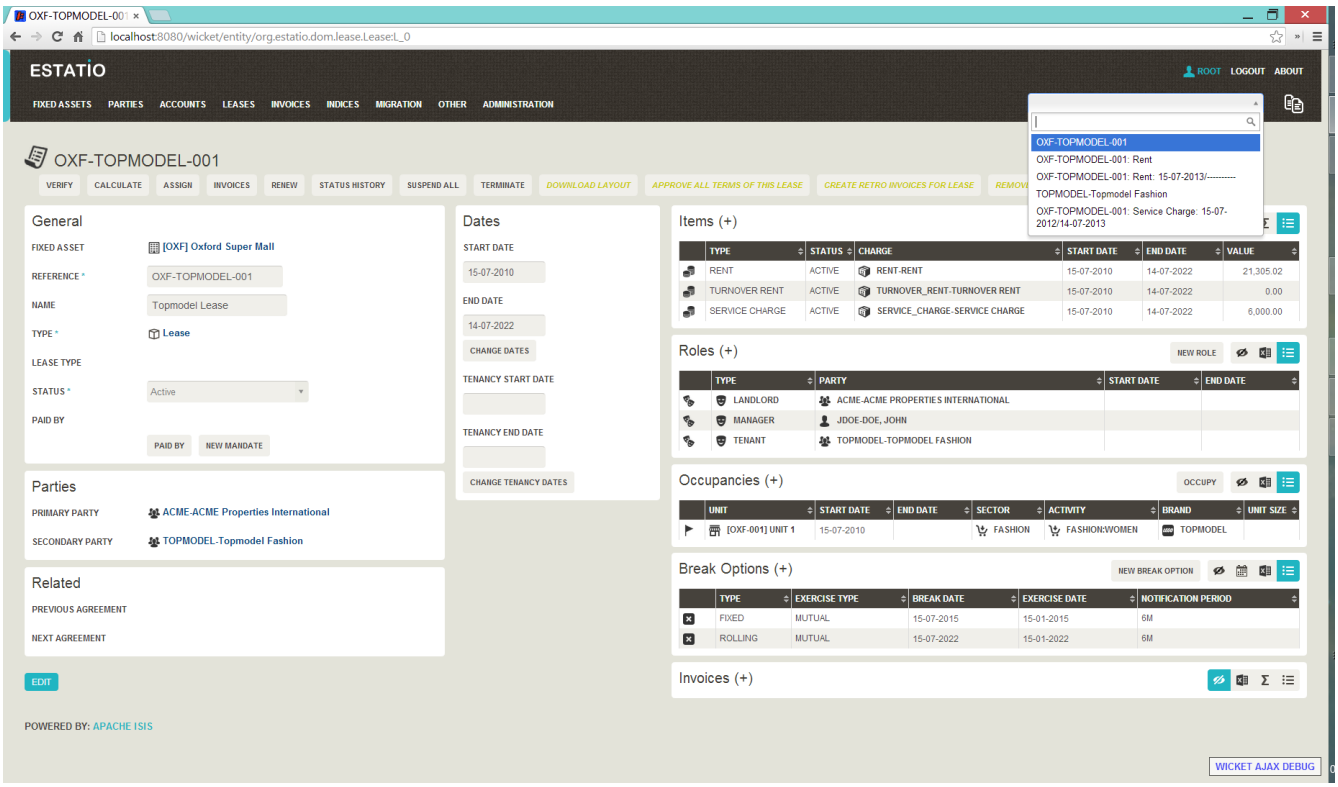
This section discusses features of the wicket viewer from the perspective of an end-user actually using your Isis application.

2.1. Recent pages (drop down)

The Wicket viewer provides a recent pages drop-down that acts as a breadcrumb trail. Using it, the user can quickly open a recently accessed domain object.

2.1.1. Screenshots

The following screenshot, taken from the Estatio application, shows the recent pages drop-down after a number of pages have been accessed.



Note that this screenshot show an earlier version of the Wicket viewer UI (specifically, pre 1.8.0).

2.1.2. Domain Code

The recent pages drop-down is automatically populated; no changes need to be made to the domain classes.

2.1.3. User Experience

Selecting the domain object from the list causes the viewer to automatically navigate to the page for the selected object.

2.1.4. Related functionality

The bookmarked pages (sliding panel) also provides links to recently visited objects, but only those explicitly marked as `@DomainObject(bookmarking=⋯)`. The bookmarks panel also nests related objects together hierarchically (the recent pages drop-down does not).

2.1.5. Configuration

The number of objects is hard-coded as 10; it cannot currently be configured.

2.2. Bookmarked pages

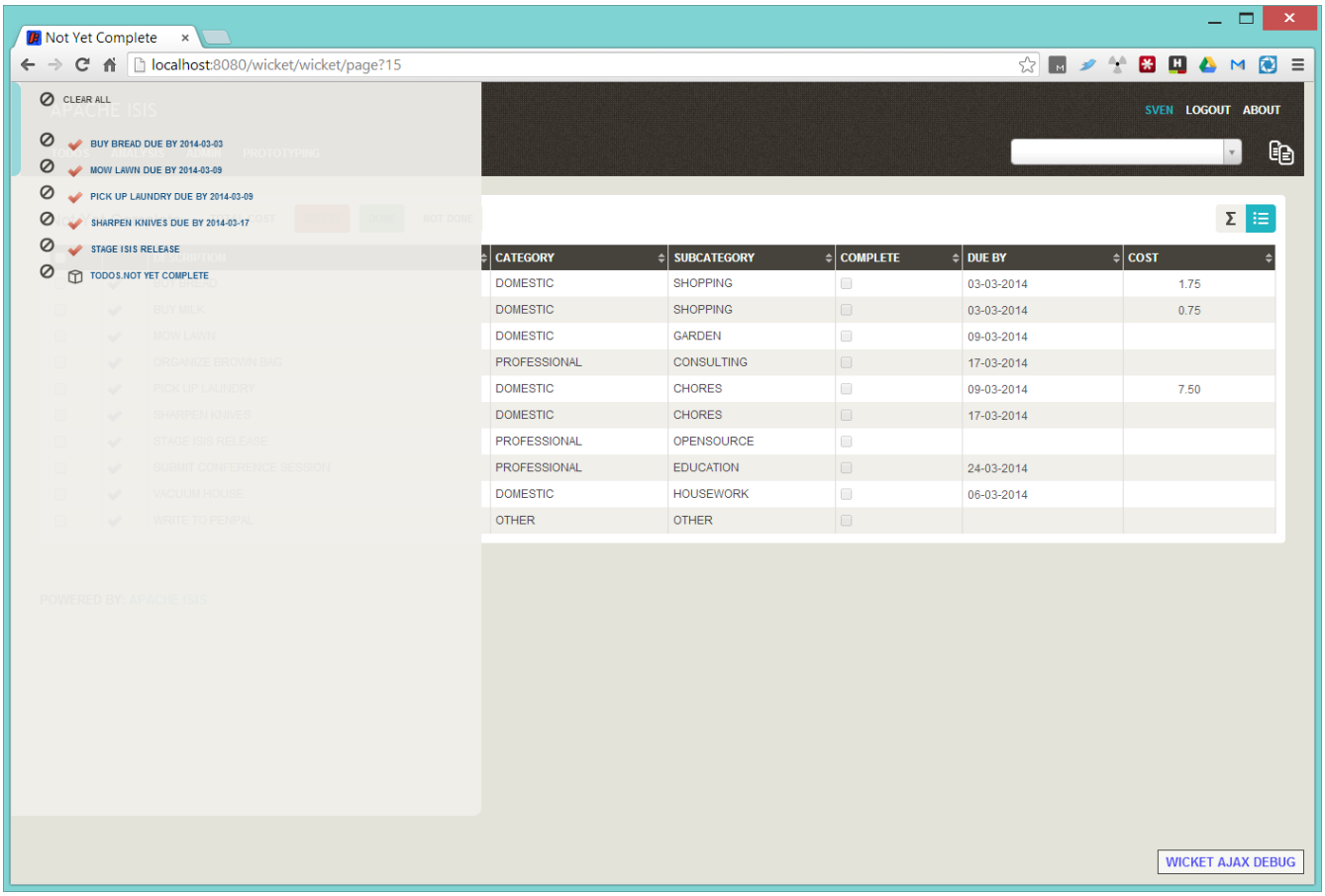
The Wicket viewer supports the bookmarking of both domain objects and query-only (@Action(semantics=...)) actions.

Domain objects, if bookmarkable, can be nested.

Bookmarking is automatic; whenever a bookmarkable object/action is visited, then a bookmark is created. To avoid the number of bookmarks from indefinitely growing, bookmarks that have not been followed after a whle are automatically removed (an MRU/LRU algorithm). The number of bookmarks to preserve can be configured.

2.2.1. Screenshots

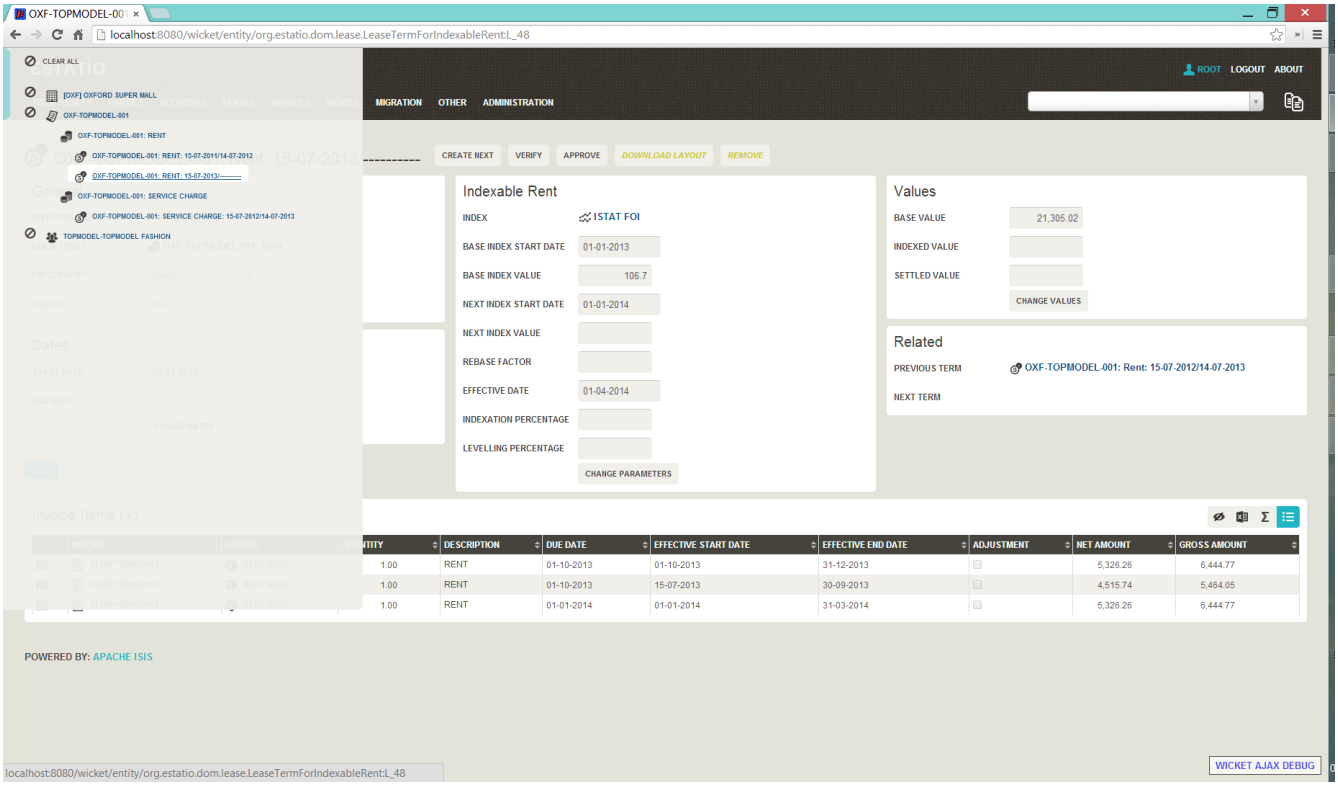
The following screenshot, taken from Isisaddons example todoapp (not ASF) shows how the bookmarks are listed in a sliding panel.



Note that these screenshots show an earlier version of the Wicket viewer UI (specifically, pre 1.8.0).

Note how the list contains both domain objects and an action ("not yet complete").

Bookmarks can also form a hierarchy. The following screenshot, also taken from the Estatio application, shows a variety of different bookmarked objects with a nested structure:



Some - like `Property`, `Lease` and `Party` - are root nodes. However, `LeaseItem` is bookmarkable as a child of `Lease`, and `LeaseTerm` is bookmarkable only as a child of `LeaseItem`. This parent/child relationship is reflected in the layout.

2.2.2. Domain Code

To indicate a class is bookmarkable, use the `@DomainObjectLayout` annotation:

```
@DomainObjectLayout(
    bookmarking=BookmarkPolicy.AS_ROOT
)
public class Lease { ... }
```

To indicate a class is bookmarkable but only as a child of some parent bookmark, specify the bookmark policy:

```
@DomainObjectLayout(
    bookmarking=BookmarkPolicy.AS_CHILD
)
public class LeaseItem { ... }
```

To indicate that a safe (query only) action is bookmarkable, use the @ActionLayout annotation:

```
public class ToDoItem ... {
    @Action(
        semantics=SemanticsOf.SAFE
    )
    @ActionLayout(
         bookmarking=BookmarkPolicy.AS_ROOT
     )
    public List<ToDoItem> notYetComplete() { ... }
    ...
}
```

The BookmarkPolicy.AS_CHILD does not have a meaning for actions; if the `bookmarking` attribute is set to any other value, it will be ignored.

2.2.3. User Experience

The sliding panel appears whenever the mouse pointer hovers over the thin blue tab (to the left of the top header region).

Alternatively, `alt+[` will toggle open/close the panel; it can also be closed using `Esc` key.

Related functionality

The Recent Pages also lists recently visited pages, selected from a drop-down.

2.2.4. Configuration

By default, the bookmarked pages panel will show a maximum of 15 'root' pages. This can be overridden using a property (in `isis.properties`), for example:

```
isis.viewer.wicket.bookmarkedPages.maxSize=20
```

2.3. Hints and copy URL

While the user can often copy the URL of a domain object directly from the browser's address bar, the Wicket viewer also allows the URL of domain objects to be easily copied from a dialog.

More interestingly, this URL can also contain hints capturing any sorting or page numbering, or hiding/viewing of collections. End-users can therefore share these URLs as a form of deep linking

into a particular view on a domain object.

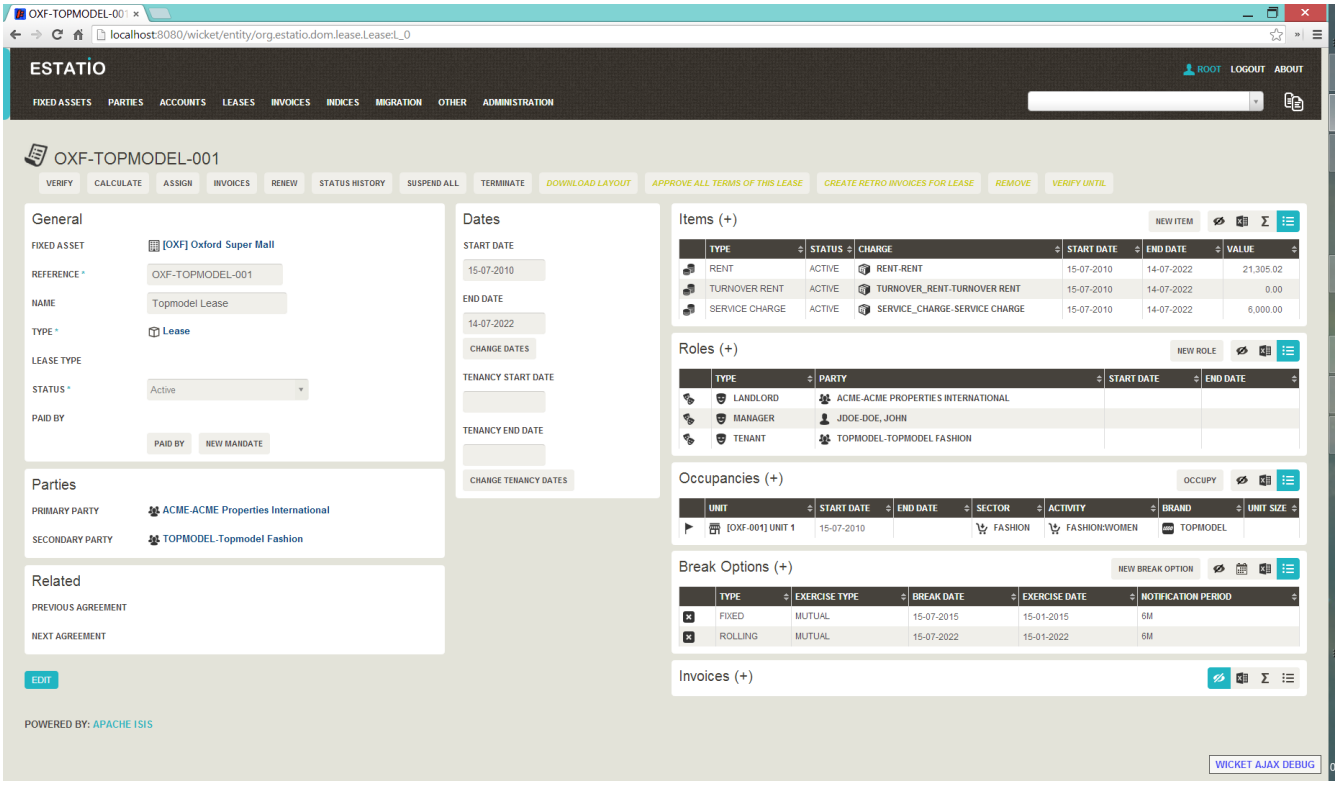
The copy URL and hinting is automatic.

2.3.1. Screenshots

The following screenshots are taken from the [Estatio](https://github.com/estatio/estatio) application.

