Hints & Tips Guide

Table of Contents

This document brings together the hints-n-tips chapters from the various user guides.

Chapter 1. Fundamentals (UI Hints)

Hints and tips for the most commonly used techniques for customising the way that the framework renders domain objects in the user interface. Taken from the Fundamentals guide.

1.1. Layout

The most significant aspect of the UI is the layout of the object's members: its properties, collections and actions. These can be organized into columns, rows and tabs.

This can be accomplished using either annotations or through a separate file-based layout. Since this is a large topic, it has its own layout chapter in the Wicket viewer guide.

1.2. Object Titles and Icons

In Apache Isis every object is identified to the user by a title (label) and an icon. This is shown in several places: as the main heading for an object; as a link text for an object referencing another object, and also in tables representing collections of objects.

The icon is often the same for all instances of a particular class, but it's also possible for an individual instance to return a custom icon. This could represent the state of that object (eg a shipped order, say, or overdue library book).

It is also possible for an object to provide a CSS class hint. In conjunction with customized CSS this can be used to apply arbitrary styling; for example each object could be rendered in a page with a different background colour.

1.2.1. Object Title

The object title is a label to identify an object to the end-user. Generally the object title is a label to identify an object to the end-user. There is no requirement for it to be absolutely unique, but it should be "unique enough" to distinguish the object from other object's likely to be rendered on the same page.

The title is always shown with an icon, so there is generally no need for the title to include information about the object's type. For example the title of a customer object shouldn't include the literal string "Customer"; it can just have the customer's name, reference or some other meaningful business identifier.

Declarative style

The `@Title` annotation can be used build up the title of an object from its constituent parts.

For example:

```
public class Customer {
    @Title(sequence="1", append=" ")
    public String getFirstName() { ... }
    @Title(sequence="2")
    public Product getLastName() { ... }
    ...
}
```

might return "Arthur Clarke", while:

```
public class CustomerAlt {
    @Title(sequence="2", prepend=", ")
    public String getFirstName() { ... }

    @Title(sequence="1")
    public Product getLastName() { ... }
    ...
}
```

could return "Clarke, Arthur".

Note that the sequence is in Dewey Decimal Format. This allows a subclass to intersperse information within the title. For example (please forgive this horrible domain modelling (!)):

```
public class Author extends Customer {
    @Title(sequence="1.5", append=". ")
    public String getMiddleInitial() { ... }
    ...
}
```

could return "Arthur C. Clarke".

Titles can sometimes get be long and therefore rather cumbersome in "parented" tables. If `@Title` has been used then the Wicket viewer will automatically exclude portions of the title belonging to the owning object.

Imperative style

Alternatively, the title can be provided simply by implementing the `title()` reserved method.

For example:

```
public class Author extends Customer {

    public String title() {
        StringBuilder buf = new StringBuilder();
        buf.append(getFirstName());
        if(getMiddleInitial() != null) {
            buf.append(getMiddleInitial()).append(". ");
        }
        buf.append(getLastName();
        return buf.toString();
    }
    ...
}
```

A variation on this approach also supports localized names; see beyond-the-basics guide for further details.

Using a UI subscriber

A third alternative is to move the responsibility for deriving the title into a separate subscriber object.

In the target object, we define an appropriate event type and use the `@DomainObjectLayout#titleUiEvent()` attribute to specify:

```
@DomainObjectLayout(
    titleUiEvent = Author.TitleUiEvent.class
)
public class Author extends Customer {
    public static class TitleUiEvent
            extends org.apache.isis.applib.services.eventbus.TitleUiEvent<Author> {}
    ...
}
```

The subscriber can then populate this event:

```
@DomainService(nature=NatureOfService.DOMAIN)
public class AuthorSubscriptions extends AbstractSubscriber {

    @org.axonframework.eventhandling.annotation.EventHandler
    @com.google.common.eventbus.Subscribe
    public void on(Author.TitleUiEvent ev) {
        Author author = ev.getSource();
        ev.setTitle(titleOf(author);
    }

    private String titleOf(Author author) {
        StringBuilder buf = new StringBuilder();
        buf.append(author.getFirstName());
        if(author.getMiddleInitial() != null) {
            buf.append(author.getMiddleInitial()).append(". ");
        }
        buf.append(author.getLastName();
        return buf.toString();
    }
}
```

1.2.2. Object Icon

The icon is often the same for all instances of a particular class, and is picked up by convention.

It's is also possible for an individual instance to return a custom icon, typically so that some significant state of that domain object is represented. For example, a custom icon could be used to represent a shipped order, say, or an overdue library loan.