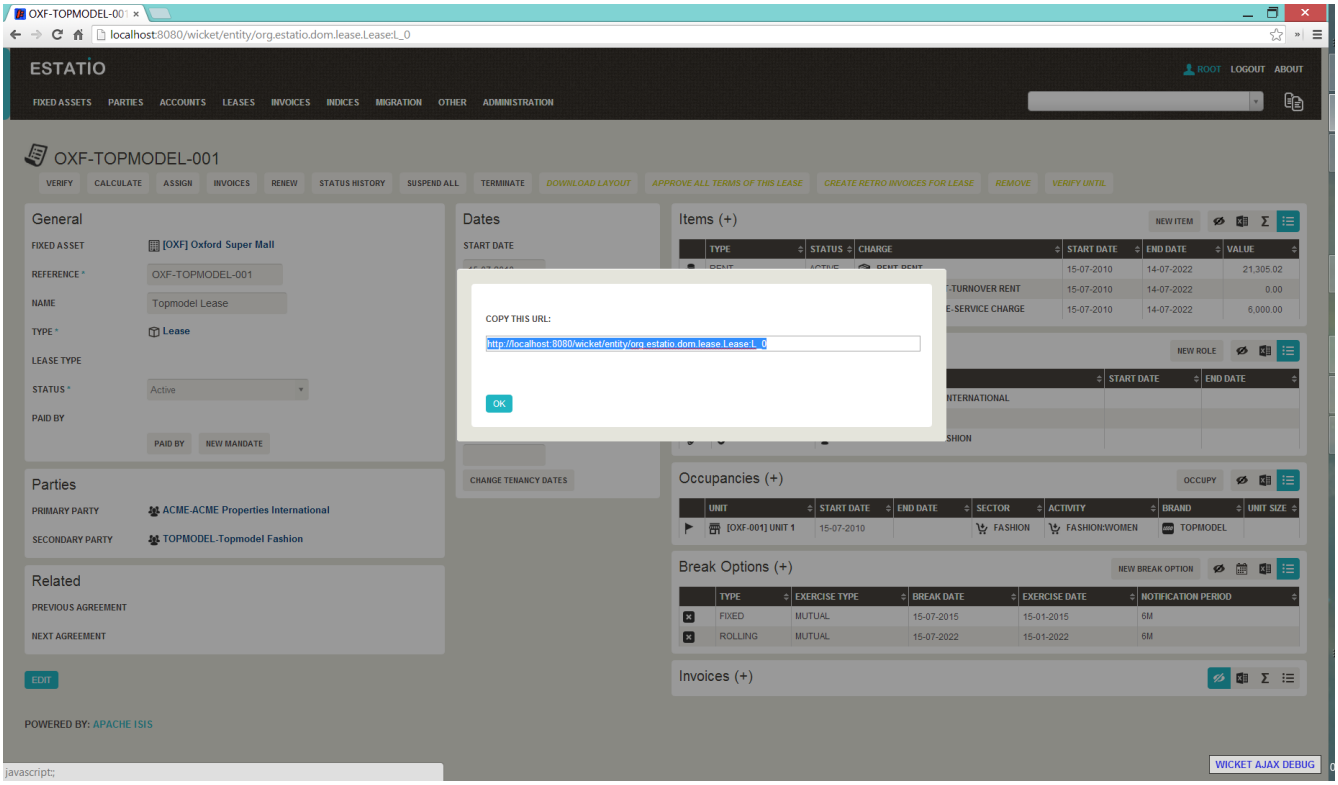
Copy URL

This screenshot shows the copy URL button (top right):



Note that these screenshots show an earlier version of the Wicket viewer UI (specifically, pre 1.8.0).

Clicking on this button brings up a dialog with the URL preselected:



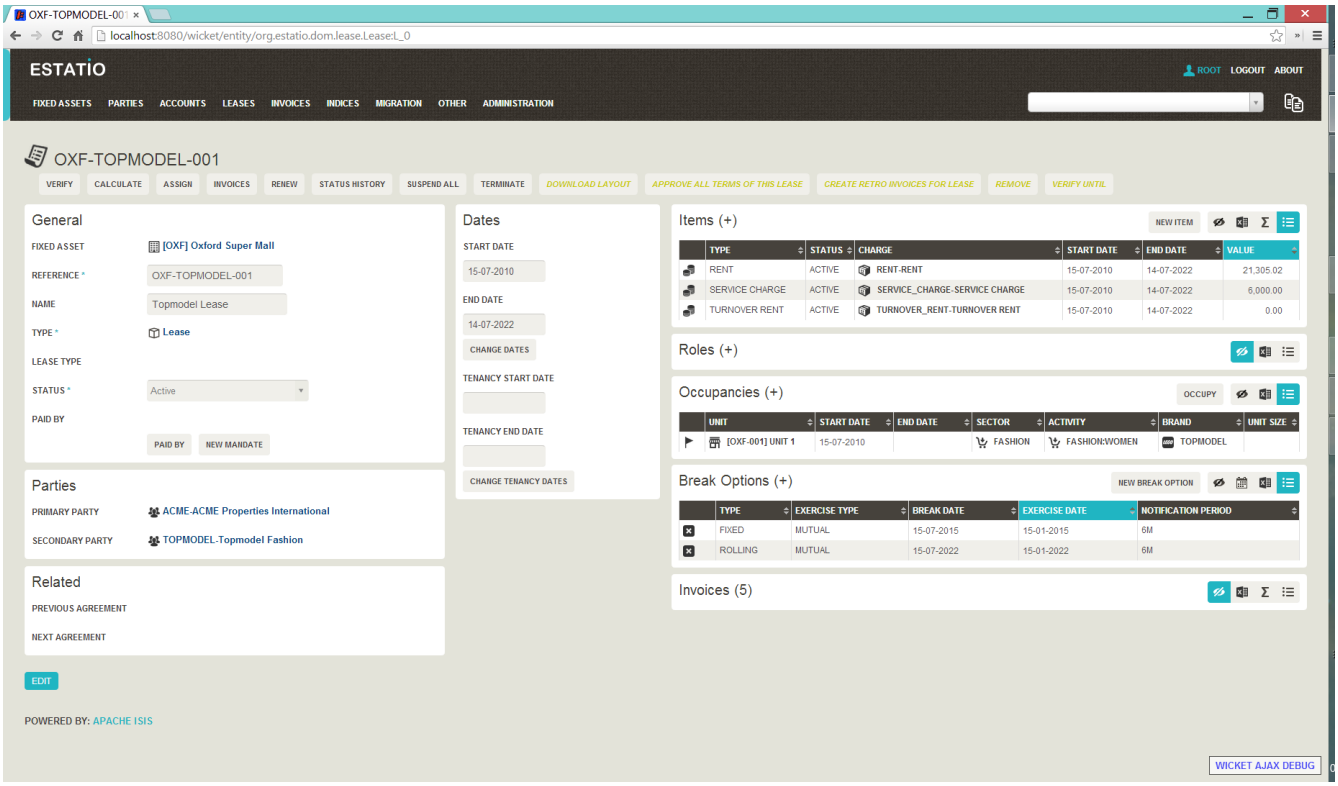
The URL in this case is something like:

```
http://localhost:8080/wicket/entity/org.estatio.dom.lease.Lease:0
```

The user can copy the link (eg `ctrl+C`) into the clipboard, then hit `OK` or `Esc` to dismiss the dialog.

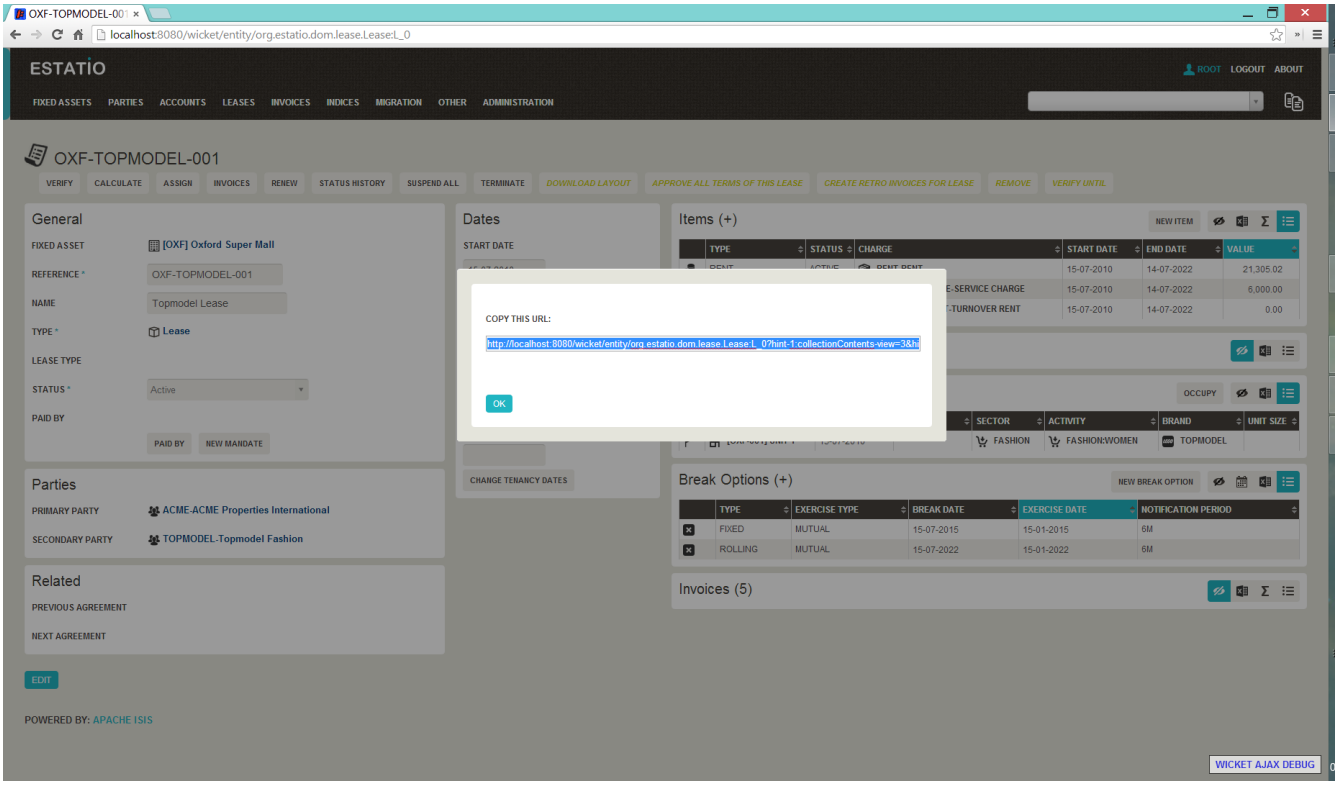
Hints

Using the viewer the user can hide/show collection tables, can sort the tables by header columns:



Also, if the collection spans multiple pages, then the individual page can be selected.

Once the view has been customised, the URL shown in the copy URL dialog is in an extended form:



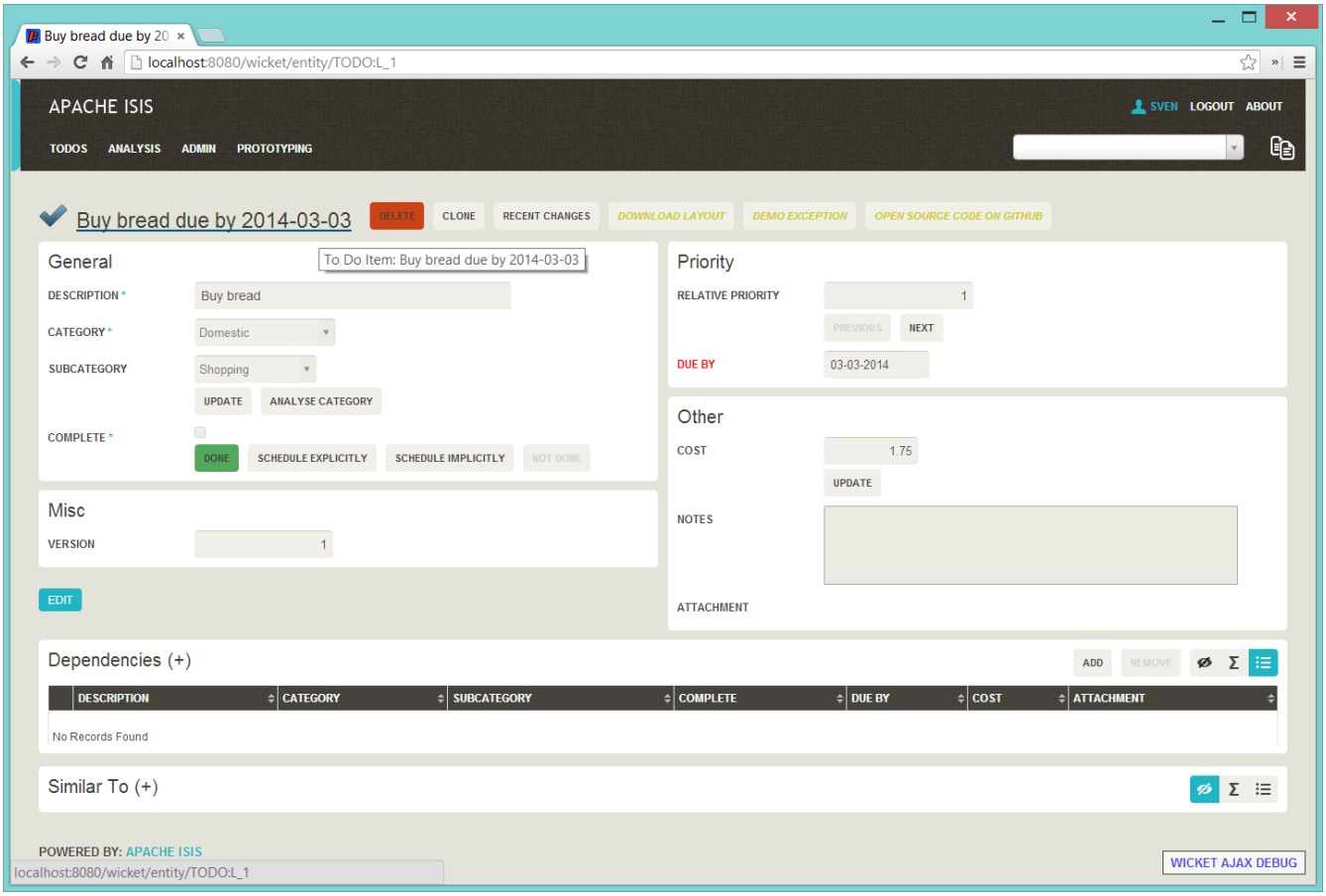
The URL in this case is something like:

```
http://localhost:8080/wicket/entity/org.estatio.dom.lease.Lease:0?hint-
1:collectionContents-view=3&hint-1:collectionContents:collectionContents-3:table-
DESCENDING=value&hint-1:collectionContents:collectionContents-3:table-
pageNumber=0&hint-2:collectionContents-view=0&hint-
2:collectionContents:collectionContents-2:table-pageNumber=0&hint-
3:collectionContents-view=2&hint-3:collectionContents:collectionContents-2:table-
pageNumber=0&hint-4:collectionContents-view=3&hint-
4:collectionContents:collectionContents-3:table-ASCENDING=exerciseDate&hint-
4:collectionContents:collectionContents-3:table-pageNumber=0&hint-
5:collectionContents-view=0&hint-5:collectionContents:collectionContents-3:table-
pageNumber=0
```

Copy URL from title

When the user invokes an action on the object, the URL (necessarily) changes to indicate that the action was invoked. This URL is specific to the user's session and cannot be shared with others.

A quick way for the user to grab a shareable URL is simply by clicking on the object's title:



2.3.2. User Experience

The copy URL dialog is typically obtained by clicking on the icon.

Alternatively, `alt+]` will also open the dialog. It can be closed with either `OK` or the `Esc` key.

2.4. Titles in Tables

Object titles can often be quite long if the intention is to uniquely identify the object. While this is appropriate for the object view, it can be cumbersome within tables.

If an object's title is specified with from `@Title` annotation then the Wicket viewer will (for parented collections) automatically "contextualize" a title by excluding the part of the title corresponding to a reference to the owning (parent) object.

In other words, suppose we have:

so that `Customer` has a collection of `Order`s:

```
public class Customer {
    public Set<Order> getOrders() { ... }
    ...
}
```

and `Product` also has a collection of `Order`s (please forgive the suspect domain modelling in this example (!)):

```
public class Product {
    public Set<Order> getOrders() { ... }
    ...
}
```

and where the `Order` class references both `Customer` and `Product`.

The `Order's might involve each of these:

```
public class Order {
    @Title(sequence="1")
    public Customer getCustomer() { ... }
    @Title(sequence="2")
    public Product getProduct() { ... }
    @Title(sequence="3")
    public String getOtherInfo() { ... }
    ...
}
```

In this case, if we view a `Customer` with its collection of `Order`s, then in that parented collection's table the customer's property will be automatically excluded from the title of the `Order` (but it would show the product). Conversely, if a `Product` is viewed then its collection of `Order`s would suppress product (but would show the customer).

This feature is a close cousin of the `@PropertyLayout(hidden=Where.REFERENCES_PARENT)` annotation, which will cause the property itself to be hidden as a column in the table. An Isis idiom is therefore:

```
public class Order {
    @Title(sequence="1")
    @PropertyLayout(hidden=Where.REFERENCES_PARENT)
    public Customer getCustomer() { ... }
    ...
}
```

The above annotations mean that titles usually "just work", altering according to the context in

which they are viewed.

It is also possible to configure the Wicket viewer to abbreviate titles or suppress them completely.

2.5. File upload/download

The Isis application library provides the Blob value type (binary large objects) and also the Clob value type (character large object), each of which also includes metadata about the data (specifically the filename and mime type).