Declarative style

If there is no requirement to customize the icon (the normal case), then the icon is usually picked up as the `.png` file in the same package as the class. For example, the icon for a class `org.mydomain.myapp.Customer` will be `org/mydomain/myapp/Customer.png` (if it exists).

Alternatively, font-awesome icon can be used. This is specified using the `@DomainObjectLayout#cssClassFa()` attribute.

For example:

```
@DomainObjectLayout(
    cssClassFa="play"                ①
)
public class InvoiceRun {
    ...
}
```

① will use the "fa-play" icon.

Imperative style

To customise the icon on an instance-by-instance basis, we implement the reserved `iconName()` method.

For example:

```
public class Order {
    public String iconName() {
        return isShipped() ? "shipped": null;
    }
    ...
}
```

In this case, if the `Order` has shipped then the framework will look for an icon image named "Order-shipped.png" (in the same package as the class). Otherwise it will just use "Order.png", as normal.

Using a UI subscriber

As for title, the determination of which image file to use for the icon can be externalized into a UI event subscriber.

In the target object, we define an appropriate event type and use the `@DomainObjectLayout#iconUiEvent()` attribute to specify.

For example

```
@DomainObjectLayout(
    iconUiEvent = Author.IconUiEvent.class
)
public class Order {
    public static class IconUiEvent
            extends org.apache.isis.applib.services.eventbus.IconUiEvent<Order> {}
    ...
}
```

The subscriber can then populate this event:

```
@DomainService(nature=NatureOfService.DOMAIN)
public class OrderSubscriptions extends AbstractSubscriber {

    @org.axonframework.eventhandling.annotation.EventHandler
    @com.google.common.eventbus.Subscribe
    public void on(Order.IconUiEvent ev) {
        Order order = ev.getSource();
        ev.setIconName(iconNameOf(order);
    }

    private String iconNameOf(Order order) {
        StringBuilder buf = new StringBuilder();
        return order.isShipped() ? "shipped": null;
    }
}
```

1.2.3. Object CSS Styling

It is also possible for an object to return a CSS class. In conjunction with customized CSS this can be used to apply arbitrary styling; for example each object could be rendered in a page with a different background colour.

Declarative style

To render an object with a particular CSS, use `@DomainObjectLayout#cssClass()`.

When the domain object is rendered on its own page, this CSS class will appear on a top-level `<div>`. Or, when the domain object is rendered as a row in a collection, then the CSS class will appear in a `<div>` wrapped by the `<tr>` of the row.

One possible use case would be to render the most important object types with a subtle background colour: `Customer`s shown in light green, or `Order`s shown in a light pink, for example.

Imperative style

To customise the icon on an instance-by-instance basis, we implement the reserved `cssClass()` method.

For example:

```
public class Order {
    public String cssClass() {
        return isShipped() ? "shipped": null;       ①
    }
    ...
}
```

① the implementation might well be the same as the `iconName()`.

If non-null value is returned then the CSS class will be rendered *in addition* to any declarative CSS class also specified.

Using a UI subscriber

As for title and icon, the determination of which CSS class to render can be externalized into a UI event subscriber.

In the target object, we define an appropriate event type and use the `@DomainObjectLayout#cssClassUiEvent()` attribute to specify.

For example

```
@DomainObjectLayout(
    cssClassUiEvent = Author.CssClassUiEvent.class
)
public class Order {
    public static class CssClassUiEvent
            extends org.apache.isis.applib.services.eventbus.CssClassUiEvent<Order> {}
    ...
}
```

The subscriber can then populate this event:

```
@DomainService(nature=NatureOfService.DOMAIN)
public class OrderSubscriptions extends AbstractSubscriber {

    @org.axonframework.eventhandling.annotation.EventHandler
    @com.google.common.eventbus.Subscribe
    public void on(Order.CssClassUiEvent ev) {
        Order order = ev.getSource();
        ev.setIconName(iconNameOf(order);
    }

    private String cssClassOf(Order order) {
        StringBuilder buf = new StringBuilder();
        return order.isShipped() ? "shipped": null;
    }
}
```

1.3. Action Icons and CSS

Apache Isis allows font awesome icons to be associated with each action, and for Bootstrap CSS to be applied to action rendered as buttons. These UI hints can be applied either to individual actions, or can be applied en-masse using pattern matching.

It is also possible to specify additional CSS for an object's members (not just actions).

1.3.1. Icons

Action icons can be specified in several ways.