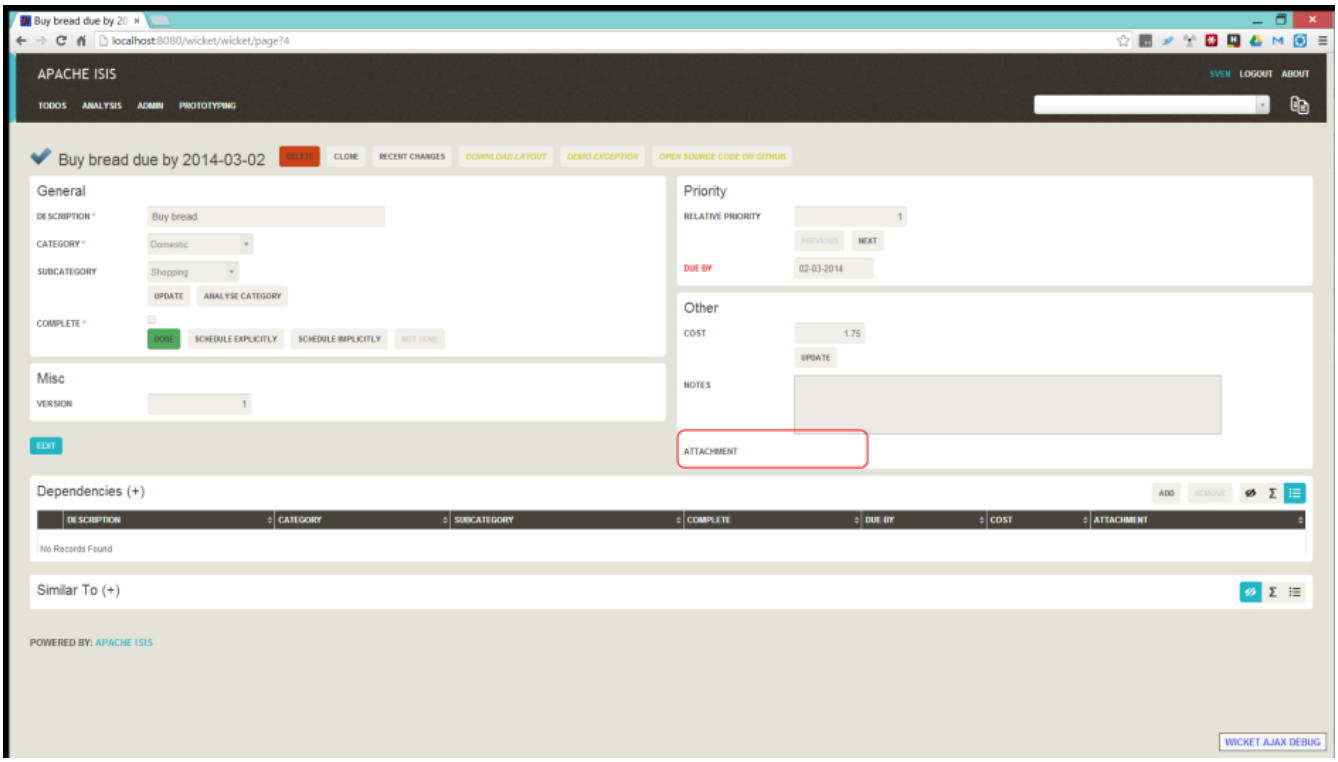
A class can define a property using either of these types, for example:

2.5.1. Screenshots

The following screenshots are taken from the Isis addons example todoapp (not ASF):

View mode, empty

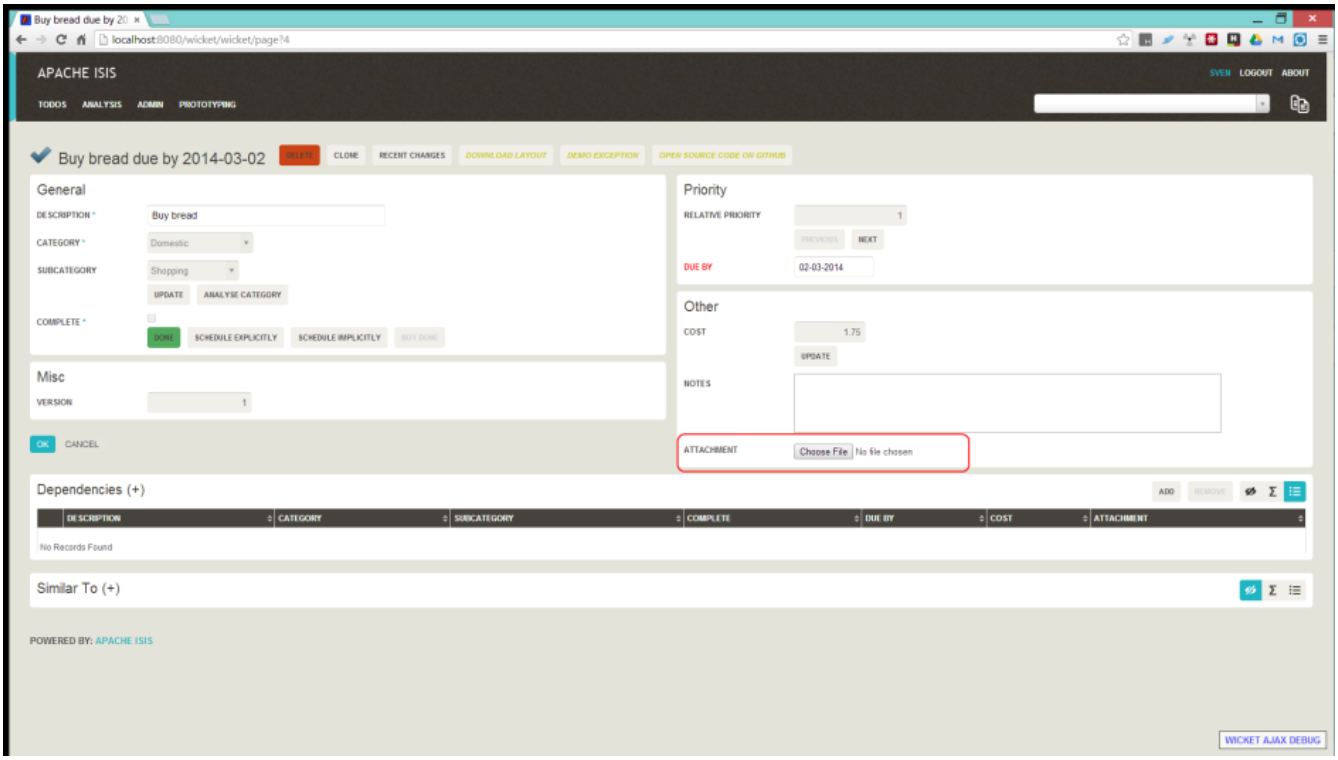
`Blob` field rendered as attachment (with no data):



Note that these screenshots show an earlier version of the Wicket viewer UI (specifically, pre 1.8.0).

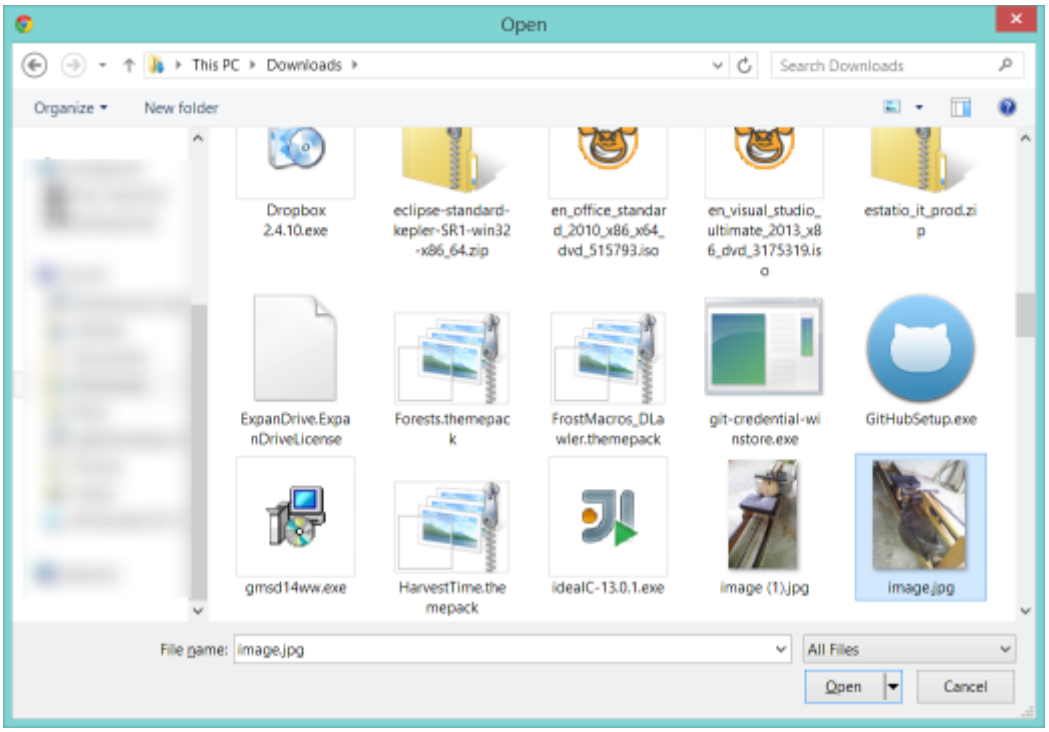
Edit mode

Hit edit; 'choose file' button appears:



Choose file

Choose file using the regular browser window:



Chosen file is indicated:

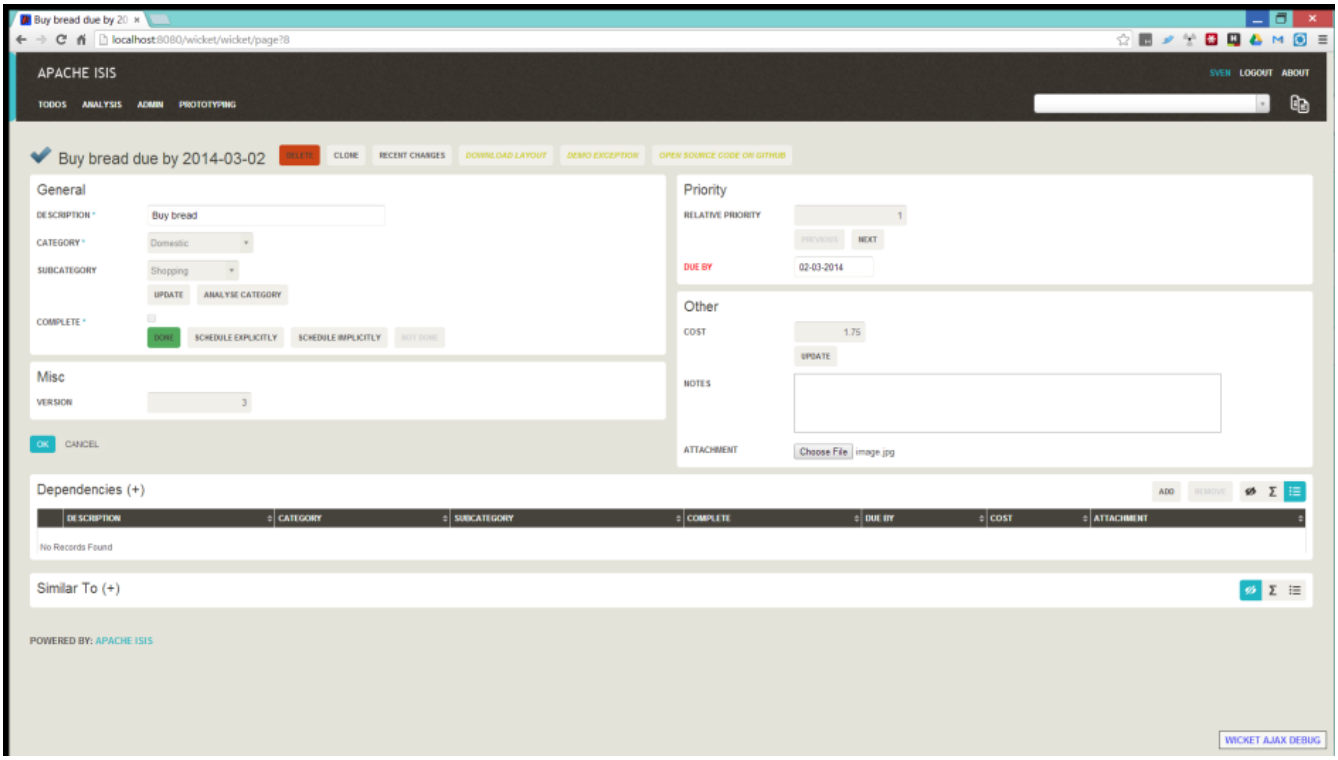
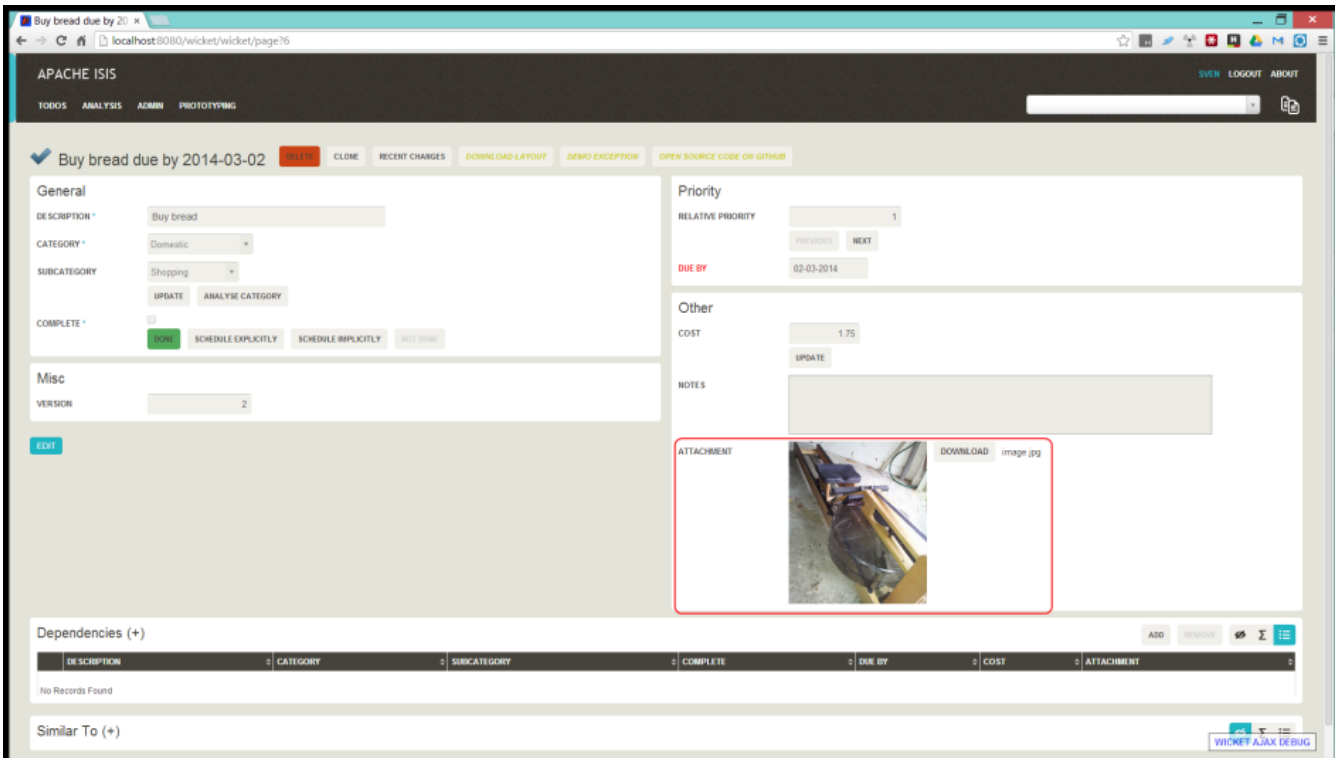


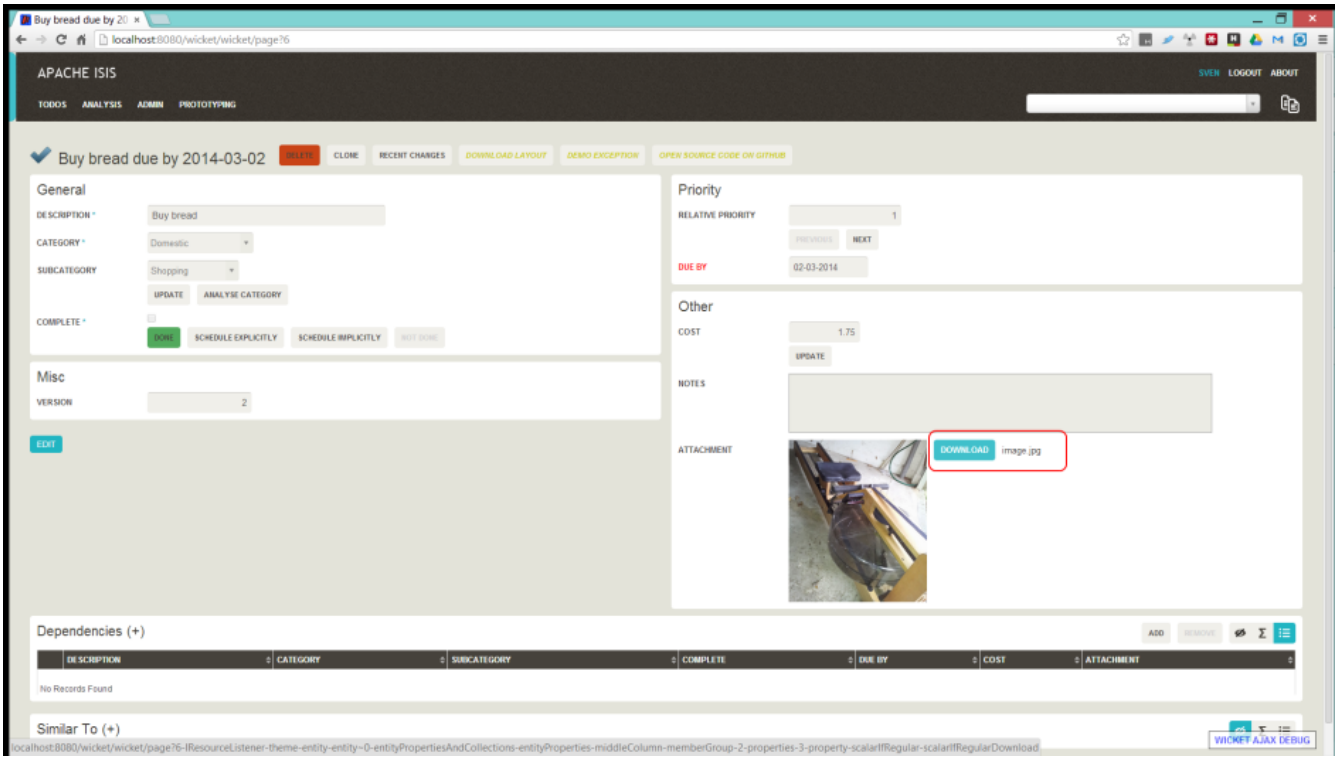
Image rendered

Back in view mode (ie once hit OK) if the `Blob` is an image, then it is shown:



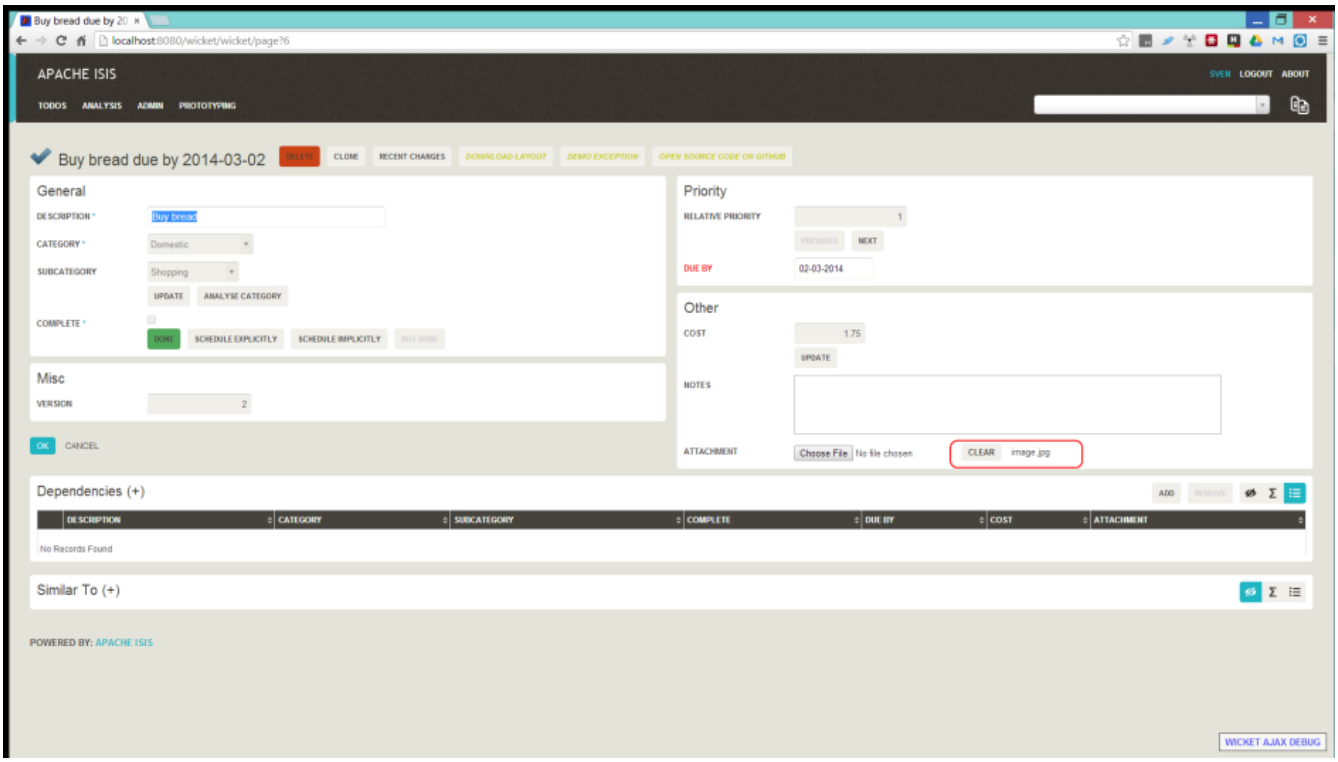
Download

`Blob` can be downloaded:



Clear

Back in edit mode, can choose a different file or clear (assuming property is not mandatory):



2.5.2. Domain Code

To define a `Blob`, use:

```
private Blob attachment;
@javax.jdo.annotations.Persistent(defaultFetchGroup="false")
    @javax.jdo.annotations.Persistent(defaultFetchGroup="false", columns = {
            @javax.jdo.annotations.Column(name = "attachment_name"),
            @javax.jdo.annotations.Column(name = "attachment_mimetype"),
            @javax.jdo.annotations.Column(name = "attachment_bytes", jdbcType =
"BLOB", sqlType = "BLOB")
    })
@Property(
        domainEvent = AttachmentDomainEvent.class,
        optionality = Optionality.OPTIONAL
)
public Blob getAttachment() { return attachment; }
public void setAttachment(final Blob attachment) { this.attachment = attachment; }
```

To define a `Clob`, use:

```
private Clob doc;
@javax.jdo.annotations.Persistent(defaultFetchGroup="false", columns = {
        @javax.jdo.annotations.Column(name = "doc_name"),
        @javax.jdo.annotations.Column(name = "doc_mimetype"),
        @javax.jdo.annotations.Column(name = "doc_chars", jdbcType = "CLOB", sqlType =
"CLOB")
})
@Property(
        optionality = Optionality.OPTIONAL
)
public Clob getDoc() { return doc; }
public void setDoc(final Clob doc) { this.doc = doc; }
```

The `Blob` and `Clob` types can also be used as parameters to actions.

2.6. User Registration

The Wicket viewer provides the ability for users to sign-up by providing a valid email address:

- from the login page the user can instead follow a link to take them to a sign-up page, where they enter their email address.
- a verification email is sent using this service; the email includes a link back to the running application.
- the user then completes the registration process by choosing a user name and password.
- the Wicket viewer then creates an account for them and logs them in.

In a similar way, if the user has forgotten their password then they can request a reset link to be sent to their email, again by providing their email address.

To support this the framework requires three services to be registered and configured:

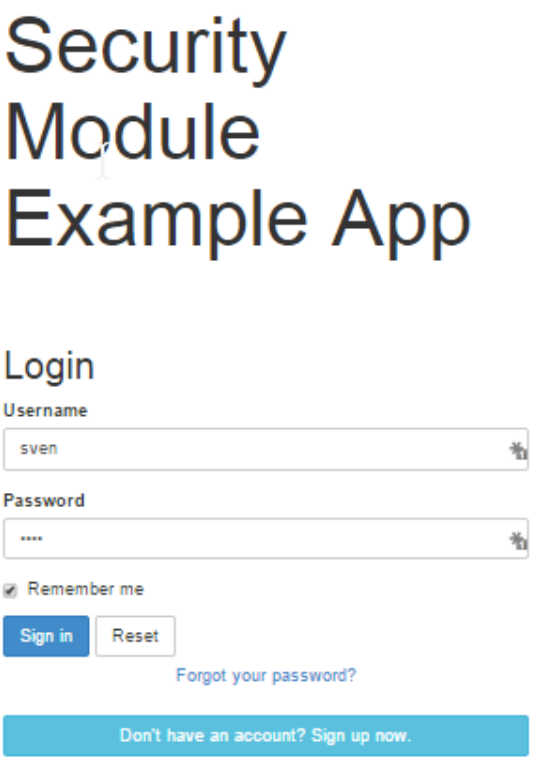
- the user registration service, which provides an API to create the user account
- the email notification service, which provides an API for to send the verification emails
- the email service, that is used by the email notification service to actually send the email.

The Apache Isis core framework provides a default implementation of both the email notification service and the email service. If your application uses the Isis addons security module (not ASF) then an implementation is provided by that module; just add to the classpath. Otherwise you will need to provide your own implementation.

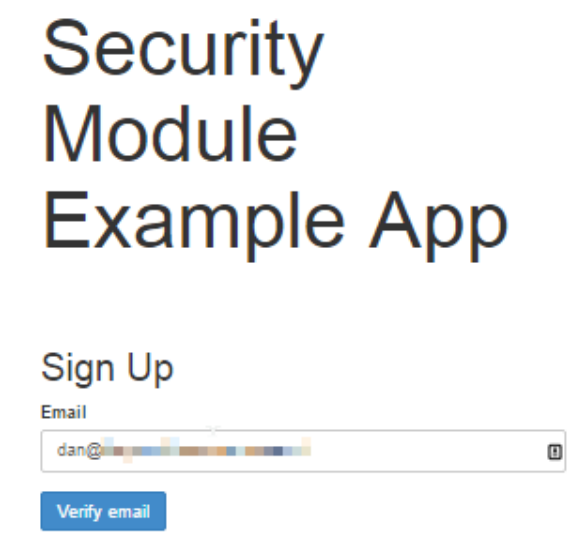
There is *no* default implementation of the user registration service in the core framework.

2.6.1. Screenshots

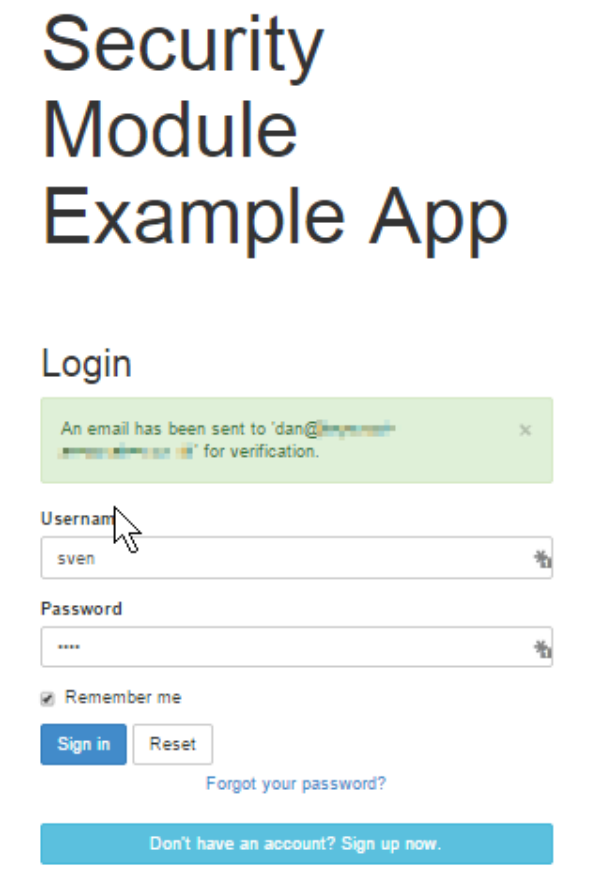
The user is presented with a login page:



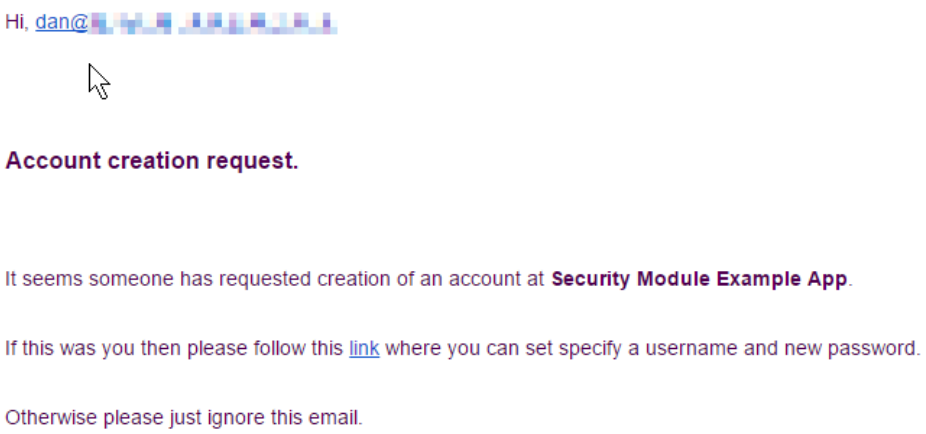
Navigate to the sign up page. Complete the page, and verify:



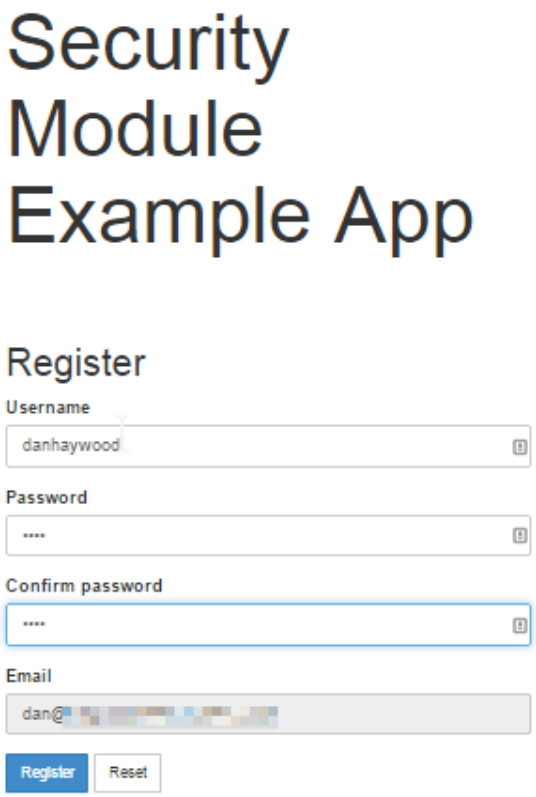
Back to the login page:



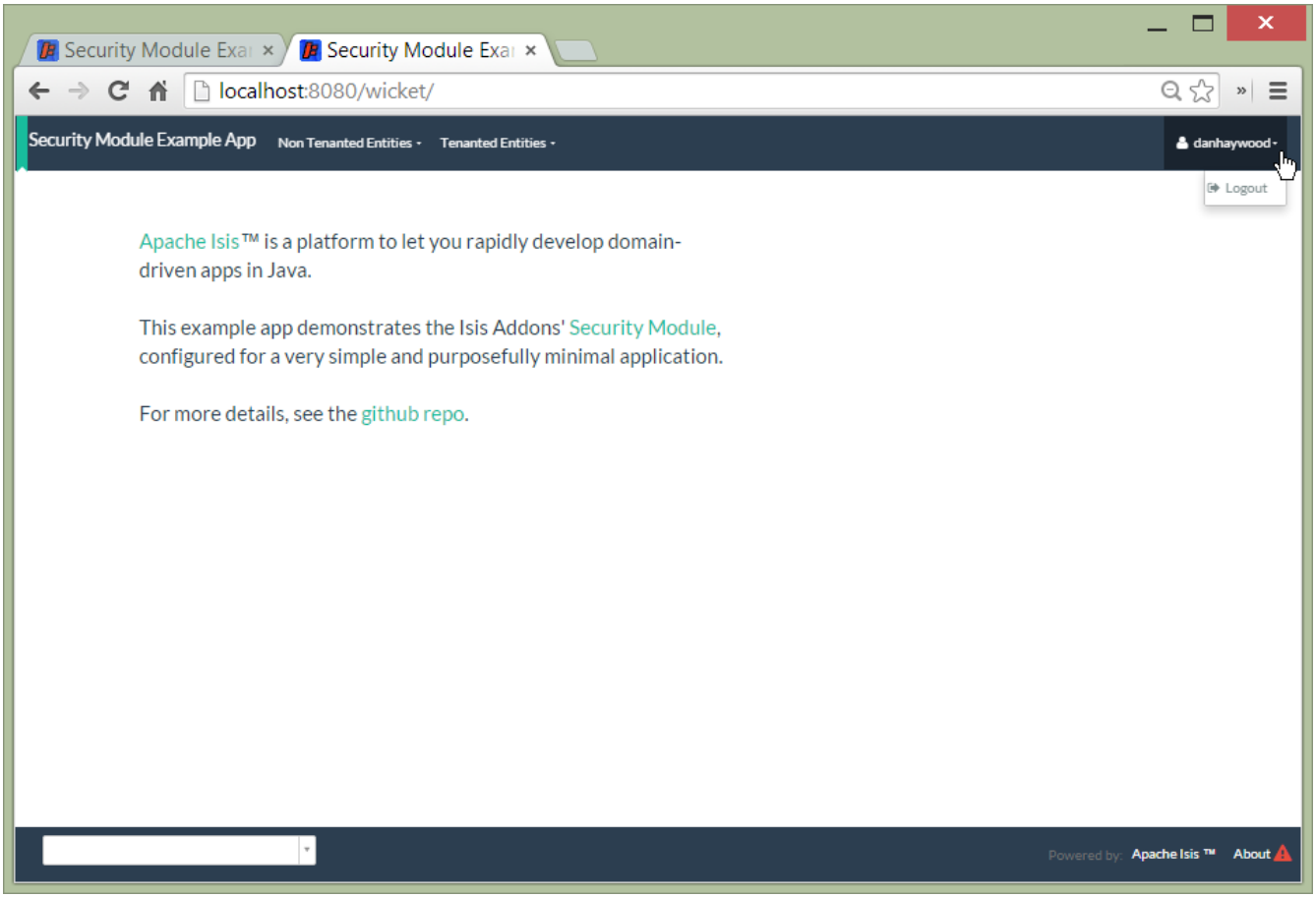
Email arrives, with link:



Follow the link, complete the page:



Automatically logged in:



2.6.2. Configuration

There are two prerequisites:

- register an implementation of the user registration service (eg by using the Isis addons security module)
- configure the email service

The latter is required if you are using the default email notification service and email service. If you are using your own alternative implementation of the email notification service then it may be omitted (and configure your own alternative implementation as required).

It is also possible to configure the Wicket viewer to suppress the sign-up page link and/or the password reset page.

Chapter 3. Configuration Properties

Wicket configuration properties alter the way in which Apache Isis' Wicket viewer renders domain objects. They are typically stored in `WEB-INF/viewer_wicket.properties`.

To tell Apache Isis that the Wicket viewer is in use (and should therefore search for the `viewer_wicket.properties` file), add the following to `WEB-INF/web.xml`:

```
<context-param>
    <param-name>isis.viewers</param-name>
    <param-value>wicket</param-value>
</context-param>
```

If you prefer you can place all configuration properties into `WEB-INF/isis.properties` (the configuration properties from all config files are merged together).

Table 1. Wicket Viewer Configuration Properties

Property	**Value** (*default value*)	**Description**
`isis.viewer.wicket.ajaxDebugMode`	`true`,`false` (`false`)	whether the Wicket debug mode should be enabled.
`isis.viewer.wicket.bookmarkedPages`	+ve int (`15`)	number of pages to bookmark
`isis.viewer.wicket.breadcrumbs.showChooser`	`true`,`false` (`true`)	Whether to show chooser for Breadcrumbs
`isis.viewer.wicket.datePattern`	date format (`dd-MM-yyyy`)	The `SimpleDateFormat` used to render dates. For the date picker (which uses `moment.js` library), this is converted dynamically into the corresponding `moment.js` format.
`isis.viewer.wicket.dateTimePattern`	date/time format (`dd-MM-yyyy HH:mm`)	The `SimpleDateFormat` used to render date/times. For the date picker (which uses `moment.js` library), this is converted dynamically into the corresponding `moment.js` format.
`isis.viewer.wicket.datePicker.maxDate`	ISO format date (`2100-01-01T00:00:00.000Z`)	Specifies a maximum date after which dates may not be specified. See datetimepicker reference docs for further details. The string must be in ISO date format (see here for further details).
`isis.viewer.wicket.datePicker.minDate`	ISO format date (`1900-01-01T00:00:00.000Z`)	Specifies a minimum date before which dates may not be specified. See datetimepicker reference docs for further details. The string must be in ISO date format (see here for further details).

Property	Value (*default value*)	Description
isis.viewer.wicket.disableDependentChoiceAutoSelection	true,false (false)	For dependent choices, whether to automatically select the first dependent (eg subcategory) when the parameter on which it depends (category) changes.
isis.viewer.wicket.disableModalDialogs	true,false (false)	No longer supported.
isis.viewer.wicket.liveReloadUrl	URL	Specifies the URL if live reload is set up, eg: http://localhost:35729/livereload.js?snipver=1
isis.viewer.wicket.maxTitleLengthInParentedTables	+ve integer, (12)	See discussion below.
isis.viewer.wicket.maxTitleLengthInStandaloneTables	+ve integer, (12)	See discussion below.
isis.viewer.wicket.maxTitleLengthInTables	+ve integer, (12)	See discussion below.
isis.viewer.wicket.regularCase	true,false (false)	Ignored for 1.8.0+; in earlier versions forced regular case rather than title case in the UI
isis.viewer.wicket.rememberMe.cookieKey	ascii chars (isisWicketRememberMe)	Cookie key holding the (encrypted) 'rememberMe' user/password. There is generally no need to change this. Valid values as per this StackOverflow answer.
isis.viewer.wicket.rememberMe.encryptionKey	any string (a random UUID each time)	Encryption key is used to encrypt the rememberMe user/password. Apache Isis leverages Apache Wicket's rememberMe support which holds remembered user/passwords in an encrypted cookie. If a hard-coded and publicly known value were to be used (as was the case prior to 1.13.0), then it would be possible for rememberMe user/password to be intercepted and decrypted, possibly compromising access. This configuration property therefore allows a private key to be specified, baked into the application. If no value is set then (for safety) a random UUID will be used as the encryption key. (The net effect of this fallback behaviour is that 'rememberMe' will work, but only until the webapp is restarted (after which the end-user will have to log in again).
isis.viewer.wicket.rememberMe.suppress	true,false (false)	Whether to suppress "remember me" checkbox on the login page.

Property	Value (*default value*)	Description
isis.viewer.wicket.stripWicketTags	true,false (true)	Whether to force Wicket tags to be stripped in prototype/development mode. See discussion below.
isis.viewer.wicket.suppressPasswordReset	true,false (false)	If user registration is enabled, whether to suppress the "password reset" link on the login page. See discussion below.
isis.viewer.wicket.suppressRememberMe	true,false (false)	(Deprecated in 1.13.0, replaced by rememberMe.suppress). Whether to suppress "remember me" checkbox on the login page.
isis.viewer.wicket.suppressSignUp	true,false (false)	If user registration is enabled, whether to suppress the "sign up" link on the login page. See discussion below.
isis.viewer.wicket.timestampPattern	date/time format (yyyy-MM-dd HH:mm:ss.SSS)	The SimpleDateFormat used to render timestamps.
isis.viewer.wicket.themes.enabled	comma separated list ...	... of bootswatch themes. Only applies if themes.showChooser==true. See discussion below.
isis.viewer.wicket.themes.showChooser	true,false (false)	Whether to show chooser for Bootstrap themes. See discussion below
isis.viewer.wicket.wicketSourcePlugin	true,false (false)	Whether the Wicketsource plugin should be enabled; by default it is not enabled. Prior to 1.12.0 this was enabled by default for prototyping (not production). However it can significantly slow down rendering, hence the introduction of this configuration setting.

3.1. Abbreviating/suppressing titles in tables

Objects whose title is overly long can be cumbersome in titles. The Wicket viewer has a mechanism to automatically shorten the titles of objects specified using @Title. As an alternative/in addition, the viewer can also be configured to simply truncate titles longer than a certain length.

The properties themselves are:

```
isis.viewer.wicket.maxTitleLengthInStandaloneTables=20
isis.viewer.wicket.maxTitleLengthInParentedTables=8
```

If you wish to use the same value in both cases, you can also specify just:

```
isis.viewer.wicket.maxTitleLengthInTables=15
```

This is used as a fallback if the more specific properties are not provided.

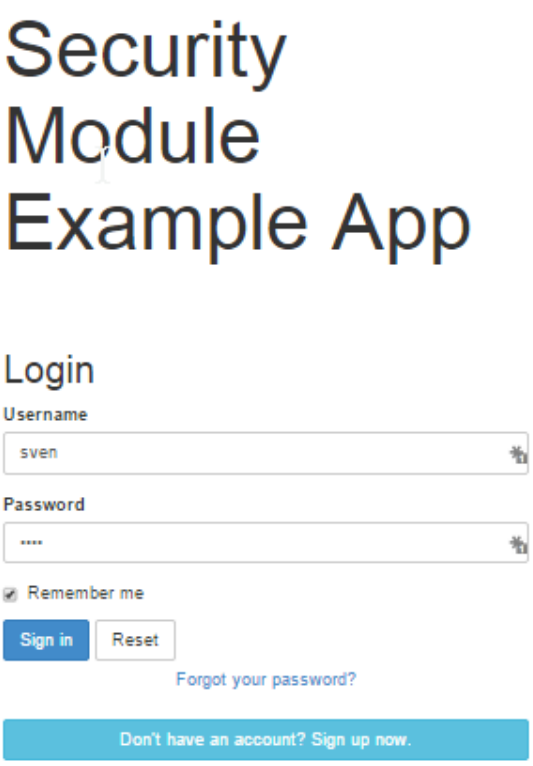
If no properties are provided, then the Wicket viewer defaults to abbreviating titles to a length of 12.

3.2. Suppressing 'remember me'

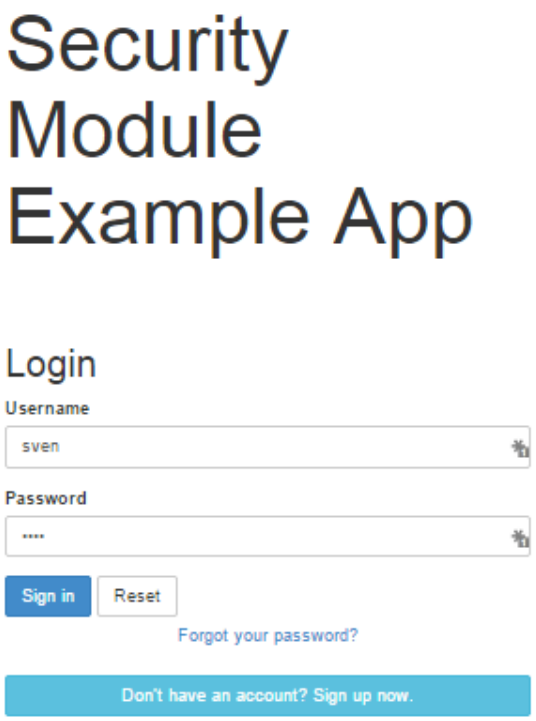
The 'remember me' checkbox on the login page can be suppressed, if required, by setting a configuration flag.

3.2.1. Screenshots

With 'remember me' not suppressed (the default):



and with the checkbox suppressed:



3.2.2. Configuration

To suppress the 'remember me' checkbox, add the following configuration flag:

```
isis.viewer.wicket.rememberMe.suppress=true
```

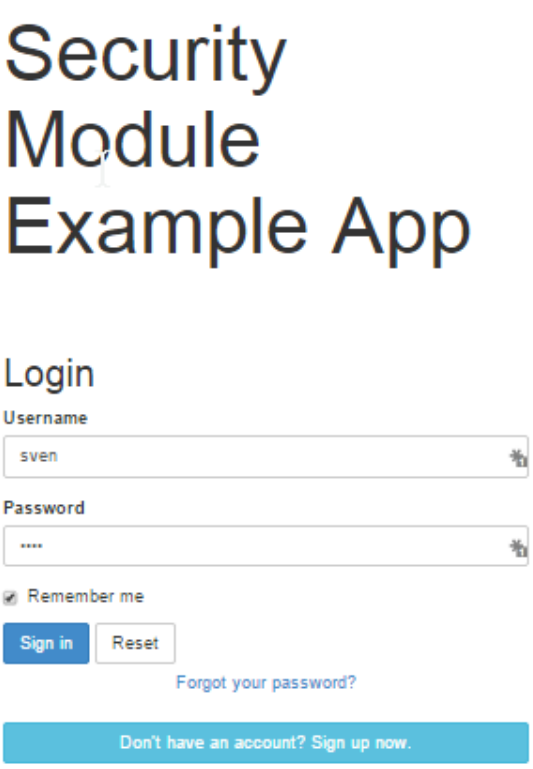
3.3. Suppressing 'sign up'

If user registration has been configured, then the Wicket viewer allows the user to sign-up a new account and to reset their password from the login page.

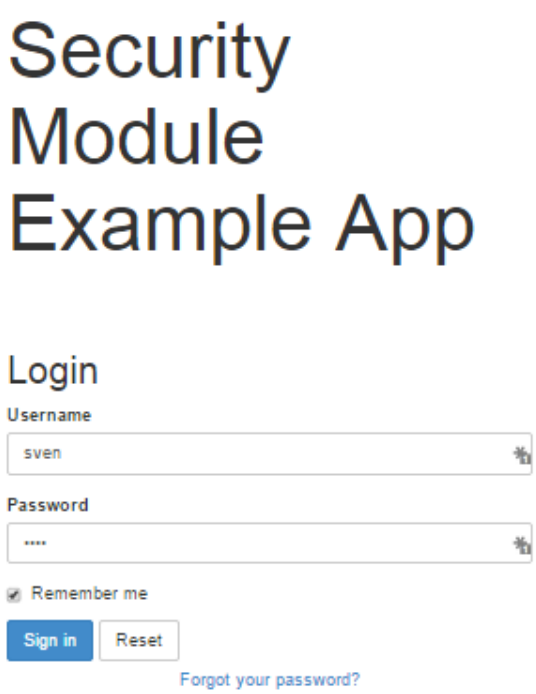
The 'sign up' link can be suppressed, if required, by setting a configuration flag.

3.3.1. Screenshots

With 'sign up' not suppressed (the default):



and with the link suppressed:



3.3.2. Configuration

To suppress the 'sign up' link, add the following configuration flag:

```
isis.viewer.wicket.suppressSignUp=true
```

3.3.3. See also

The password reset link can be suppressed in a similar manner.

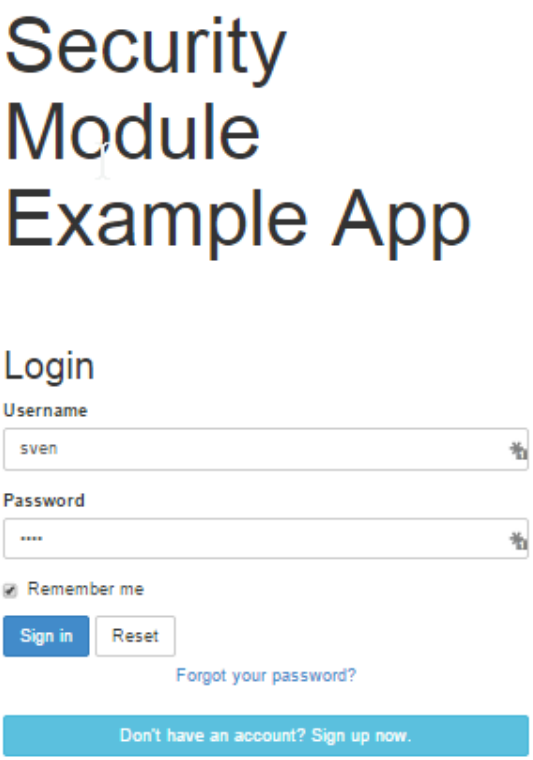
3.4. Suppressing 'password reset'

If user registration has been configured, then the Wicket viewer allows the user to sign-up a new account and to reset their password from the login page.

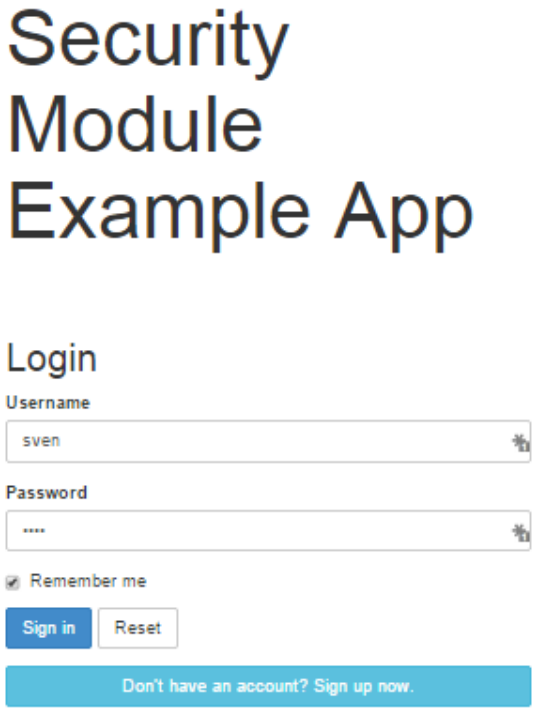
The 'password reset' link can be suppressed, if required, by setting a configuration flag.

3.4.1. Screenshots

With 'password reset' not suppressed (the default):



and with the link suppressed:



3.4.2. Configuration

To suppress the 'password reset' link, add the following configuration flag:

```
isis.viewer.wicket.suppressPasswordReset=true
```

Typically this should be added to the `viewer_wicket.properties` file (in `WEB-INF`), though you can add to `isis.properties` if you wish.

3.4.3. See also

The sign up link can be suppressed in a similar manner.

3.5. Stripped Wicket tags

By default the Apache Isis Wicket viewer will always strip wicket tags. However, when running in prototype mode, this behaviour can be overridden using a configuration property:

```
isis.viewer.wicket.stripWicketTags=false
```

In 1.7.0 and earlier, the behaviour is different; the Apache Isis Wicket viewer will preserve wicket tags when running in Apache Isis' prototype/development mode, but will still strip wicket tags in Apache Isis' server/deployment mode.

We changed the behaviour in 1.8.0 because we found that Internet Explorer can be sensitive to the presence of Wicket tags.

3.6. Showing a theme chooser

The Wicket viewer uses Bootstrap styles and components (courtesy of the Wicket Bootstrap integration).

Unless a default theme has been specified, the viewer uses the default bootstrap theme. However, the viewer can also be configured to allow the end-user to switch theme to another theme, in particular one of those provided by bootswatch.com.