One option is to use the `@ActionLayout#cssClassFa()`. For example:

```
@ActionLayout(cssClassFa="refresh")
public void renew() {
    ...
}
```

Alternatively, you can specify these hints dynamically in the `Xxx.layout.xml` for the entity:

```
<cpt:action id="renew" cssClassFa="refresh"/>
```

Rather than annotating every action with `@ActionLayout#cssClassFa()` and `@ActionLayout#cssClass()` you can instead specify the UI hint globally using regular expressions. Not only does this save a lot of boilerplate/editing, it helps ensure consistency across all actions.

To declare fa classes globally, use the configuration property `isis.reflector.facet.cssClassFa.patterns` (a comma separated list of key:value pairs).

For example:

```
isis.reflector.facet.cssClassFa.patterns=\
                        new.*:fa-plus,\
                        add.*:fa-plus-square,\
                        create.*:fa-plus,\
                        renew.*:fa-refresh,\
                        list.*:fa-list, \
                        all.*:fa-list, \
                        download.*:fa-download, \
                        upload.*:fa-upload, \
                        execute.*:fa-bolt, \
                        run.*:fa-bolt
```

Here:

- the key is a regex matching action names (eg `create.*`), and
- the value is a font-awesome icon name

For example, "fa-plus" is applied to all action members called "newXxx"

1.3.2. CSS

Similarly, a CSS class can be specified for object members:

- `@ActionLayout#cssClass()` for actions
- `@PropertyLayout#cssClass()` for properties, and
- `@CollectionLayout#cssClass()` for collections.

Again, this CSS class will be attached to an appropriate containing `<div>` or `<span>` on the rendered page.

Possible use cases for this is to highlight the most important properties of a domain object.

It is also possible to specify CSS classes globally, using the configuration property `isis.reflector.facet.cssClass.patterns` configuration property.

For example:

```
isis.reflector.facet.cssClass.patterns=\
                        delete.*:btn-warning
```

where (again):

- the key is a regex matching action names (eg `delete.*`), and
- the value is a Bootstrap CSS button class (eg `btn-warning) to be applied

1.4. Names and Descriptions

The name of classes and class members are usually inferred from the Java source code directly. For example, an action method called `placeOrder` will be rendered as "Place Order", and a collection called `orderItems` is rendered as "Order Items".

Occasionally though the desired name is not possible; either the name is a Java reserved word (eg "class"), or might require characters that are not valid, for example abbreviations.

In such cases the name can be specified declaratively.

It is also possible to specify a description declaratively; this is used as a tooltip in the UI.

The table below summarizes the annotations available:

Table 1. Names and descriptions

Feature	Named	Description
Class	`@DomainObjectLayout#named()`	`@DomainObjectLayout#describedAs()`
Property	`@PropertyLayout#named()`	`@PropertyLayout#describedAs()`
Collection	`@CollectionLayout#named()`	`@CollectionLayout#describedAs()`
Action	`@ActionLayout#named()`	`@ActionLayout#describedAs()`

Feature	Named	Description
Action Parameters	`@ParameterLayout#named()`	`@ParameterLayout#describedAs()`

If you're running on Java 8, then note that it's possible to write Isis applications without using `@ParameterLayout(named=…)` annotation. Support for this can be found in the (non-ASF) Incode Platform's paraname8 metamodel extension (non-ASF). (In the future we'll fold this into core). See also our guidance on upgrading to Java 8.

The framework also supports i18n: locale-specific names and descriptions. for more information, see the beyond-the-basics guide.

1.5. Eager rendering

By default, parented collections all rendered lazily, in other words in a "collapsed" table view.

For the more commonly used collections we want to show the table expanded:

For this we annotate the collection using `@CollectionLayout#defaultView()`; for example

```
@javax.jdo.annotations.Persistent(table="ToDoItemDependencies")
private Set<ToDoItem> dependencies = new TreeSet<>();
@Collection
@CollectionLayout(
    defaultView = "table"
)
public Set<ToDoItem> getDependencies() {
    return dependencies;
}
```

The `defaultView()` attribute replaces the deprecated approach of using `@CollectionLayout#render()` eagerly.

Alternatively, it can be specified the `Xxx.layout.xml` file:

```
<bs3:col span="5" cssClass="custom-padding-top-20">
    <cpt:collection id="notYetComplete" defaultView="table" />
    <cpt:collection id="complete" defaultView="table"/>
</bs3col>
```

It might be thought that collections that are eagerly rendered should also be eagerly loaded from the database by enabling the `defaultFetchGroup` attribute:

```
@javax.jdo.annotations.Persistent(table="ToDoItemDependencies", defaultFetchGroup=
"true")
private Set<ToDoItem> dependencies = new TreeSet<>();
...
```

While this can be done, it's likely to be a bad idea, because doing so will cause DataNucleus to query for more data irrespective of how the object is used/rendered.