This is done using the following configuration property (in `WEB-INF/viewer_wicket.properties`):

```
isis.viewer.wicket.themes.showChooser=true
```

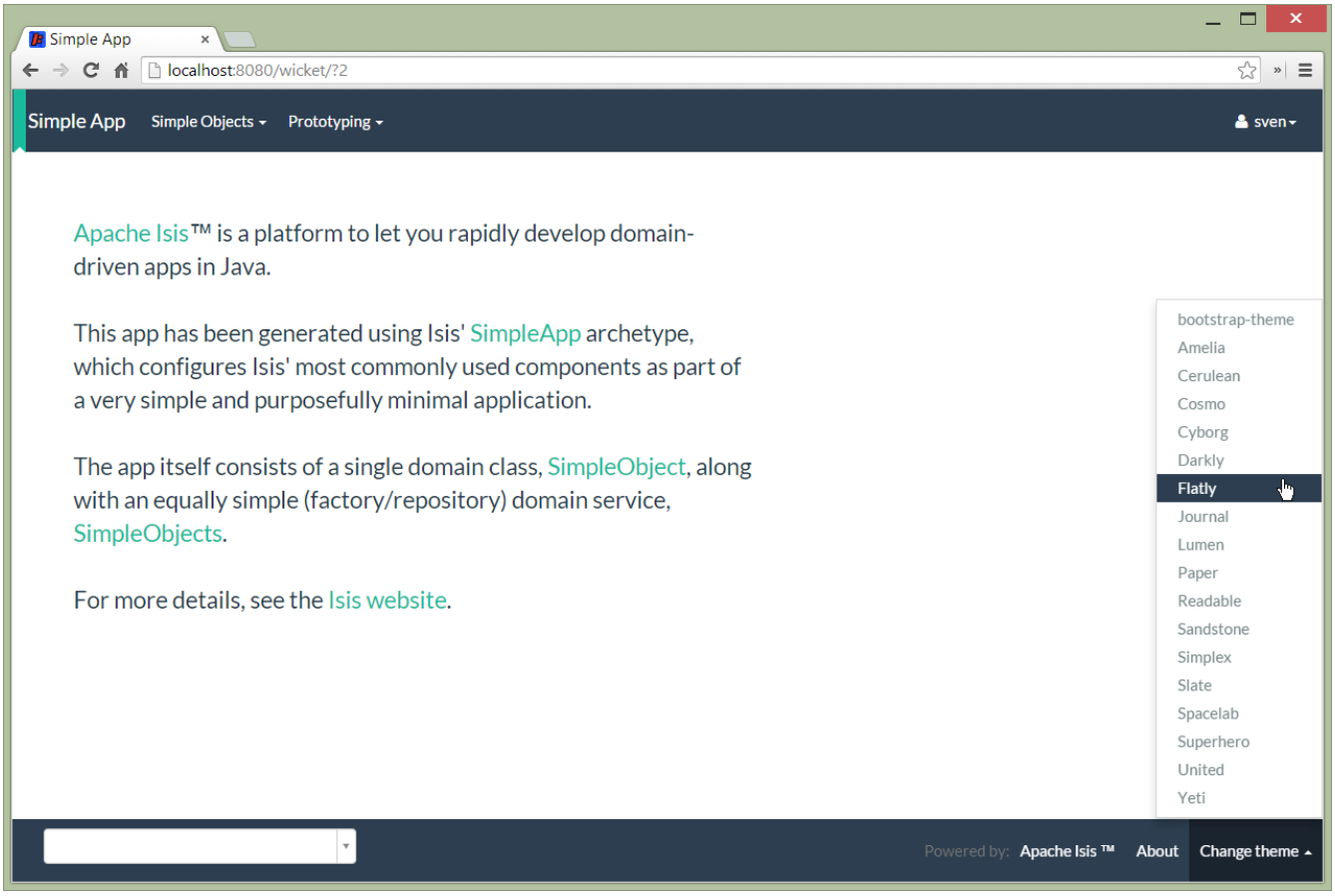


Figure 1. Example 1

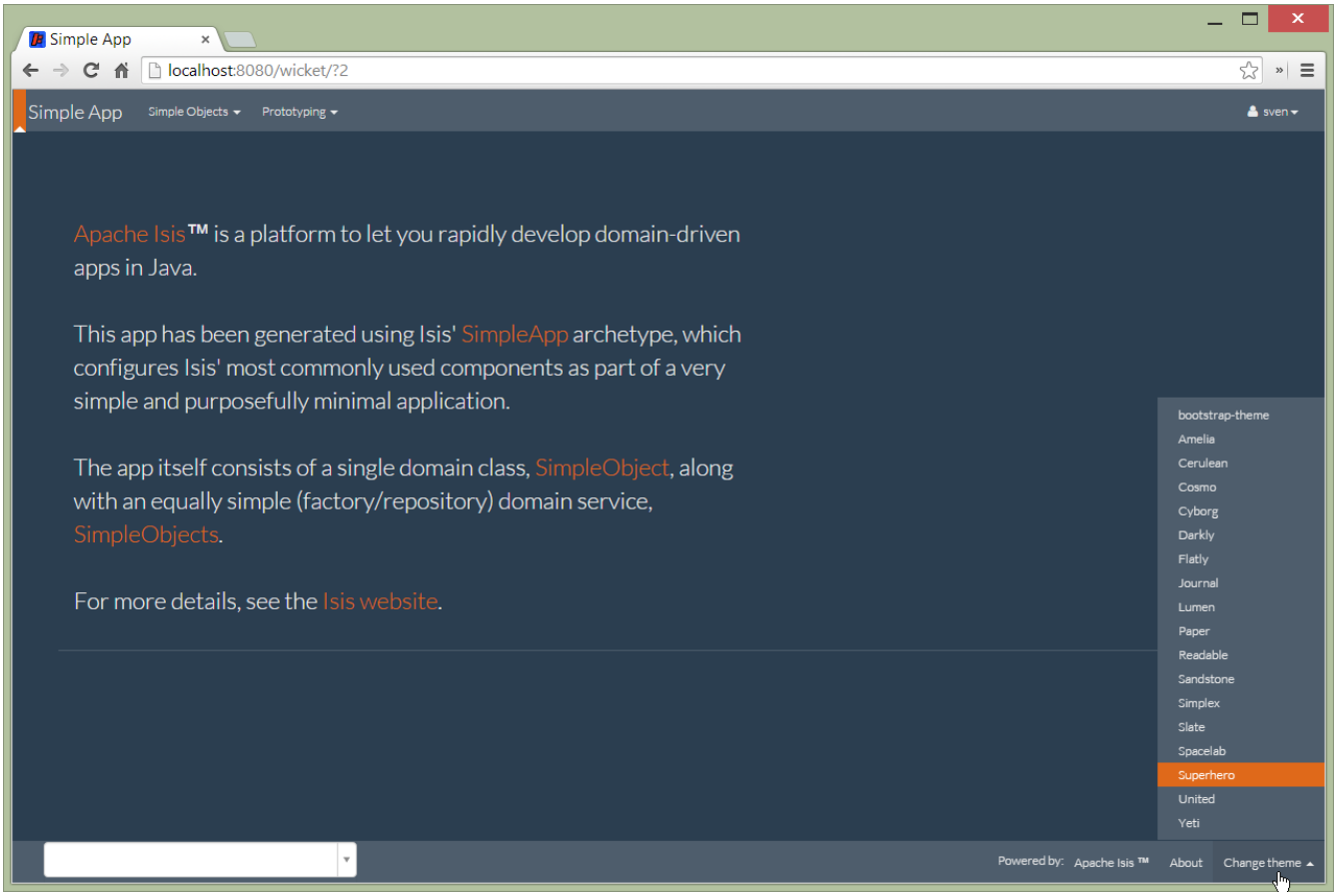


Figure 2. Example 2:

It is also possible to restrict the themes shown to some subset of those in bootswatch. This is done using a further property:

```
isis.viewer.wicket.themes.enabled=bootstrap-theme,Cosmo,Flatly,Darkly,Sandstone,United
```

where the value is the list of themes (from bootswatch.com) to be made available.

You can also develop and install a custom themes (eg to fit your company's look-n-feel/interface guidelines); see the Extending chapter for further details.

Chapter 4. Request Parameters

This section describes features that are dependent on HTTP request parameters (**not** by setting configuration properties in `isis.properties`).

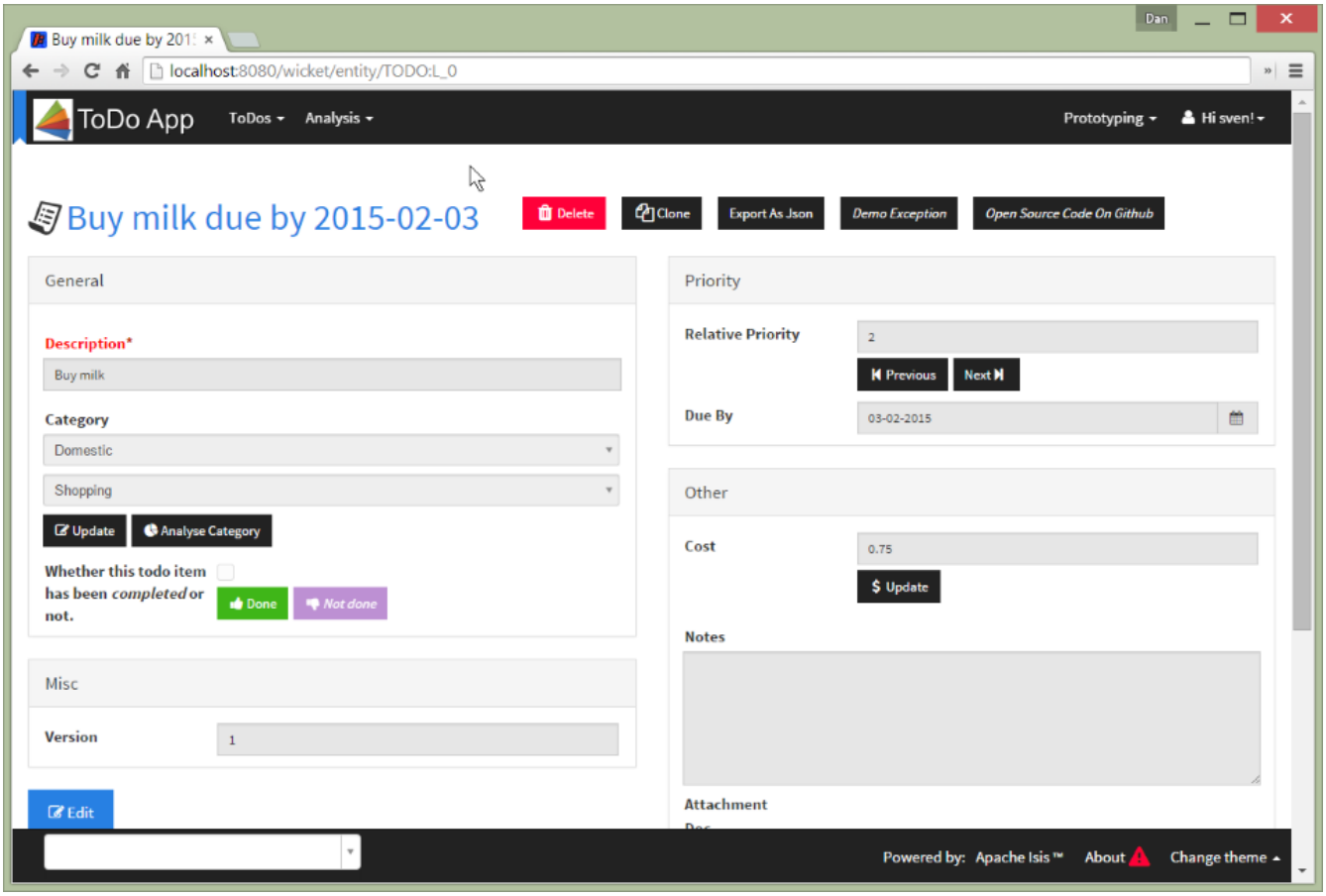
4.1. Suppressing Header and Footer (Embedded View)

The Wicket viewer provides some support such that an Isis application can be embedded within a host webapp, for example within an iframe.

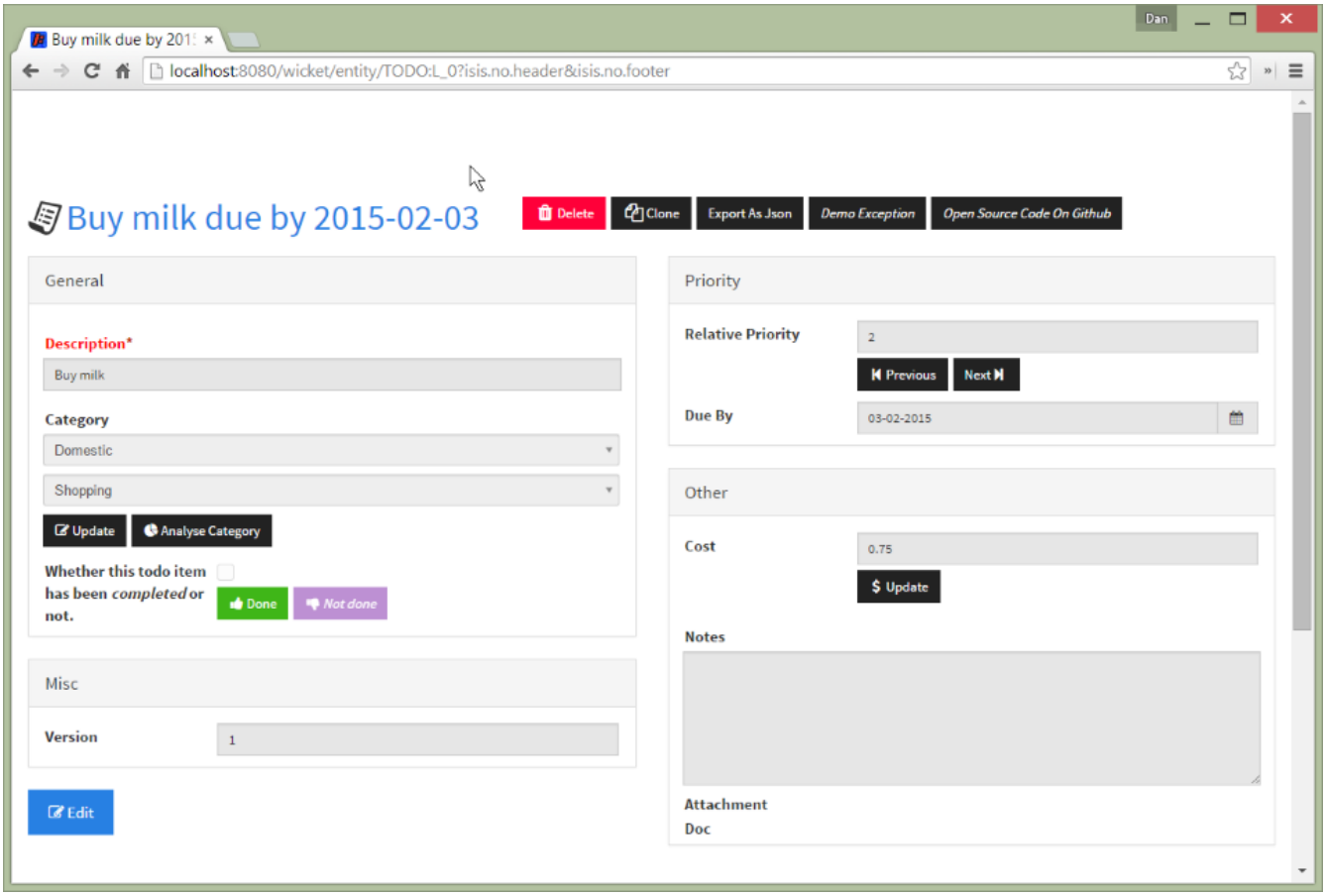
Currently this support consists simply of being able to suppress the header and/or footer.

4.1.1. Screenshots

For example, the regular view is:



With the header and footer both suppressed only the main content is shown:



It is also possible to suppress just the header, or just the footer.

4.1.2. Request parameters

To suppress the header, add the following as a request parameter:

```
isis.no.header
```

and to suppress the header, add the following as a request parameter:

```
isis.no.footer
```

For example,

```
http://localhost:8080/wicket/entity/TODO:0?isis.no.header&isis.no.footer
```

Chapter 5. Layout

The layout of domain objects can be controlled either through annotations, or through the supplementary `layout.xml` file. Of these, the `layout.xml` file is superior; it offers more flexibility, and can be reloaded at runtime, thereby reducing the feedback loop.

In addition, the layout can be fine-tuned using the `TableColumnOrderService` optional SPI service (`1.14.0-SNAPSHOT`).

5.1. `layout.xml`

For more information, see:

- the user guide fundamentals (layout chapter);
- `LayoutService` (whose functionality is exposed on the prototyping menu as an action) and lso the a mixin action
- `GridService` and its supporting services, `GridLoaderService` and `GridSystemService`
- grid layout classes, defined in the Apache Isis applib

5.2. Reordering columns (`1.14.0-SNAPSHOT`)

The optional `TableColumnOrderService` SPI service can be used to reorder columns in a table, either for a parented collection (owned by parent domain object) or a standalone collection (returned from an action invocation).

For example, suppose there is a `Customer` and an `Order`:

```
Customer "1" *--> "many" Order : orders

class Order {
    int num
    Date placedOn
    Date shippedOn
    State state
}
```

The order of these properties of `Order`, when rendered in the context of its owning `Customer`, can be controlled using this implementation of `TableColumnOrderService`:

```
@DomainService(
    nature = NatureOfService.DOMAIN,
    menuOrder = "100"                                   ①
)
public class TableColumnOrderServiceForCustomerOrders
                implements TableColumnOrderService {
    public List<String> orderParented(
            final Object parent,
            final String collectionId,
            final Class<?> collectionType,
            final List<String> propertyIds) {
        return parent instanceof Customer && "orders".equals(collectionId)
             ? Arrays.asList("num", "placedOn", "state", "shippedOn")
             : null;
    }
    public List<String> orderStandalone(
            final Class<?> collectionType,
            final List<String> propertyIds) {
        return null;
    }
}
```

① specifies the order in which the `TableColumnOrderService` implementations are called.