Of course, your mileage may vary, so do experiment.

Chapter 2. Wicket Viewer

Hints and tips for more advanced customisation of the UI. Taken from the Wicket viewer guide.

2.1. Per-user Themes

From this thread on the Apache Isis users mailing list:

- *Is it possible to have each of our resellers (using our Isis application) use there own theme/branding with their own logo and colors? Would this also be possible for the login page, possibly depending on the used host name?*

Yes, you can do this, by installing a custom implementation of the Wicket Bootstrap's `ActiveThemeProvider`.

The Isis addons' todoapp (non-ASF) actually does this, storing the info via the (non-ASF) Incode Platform's settings module :

IActiveThemeProvider implementation

```
public class UserSettingsThemeProvider implements ActiveThemeProvider {
    ...
    @Override
    public ITheme getActiveTheme() {
        if(IsisContext.getSpecificationLoader().isInitialized()) {
            final String themeName = IsisContext.doInSession(new Callable<String>() {
                @Override
                public String call() throws Exception {
                    final UserSettingsService userSettingsService =
                        lookupService(UserSettingsService.class);
                    final UserSetting activeTheme = userSettingsService.find(
                            IsisContext.getAuthenticationSession().getUserName(),
                            ACTIVE_THEME);
                    return activeTheme != null ? activeTheme.valueAsString() : null;
                }
            });
            return themeFor(themeName);
        }
        return new SessionThemeProvider().getActiveTheme();
    }
    @Override
    public void setActiveTheme(final String themeName) {
        IsisContext.doInSession(new Runnable() {
            @Override
            public void run() {
                final String currentUsrName =
                    IsisContext.getAuthenticationSession().getUserName();
                final UserSettingsServiceRW userSettingsService =
                        lookupService(UserSettingsServiceRW.class);
                final UserSettingJdo activeTheme =
```

```
                        (UserSettingJdo) userSettingsService.find(
                                                currentUsrName, ACTIVE_THEME);
                if(activeTheme != null) {
                    activeTheme.updateAsString(themeName);
                } else {
                    userSettingsService.newString(
                        currentUsrName, ACTIVE_THEME, "Active Bootstrap theme for
user", themeName);
                }
            }
        });
    }
    private ITheme themeFor(final String themeName) {
        final ThemeProvider themeProvider = settings.getThemeProvider();
        if(themeName != null) {
            for (final ITheme theme : themeProvider.available()) {
                if (themeName.equals(theme.name()))
                    return theme;
            }
        }
        return themeProvider.defaultTheme();
    }
    ...
}
```

and

Using the ActiveThemeProvider

```
@Override
protected void init() {
    super.init();

    final IBootstrapSettings settings = Bootstrap.getSettings();
    settings.setThemeProvider(new BootswatchThemeProvider(BootswatchTheme.Flatly));

    settings.setActiveThemeProvider(new UserSettingsThemeProvider(settings));
}
```

2.2. How i18n the Wicket viewer?

From this thread on the Apache Isis users mailing list:

- *I am trying to internationalize the label descriptions of form actions, eg those in* `ActionParametersFormPanel`. *Referencing those via their message id inside a .po file didn't work either. Can this be done?*

The above FAQ was raised against `1.10.0`. As of `1.11.0` (due to ISIS-1093) it *is* now possible to internationalize both the Wicket viewer's labels as well as the regular translations of the domain

object metadata using the `.po` translation files as supported by the `TranslationService`.

Full details of the `msgId`s that must be added to the `translations.po` file can be found in i18n section of the beyond the basics guide.

In prior releases (`1.10.0` and earlier) it was necessary to use Wicket's internationalization support, namely resource bundles. This is still supported (as a fallback):

- create a directory structure inside the webapp resource folder following that pattern `org.apache.isis.viewer.wicket.ui.components.actions`
- Inside there create an equivalent `ActionParametersFormPanel_xx_XX.properties` or `ActionParametersFormPanel_xx.properties` file for the various locales that you want to support (eg `ActionParametersFormPanel_en_UK.properties`, `ActionParametersFormPanel_en_US.properties`, `ActionParametersFormPanel_de.properties` and so on).

2.3. SVG Support

(As per ISIS-1604), SVG images can be used:

- as Logo in the upper left corner (Wicket Menubar)
- on the Login Page (`login.html`)
- as favicon (`image/svg+xml`, cf. ISIS-1115)

However, SVGs are not, by default, displayed on the welcome page. SVGs can be attached as `Blob`s, but they are displayed as bitmaps (by means of the Batik rasterizer) and do not scale. The rasterizer (of course) can not deal with animations (cf. attachment).