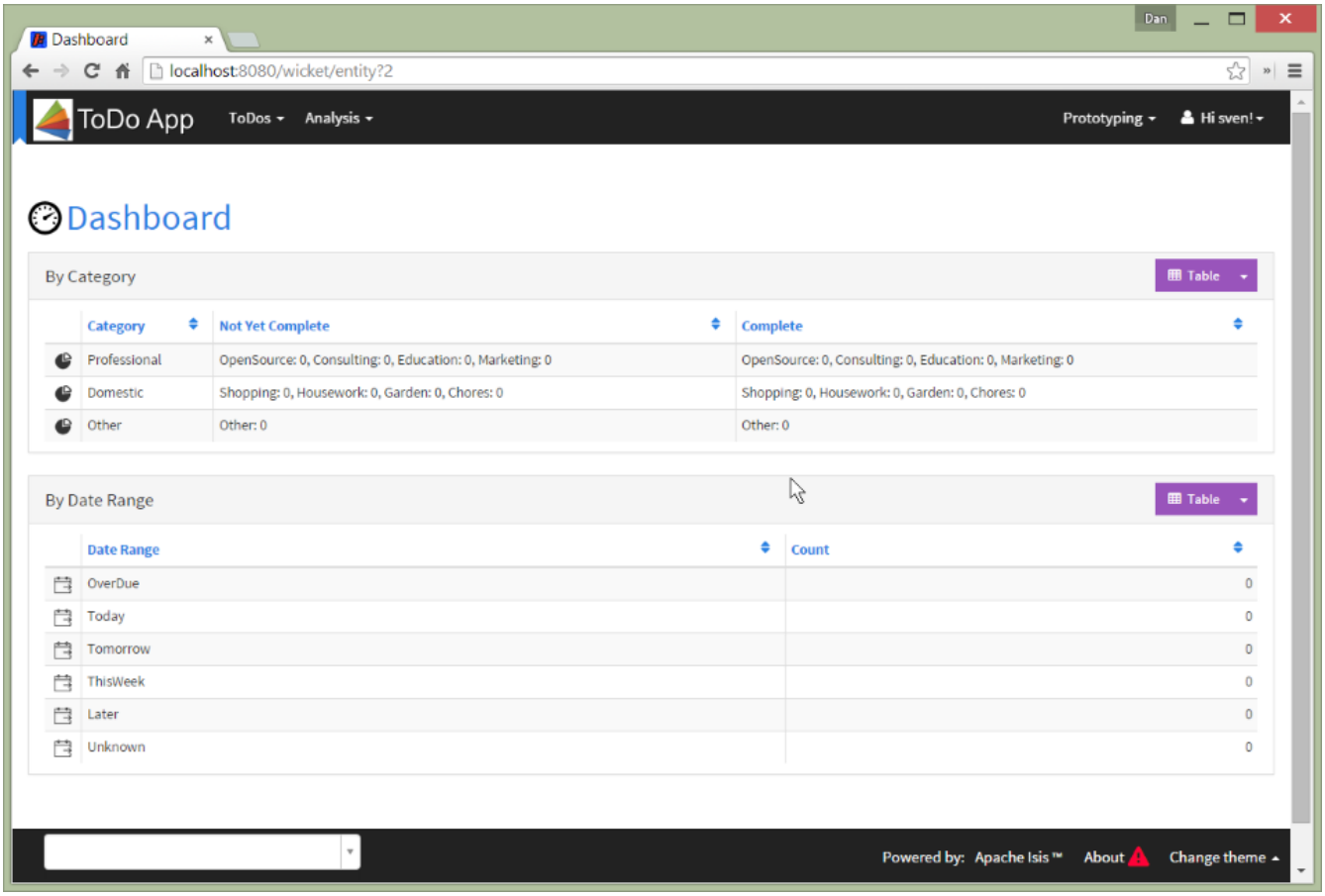
Chapter 6. Customisation

6.1. Brand logo

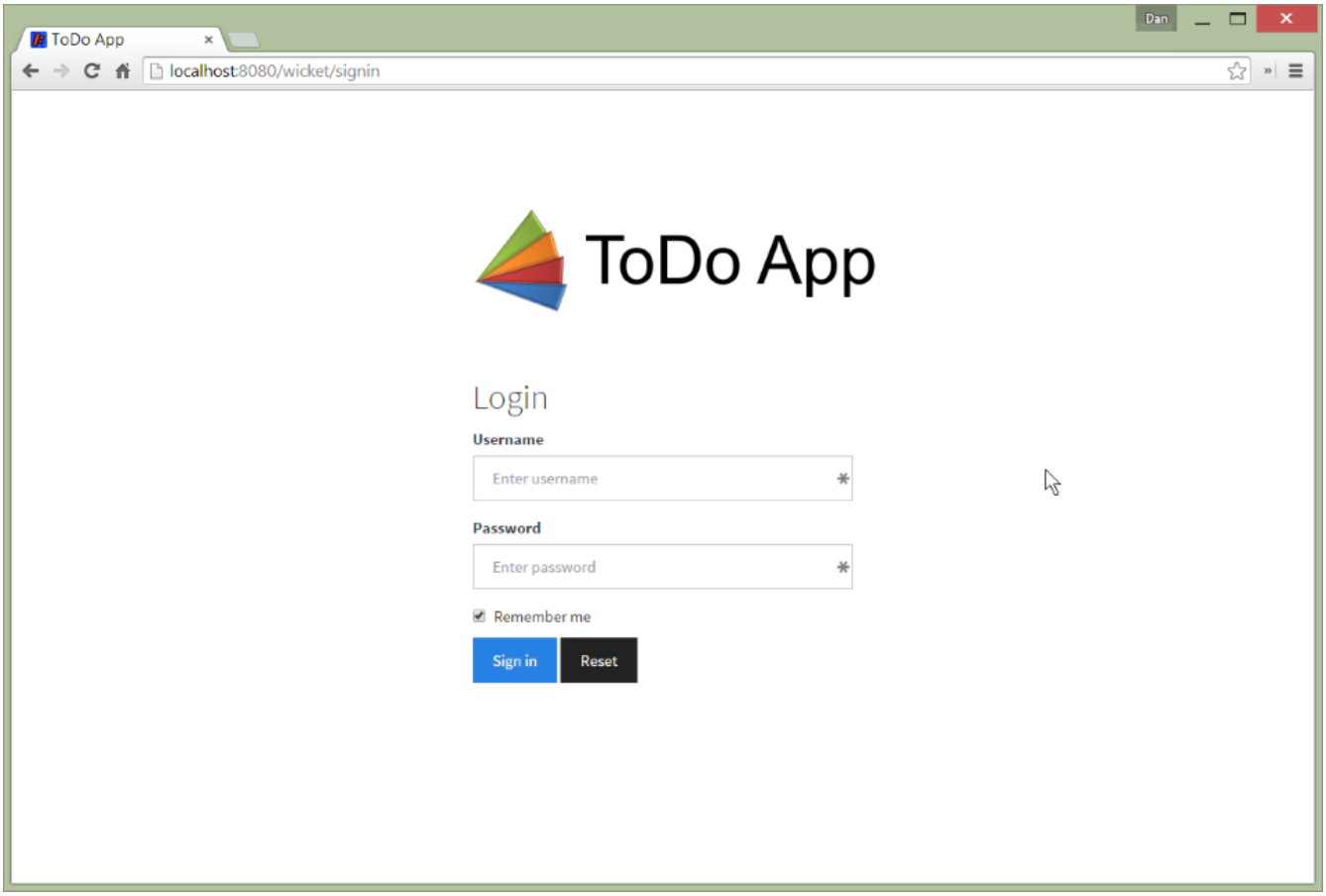
By default the Wicket viewer will display the application name top-left in the header menu. This can be changed to display a png logo instead.

6.1.1. Screenshots

The screenshot below shows the Isis addons example todoapp (not ASF) with a 'brand logo' image in its header:



A custom brand logo (typically larger) can also be specified for the signin page:



6.1.2. Configuration

In the application-specific subclass of `IsisWicketApplication`, bind:

- a string with name "brandLogoHeader" to the URL of a header image. A size of 160x40 works well.
- a string with name "brandLogoSignin" to the URL of a image for the sign-in page. A size of 400x100 works well.

For example:

```
@Override
protected Module newIsisWicketModule() {
    final Module isisDefaults = super.newIsisWicketModule();

    final Module overrides = new AbstractModule() {
        @Override
        protected void configure() {
            ...
            bind(String.class).annotatedWith(Names.named("brandLogoHeader"))
                              .toInstance("/images/todoapp-logo-header.png");
            bind(String.class).annotatedWith(Names.named("brandLogoSignin"))
                              .toInstance("/images/todoapp-logo-signin.png");
            ...
        }
    };

    return Modules.override(isisDefaults).with(overrides);
}
```

If the logo is hosted locally, add to the relevant directory (eg `src/main/webapp/images`). It is also valid for the URL to be absolute.

You may also wish to tweak the `application.css`. For example, a logo with height 40px works well with the following:

```
.navbar-brand img {
    margin-top: -5px;
    margin-left: 5px;
}
```

6.2. Specifying a default theme

The Apache Isis Wicket viewer uses Bootstrap styles and components (courtesy of the Wicket Bootstrap integration).

Unless specified otherwise, the viewer uses the default bootstrap theme. However, this can be changed by overriding `init()` in the application's subclass of `IsisWicketApplication`. For example, to set the bootswatch.com flatly theme as the default, use:

```
@Override
protected void init() {
    super.init();
    IBootstrapSettings settings = Bootstrap.getSettings();
    settings.setThemeProvider(new BootswatchThemeProvider(BootswatchTheme.Flatly));
}
```

If you have developed a custom Bootstrap theme (as described here) then this can also be specified using the Wicket Bootstrap API.

6.3. Welcome page

It's possible to customize the application name, welcome message and about message can also be customized. This is done by adjusting the Guice bindings (part of Apache Isis' bootstrapping) in your custom subclass of `IsisWicketApplication`:

```
public class MyAppApplication extends IsisWicketApplication {
    @Override
    protected Module newIsisWicketModule() {
        final Module isisDefaults = super.newIsisWicketModule();
        final Module myAppOverrides = new AbstractModule() {
            @Override
            protected void configure() {
                ...
                bind(String.class)
                    .annotatedWith(Names.named("applicationName"))
                     .toInstance("My Wonderful App");
                bind(String.class)
                    .annotatedWith(Names.named("welcomeMessage"))
                    .toInstance(readLines("welcome.html"));          ①
                bind(String.class)
                    .annotatedWith(Names.named("aboutMessage"))
                    .toInstance("My Wonderful App v1.0");
                ...
            }
        };

        return Modules.override(isisDefaults).with(myAppOverrides);
    }
}
```

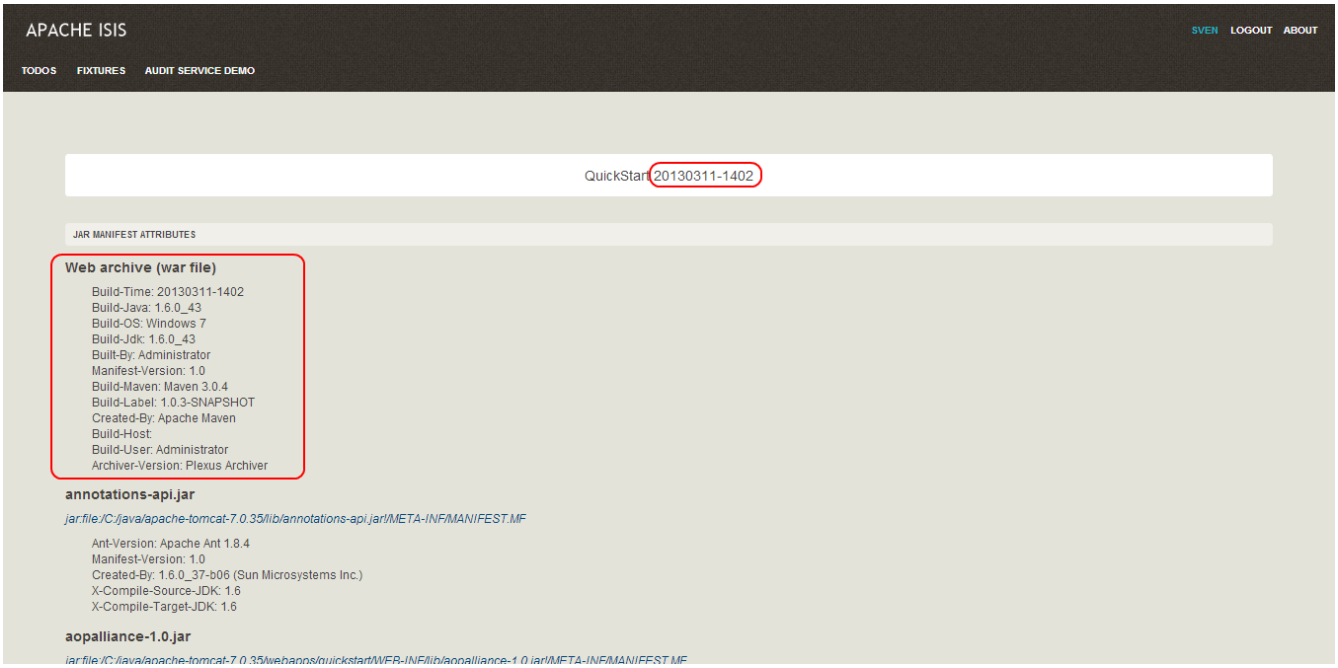
① the `welcome.html` file is resolved relative to `src/main/webapp`.

6.4. About page

Isis' Wicket viewer has an About page that, by default, will provide a dump of the JARs that make up the webapp. This page will also show the manifest attributes of the WAR archive itself, if there are any. One of these attributes may also be used as the application version number.

6.4.1. Screenshot

Here's what the About page looks like with this configuration added:



Note that this screenshot shows an earlier version of the Wicket viewer UI (specifically, pre 1.8.0).

Note that the `Build-Time` attribute has been used as the version number. The Wicket viewer is hard-coded to search for specific attributes and use as the application version. In order, it searches for:

- `Implementation-Version`
- `Build-Time`

If none of these are found, then no version is displayed.

6.4.2. Configuration

This configuration is included within the SimpleApp archetype.

Adding attributes to the WAR's manifest

Add the following to the webapp's `pom.xml` (under `<build>/<plugins>`):

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>1.5</version>
      <executions>
        <execution>
          <phase>validate</phase>
          <goals>
            <goal>maven-version</goal>
          </goals>
        </execution>
      </executions>
</plugin>

<plugin>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <addDefaultImplementationEntries>
true</addDefaultImplementationEntries>
            </manifest>
            <manifestEntries>
                <Build-Time>${maven.build.timestamp}</Build-Time>
                <Build-Number>${buildNumber}</Build-Number>
                <Build-Host>${agent.name}</Build-Host>
                <Build-User>${user.name}</Build-User>
                <Build-Maven>Maven ${maven.version}</Build-Maven>
                <Build-Java>${java.version}</Build-Java>
                <Build-OS>${os.name}</Build-OS>
                <Build-Label>${project.version}</Build-Label>
            </manifestEntries>
        </archive>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>war</goal>
            </goals>
            <configuration>
                <classifier>${env}</classifier>
            </configuration>
        </execution>
    </executions>
</plugin>
```

If you then build the webapp from the Maven command line (`mvn clean package`), then the WAR should contain a `META-INF/MANIFEST.MF` with those various attribute entries.

Exporting the attributes into the app

The manifest attributes are provided to the rest of the application by way of the Wicket viewer's integration with Google Guice.

In your subclass of `IsisWicketApplication`, there is a method `newIsisWicketModule()`. In this method you need to bind an `InputStream` that will read the manifest attributes. This is all boilerplate so you can just copy-n-paste:

```
@Override
protected Module newIsisWicketModule() {
    ...
    final Module simpleappOverrides = new AbstractModule() {
        @Override
        protected void configure() {
            ...
            bind(InputStream.class)
                .annotatedWith(Names.named("metaInfManifest"))
                .toProvider(Providers.of(
                    getServletContext().getResourceAsStream("/META-
INF/MANIFEST.MF")));
        }
    };
    ...
}
```

With that you should be good to go!

6.5. Tweaking CSS classes

The HTML generated by the Wicket viewer include plenty of CSS classes so that you can easily target the required elements as required. For example, you could use CSS to suppress the entity's icon alongside its title. This would be done using:

```
.entityIconAndTitlePanel a img {
    display: none;
}
```

These customizations should generally be added to `application.css`; this file is included by default in every webpage served up by the Wicket viewer.

6.5.1. Individual members

For example, the `ToDoItem` object of the Isis addons example todoapp (not ASF) has a `notes` property. The HTML for this will be something like:

```
<div>
  <div class="property ToDoItem-notes">
    <div class="multiLineStringPanel scalarNameAndValueComponentType">
      <label for="id83" title="">
        <span class="scalarName">Notes</span>
        <span class="scalarValue">
           <textarea

name="middleColumn:memberGroup:1:properties:4:property:scalarIfRegular:scalarValue"
              disabled="disabled"
              id="id83" rows="5" maxlength="400" size="125"
              title="">
           </textarea>
         <span>
      </label>
    </div>
  </div>
</div>
```

The `application.css` file is the place to add application-specific styles. By way of an example, if (for some reason) we wanted to completely hide the notes value, we could do so using:

```
div.ToDoItem-notes span.scalarValue {
    display: none;
}
```

You can use a similar approach for collections and actions.

6.5.2. Custom CSS styles

The above technique works well if you know the class member to target, but you might instead want to apply a custom style to a set of members. For this, you can use the `@CssClass`.

For example, in the `ToDoItem` class the following annotation (indicating that this is a key, important, property) :

```
@PropertyLayout(cssClass="x-myapp-highlight")
public LocalDate getDueBy() {
    return dueBy;
}
```

would generate the HTML:

```
<div>
    <div class="property ToDoItem-dueBy x-myapp-highlight">
        ...
    </div>
</div>
```

This can then be targeted, for example using:

```
div.x-myapp-highlight span.scalarName {
    color: red;
}
```

Note also that instead of using `@PropertyLayout(cssClass=…)` annotation, you can also specify the CSS style using a dynamic layout JSON file:

```
"dueBy": {
    "propertyLayout": {
        "cssClass": "x-myapp-important"
    }
},
```

6.5.3. Table columns

Sometimes you may want to apply styling to specific columns of tables. For example, you might want to adjust width so that for certain properties have more (or less) room than they otherwise would; or you might want to hide the column completely. This also applies to the initial icon/title column.

There is also the issue of scoping:

- You may wish the style to apply globally: that is, dependent on the type of entity being rendered in the table, irrespective of the page on which it is shown.
- Alternativel, you may wish to target the CSS for a table as rendered either as a parented collection (owned by some other entity) or rendered as a standarlone collection (the result of invoking an action).

In each of these cases the Wicket viewer adds CSS classes either to containing `div`s or to the `<th>` and `<td>` elements of the table itself so that it can custom styles can be appropriately targetted.

Applying styles globally

Every rendered collection containing a domain class will be wrapped in a `<div>` that lists that domain class (in CSS safe form). For example:

```
<div class="entityCollection com-mycompany-myapp-Customer">
    ...
    <table>
        <tr>
            <th class="title-column">...</th>
            <th class="firstName">...</th>
            <th class="lastName">...</th>
            ...
        </tr>
        <tr>
            <td class="title-column">...</td>
            <td class="firstName">...</td>
            <td class="lastName">...</td>
            ...
        </tr>
        ...
    </table>
    ...
</div>
```

Using this, the `lastName` property could be targeted using:

```
.com-mycompany-myapp-Customer th.lastName {
    width: 30%;
}
```

Parented collections

Parented collections will be wrapped in `<div>`s that identify both the entity type and also the collection Id. For example:

```
<div class="entityPage com-mycompany-myapp-Customer">       ①
    ...
    <div class="orders">                                    ②
        <table>
            <tr>
                <th class="title-column">...</th>
                <th class="productRef">...</th>
                <th class="quantity">...</th>
                ...
            </tr>
            <tr>
                <td class="title-column">...</td>
                <td class="productRef">...</td>
                <td class="quantity">...</td>
                ...
            </tr>
            ...
        </table>
        ...
    </div>
    ...
</div>
```

① the parent class identifier

② the collection identifier. This element's class also has the entity type within the collection (as discussed above).

Using this, the `productRef` property could be targeted using:

```
.com-mycompany-myapp-Customer orders td.productRef {
    font-style: italic;
}
```

Standalone collections

Standalone collections will be wrapped in a `<div>` that identifies the action invoked. For example:

```
<div class="standaloneCollectionPage">
    <div class="com-mycompany-myapp-Customer_mostRecentOrders ...">     ①
        ...
        <div class="orders">
            <table>
                <tr>
                    <th class="title-column">...</th>
                    <th class="productRef">...</th>
                    <th class="quantity">...</th>
                    ...
                </tr>
                <tr>
                    <td class="title-column">...</td>
                    <td class="productRef">...</td>
                    <td class="quantity">...</td>
                    ...
                </tr>
                ...
            </table>
            ...
        </div>
        ...
    </div>
</div>
```

① action identifier. This element's class also identifies the entity type within the collection (as discussed above).

Using this, the `quantity` property could be targeted using:

```
.com-mycompany-myapp-Customer_mostRecentOrders td.quantity {
    font-weight: bold;
}
```

6.6. Cheap-n-cheerful theme

The application name (as defined in the `IsisWicketApplication` subclass) is also used (in sanitized form) as the CSS class in a `<div>` that wraps all the rendered content of every page.

For example, if the application name is "ToDo App", then the `<div>` generated is:

```
<div class="todo-app">
    ...
</div>
```

You can therefore use this CSS class as a way of building your own "theme" for the various elements of the wicket viewer pages.

Alternatively you could "do it properly" and create your own Bootstrap theme, as described in the Extending chapter.

6.7. Using a different CSS file

If for some reason you wanted to name the CSS file differently (eg `stylesheets/myapp.css`), then adjust the Guice bindings (part of Apache Isis' bootstrapping) in your custom subclass of `IsisWicketApplication`:

```
public class MyAppApplication extends IsisWicketApplication {
    @Override
    protected Module newIsisWicketModule() {
        final Module isisDefaults = super.newIsisWicketModule();
        final Module myAppOverrides = new AbstractModule() {
            @Override
            protected void configure() {
                ...
                bind(String.class)
                    .annotatedWith(Names.named("applicationCss"))
                    .toInstance("stylesheets/myapp.css");
                ...
            }
        };

        return Modules.override(isisDefaults).with(myAppOverrides);
    }
}
```

As indicated above, this file is resolved relative to `src/main/webapp`.

6.8. Custom Javascript

The Wicket viewer ships with embedded JQuery, so this can be leveraged to perform arbitrary transformations of the rendered page (eg to run some arbitrary JQuery on page load).

Just because something is possible, it doesn't necessarily mean we encourage it. Please be aware that there is no formal API for any custom javascript that you might implement to target; future versions of Apache Isis might break your code.

If possible, consider using the `ComponentFactory` API described in the Extending chapter.

To register your Javascript code, adjusting the Guice bindings (part of Apache Isis' bootstrapping) in your custom subclass of `IsisWicketApplication`:

```
public class MyAppApplication extends IsisWicketApplication {
    @Override
    protected Module newIsisWicketModule() {
        final Module isisDefaults = super.newIsisWicketModule();
        final Module myAppOverrides = new AbstractModule() {
            @Override
            protected void configure() {
                ...
                bind(String.class)
                    .annotatedWith(Names.named("applicationJs"))
                    .toInstance("scripts/application.js");
                ...
            }
        };
        return Modules.override(isisDefaults).with(myAppOverrides);
    }
}
```

Currently only one such `.js` file can be registered.

6.9. Auto-refresh page

This requirement from the users mailing list:

Suppose you want to build a monitoring application, eg for an electricity grid. Data is updated in the background (eg via the Restful Objects REST API). What is needed is the ability to show an entity that includes a map, and have it auto-refresh every 5 seconds or so.

Here's one (somewhat crude, but workable) way to accomplish this.

- First, update the domain object to return custom CSS:

```
public class MyDomainObject {
    ...
    public String cssClass() {return "my-special-auto-updating-entity"; }
    ...
}
```

- Then, use javascript in `webapp/src/main/webapp/scripts/application.js` to reload:

```
$(function() {
    if ($(".my-special-auto-updating-entity").length) {
        setTimeout(function() {document.location.reload();}, 5000); // 1000 is 5
sec
    }
});
```

Chapter 7. Extending the Viewer

The Wicket viewer allows you to customize the GUI in several (progressively more sophisticated) ways:

- by tweaking the UI using CSS
- by tweaking the UI using Javascript
- by writing a custom bootstrap theme
- by replacing elements of the page using the `ComponentFactory` interface
- by implementing replacement page implementations for the standard page types

The first two of these options are discussed in the Wicket viewer chapter. This chapter describes the remaining "heavier-weight/more powerful" options.

The chapter wraps up with a technique for prototyping, allowing user/passwords to be specified as query arguments.

7.1. Custom Bootstrap theme

The Apache Isis Wicket viewer uses Bootstrap styles and components (courtesy of the Wicket Bootstrap integration).

By default the viewer uses the default bootstrap theme. It is possible to configure the Wicket viewer to allow the user to select other themes provided by bootswatch.com, and if required one of these can be set as the default.

However, you may instead want to write your own custom theme, for example to fit your company's look-n-feel/interface guidelines. This is done by implementing Wicket Bootstrap's `de.agilecoders.wicket.core.settings.ITheme` class. This defines:

- the name of the theme
- the resources it needs (the CSS and optional JS and/or fonts), and
- optional urls to load them from a Content Delivery Network (CDN).

To make use of the custom `ITheme` the application should register it by adding the following snippet in (your application's subclass of) `IsisWicketApplication`:

```
public void init() {
    ...
    IBootstrapSettings settings = new BootstrapSettings();
    ThemeProvider themeProvider = new SingleThemeProvider(new MyTheme());
    settings.setThemeProvider(themeProvider);
    Bootstrap.install(getClass(), settings);
}
```

7.2. Replacing page elements

Replacing elements of the page is the most powerful general-purpose way to customize the look-n-feel of the viewer. Examples at the (non-ASF) Isis Add-ons include the gmap3 extension, the calendar extension, the excel download and the wickedcharts charting integration.

The pages generated by Apache Isis' Wicket viewer are built up of numerous elements, from fine-grained widgets for property/parameter fields, to much larger components that take responsibility for rendering an entire entity entity, or a collection of entities. Under the covers these are all implementations of the the Apache Wicket `Component` API. The larger components delegate to the smaller, of course.

7.2.1. How the viewer selects components

Components are created using Apache Isis' `ComponentFactory` interface, which are registered in turn through the `ComponentFactoryRegistrar` interface. Every component is categorizes by type (the `ComponentType` enum), and Apache Isis uses this to determine which `ComponentFactory` to use. For example, the `ComponentType.BOOKMARKED_PAGES` is used to locate the `ComponentFactory` that will build the bookmarked pages panel.

Each factory is also handed a model (an implementation of `org.apache.wicket.IModel`) appropriate to its `ComponentType`; this holds the data to be rendered. For example, `ComponentType.BOOKMARKED_PAGES` is given a `BookmarkedPagesModel`, while `ComponentType.SCALAR_NAME_AND_VALUE` factories are provided a model of type of type `ScalarModel` .

In some cases there are several factories for a given `ComponentType`; this is most notably the case for `ComponentType.SCALAR_NAME_AND_VALUE`. After doing a first pass selection of candidate factories by `ComponentType`, each factory is then asked if it `appliesTo(Model)`. This is an opportunity for the factory to check the model itself to see if the data within it is of the appropriate type.

Thus, the `BooleanPanelFactory` checks that the `ScalarModel` holds a boolean, while the `JodaLocalDatePanelFactory` checks to see if it holds `org.joda.time.LocalDate`.

There will typically be only one `ComponentFactory` capable of rendering a particular `ComponentType` /`ScalarModel` combination; at any rate, the framework stops as soon as one is found.

There is one refinement to the above algorithm where multiple component factories might be used to render an object; this is discussed in Additional Views of Collections, below.

7.2.2. How to replace a component

This design (the chain of responsibility design pattern) makes it quite straightforward to change the rendering of any element of the page. For example, you might switch out Apache Isis' sliding bookmark panel and replace it with one that presents the bookmarks in some different fashion.

First, you need to write a `ComponentFactory` and corresponding `Component`. The recommended approach is to start with the source of the `Component` you want to switch out. For example:

```
public class MyBookmarkedPagesPanelFactory extends ComponentFactoryAbstract {
    public MyBookmarkedPagesPanelFactory() {
        super(ComponentType.BOOKMARKED_PAGES);
    }
    @Override
    public ApplicationAdvice appliesTo(final IModel<?> model) {
        return appliesIf(model instanceof BookmarkedPagesModel);
    }
    @Override
    public Component createComponent(final String id, final IModel<?> model) {
        final BookmarkedPagesModel bookmarkedPagesModel = (BookmarkedPagesModel)
model;
        return new MyBookmarkedPagesPanel(id, bookmarkedPagesModel);
    }
}
```

and

```
public class MyBookmarkedPagesPanel
    extends PanelAbstract<BookmarkedPagesModel> {
   ...
}
```

Here `PanelAbstract` ultimately inherits from `org.apache.wicket.Component`. Your new `Component` uses the information in the provided model (eg `BookmarkedPagesModel`) to know what to render.

Next, you will require a custom implementation of the `ComponentFactoryRegistrar` that registers your custom `ComponentFactory` as a replacement:

```
@Singleton
public class MyComponentFactoryRegistrar extends ComponentFactoryRegistrarDefault {
    @Override
    public void addComponentFactories(ComponentFactoryList componentFactories) {
        super.addComponentFactories(componentFactories);
        componentFactories.add(new MyBookmarkedPagesPanelFactory());
    }
}
```

This will result in the new component being used instead of (that is, discovered prior to) Isis' default implementation.

Previously we suggested using "replace" rather than "add"; however this has unclear semantics for some component types; see ISIS-996.

Finally (as for other customizations), you need to adjust the Guice bindings in your custom subclass of `IsisWicketApplication`:

```
public class MyAppApplication extends IsisWicketApplication {
    @Override
    protected Module newIsisWicketModule() {
        final Module isisDefaults = super.newIsisWicketModule();
        final Module myAppOverrides = new AbstractModule() {
            @Override
            protected void configure() {
                ...
                bind(ComponentFactoryRegistrar.class)
                    .to(MyComponentFactoryRegistrar.class);
                ...
            }
        };

        return Modules.override(isisDefaults).with(myAppOverrides);
    }
}
```

7.2.3. Additional Views of Collections

As explained above, in most cases Apache Isis' Wicket viewer will search for the first `ComponentFactory` that can render an element, and use it. In the case of (either standalone or parented) collections, though, Apache Isis will show all available views.

For example, out-of-the-box Apache Isis provides a table view, a summary view (totals/sums/averages of any data), and a collapsed view (for `@Render(LAZILY)` collections). These are selected by clicking on the toolbar by each collection.

Additional views though could render the objects in the collection as a variety of ways. Indeed, some third-party implementations already exist:

- excel integration (collection as a downloadable excel spreadsheet)
- google maps v3 integration (render any objects with a location on a map)
- wicked charts integration (barchart of any data)
- full calendar integration (render any objects with date properties on a calendar)

Registering these custom views is just a matter of adding the appropriate Maven module to the classpath. Apache Isis uses the JDK `ServiceLoader` API to automatically discover and register the `ComponentFactory` of each such component.

If you want to write your own alternative component and auto-register, then include a file `META-INF/services/org.apache.isis.viewer.wicket.ui.ComponentFactory` whose contents is the fully-qualified class name of the custom `ComponentFactory` that you have written.

Wicket itself has lots of components available at its wicketstuff.org companion website; you might find some of these useful for your own customizations.

7.2.4. Custom object view (eg dashboard)

One further use case in particular is worth highlighting; the rendering of an entire entity. Normally entities this is done using `EntityCombinedPanelFactory`, this being the first `ComponentFactory` for the `ComponentType.ENTITY` that is registered in Apache Isis default `ComponentFactoryRegistrarDefault`.

You could, though, register your own `ComponentFactory` for entities that is targeted at a particular class of entity - some sort of object representing a dashboard, for example. It can use the `EntityModel` provided to it to determine the class of the entity, checking if it is of the appropriate type. Your custom factory should also be registered before the `EntityCombinedPanelFactory` so that it is checked prior to the default `EntityCombinedPanelFactory`:

```
@Singleton
public class MyComponentFactoryRegistrar extends ComponentFactoryRegistrarDefault {
    @Override
    public void addComponentFactories(ComponentFactoryList componentFactories) {
        componentFactories.add(new DashboardEntityFactory());
        ...
        super.addComponentFactories(componentFactories);
        ...
    }
}
```

7.3. Custom pages

In the vast majority of cases customization should be sufficient by replacing elements of a page. However, it is also possible to define an entirely new page for a given page type.

Isis defines eight page types (see the `org.apache.isis.viewer.wicket.model.models.PageType` enum):

Table 2. PageType enum

Page type	Renders
SIGN_IN	The initial sign-in (aka login) page
SIGN_UP	The sign-up page (if user registration is enabled).
SIGN_UP_VERIFY	The sign-up verification page (if user registration is enabled; as accessed by link from verification email)
PASSWORD_RESET	The password reset page (if enabled).
HOME	The home page, displaying either the welcome message or dashboard
ABOUT	The about page, accessible from link top-right
ENTITY	Renders a single entity or view model
STANDALONE_COLLECTION	Page rendered after invoking an action that returns a collection of entites

Page type	Renders
VALUE	After invoking an action that returns a value type (though not URLs or Blob/Clobs, as these are handled appropriately automatically).
VOID_RETURN	After invoking an action that is `void`
ACTION_PROMPT	(No longer used).

The `PageClassList` interface declares which class (subclass of `org.apache.wicket.Page` is used to render for each of these types. For example, Apache Isis' `WicketSignInPage` renders the signin page.

To specify a different page class, create a custom subclass of `PageClassList`:

```
@Singleton
public class MyPageClassList extends PageClassListDefault {
    protected Class<? extends Page> getSignInPageClass() {
        return MySignInPage.class;
    }
}
```

You then need to register your custom `PageClassList`. This is done by adjusting the Guice bindings (part of Apache Isis' bootstrapping) in your custom subclass of `IsisWicketApplication`:

```
public class MyAppApplication extends IsisWicketApplication {
    @Override
    protected Module newIsisWicketModule() {
        final Module isisDefaults = super.newIsisWicketModule();
        final Module myAppOverrides = new AbstractModule() {
            @Override
            protected void configure() {
                ...
                bind(PageClassList.class).to(MyPageClassList.class);
                ...
            }
        };
        return Modules.override(isisDefaults).with(myAppOverrides);
    }
}
```

7.4. Isis Addons Extensions

TODO

Note that Isis addons, while maintained by Apache Isis committers, are not part of the ASF.

7.5. Login via Query Args (for Prototyping)

This section describes a (slightly hacky) way of allowing logins using query args, eg http://localhost:8080/?user=sven&pass=pass. This might be useful while prototyping or demonstrating a scenario involving multiple different interacting users.

Add the following code to your application's subclass of `IsisWicketApplication`:

```
private final static boolean DEMO_MODE_USING_CREDENTIALS_AS_QUERYARGS = false;

@Override
public Session newSession(final Request request, final Response response) {
    if(!DEMO_MODE_USING_CREDENTIALS_AS_QUERYARGS) {
        return super.newSession(request, response);
    }
    // else demo mode
    final AuthenticatedWebSessionForIsis s = (AuthenticatedWebSessionForIsis)
super.newSession(request, response);
    IRequestParameters requestParameters = request.getRequestParameters();
    final org.apache.wicket.util.string.StringValue user =
requestParameters.getParameterValue("user");
    final org.apache.wicket.util.string.StringValue password =
requestParameters.getParameterValue("pass");
    s.signIn(user.toString(), password.toString());
    return s;
}

@Override
public WebRequest newWebRequest(HttpServletRequest servletRequest, String filterPath)
{
    if(!DEMO_MODE_USING_CREDENTIALS_AS_QUERYARGS) {
        return super.newWebRequest(servletRequest, filterPath);
    }
    // else demo mode
    try {
        String uname = servletRequest.getParameter("user");
        if (uname != null) {
            servletRequest.getSession().invalidate();
        }
    } catch (Exception e) {
    }
    WebRequest request = super.newWebRequest(servletRequest, filterPath);
    return request;
}
```

Rather than using the static `DEMO_MODE_USING_CREDENTIALS_AS_QUERYARGS`, you might also explore using the feature toggle library provided by the (non-ASF) Isis addons' togglz module.

Chapter 8. Isis Add-ons (not ASF)

The (non-ASF) Isis Addons website provides a number of extensions to the Wicket viewer (leveraging the APIs described in Extending the Wicket viewer section, later. While you are free to fork and adapt any of them to your needs, they are also intended for use "out-of-the-box".

Note that Isis addons, while maintained by Apache Isis committers, are not part of the ASF.

This short section gives an overview of their capabilities. For full information, check out the README on their respective github repo. Each wicket extension also includes a demo app so you can (a) see what the extension does and (b) see how to use it within your own code.

8.1. Excel download

TODO

8.2. Fullcalendar2

TODO

8.3. Gmap3

TODO

8.4. Wicked charts

TODO