To fix this, you can add the following dependencies:

```
<dependency>
    <groupId>com.twelvemonkeys.imageio</groupId>
    <artifactId>imageio-batik</artifactId> <!-- svg -->
    <version>3.3.2</version>
</dependency>
<dependency>
    <groupId>com.twelvemonkeys.imageio</groupId>
    <artifactId>imageio-batik</artifactId> <!-- svg -->
    <version>3.3.2</version>
    <type>test-jar</type>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-transcoder</artifactId>
    <version>1.8</version>
</dependency>
```

However, **please note** that these dependencies have high CVE values, and so may constitute a

security risk.

Further discussion on this mailing list thread.

Chapter 3. Restful Objects Viewer

Hints and tips for working with the REST API, for example when developing a custom application. Taken from the Restful Objects viewer guide.

3.1. Using Chrome Dev Tools

This screencast shows how to explore the Restful API using Chrome plugins/extensions, and how we use them to write end-2-end (TCK) tests for the Restful Objects viewer.

3.2. Angular Tips

The hypermedia API exposed by Apache Isis' Restful Objects viewer is intended be support both bespoke custom-written viewers as well as generic viewers. Indeed, we expect most clients consuming the API will be bespoke, not generic.

This page captures one or two tips on using Angular to write such a bespoke client.

3.2.1. Invoking a GET link (eg invoking a query action)

Suppose you have a `CustomerService` providing a `findCustomer` action:

```
public class CustomerService {
    public String id() { return "customers"; }
    @Action(semantics=SemanticsOf.SAFE)
    public Customer findCustomer(
            @ParameterLayout(named="customerName")
            final String customerName) {
        ...
    }
}
```

Restful Objects will expose this as action with the following link that looks something like:

```
{
  "rel" : "urn:org.restfulobjects:rels/invoke",
  "href" :
"http://localhost:8080/restful/services/customers/actions/findCustomer/invoke",
  "method" : "GET",
  "type" : "application/json;profile=\"urn:org.restfulobjects:repr-types/action-
result\"",
  "arguments" : {
    "customerName" : {
      "value" : null
    }
  }
}
```

You can then invoke this using Angular' `$resource` service as follows.

```
var findCustomer = $resource(
"http://localhost:8080/restful/services/customers/actions/findCustomer/invoke?:queryString");
var findCustomerArgs = {
  "customerName": {
      "value": "Fred"
    }
};
findCustomer.get({queryString: JSON.stringify(findCustomerArgs)}, function(data) { ...
} )
```

Here the `:queryString` placeholder in the initial `$resource` constructor is expanded with a stringified version of the JSON object representing the args. Note how the `findCustomerArgs` is the same as the `"arguments"` attribute in the original link (with a value provided instead of `null`).

3.2.2. Invoking a PUT or POST link

If the method is a PUT or a POST, then no `:queryString` placeholder is required in the URL, and the args are instead part of the body.

Use `$resource.put(…)` or `$resource.post(…)` instead.

3.3. Pretty printing

The JSON representations generated by the Restful Objects viewer are in compact form if the deployment type is SERVER (ie production), but will automatically be "pretty printed" (in other words indented) if the deployment type is PROTOTYPE.

3.4. How parse images in RO viewer?

From this thread on the Apache Isis users mailing list:

- *I am trying to display an image in a JavaScript client app, the image comes from an Isis RO web service as a string, but it won't show. Is there something I should do to change the message?*

The RO viewer returns the image as a string, in the form:

```
"Tacos.jpg:image/jpeg:/9j//4AAQSkZJRgABAQEAlgCWAAD/  ...."
```

This is in the form:

```
(filename):(mime type):(binary data in base64)
```

This is basically the `Blob` value type, in string form.

To use, split the parts then format the mime type and base64 data correctly before using as source in an `<img>` tag.

3.5. View Model as Parameter

As discussed on the mailing list.

3.5.1. Query

I must provide a REST service accepting more complex view model as input parameter.

My view model parameter would look like

```
@DomainObject(
    nature = Nature.VIEW_MODEL,
    objectType = "OfferTemplateFilter"
)
@XmlRootElement(name = "OfferTemplateFilter")
@XmlAccessorType(XmlAccessType.FIELD)
@Getter @Setter
public class OfferTemplateFilter {
    public List<String> selectedDeviceManufacturer = new ArrayList<>();
    public List<String> selectedDeviceSizes = new ArrayList<>();
}
```

My REST domain service would be someting like

```
@DomainService(
    nature = NatureOfService.VIEW_REST_ONLY,
    objectType = "OfferRestService"
)
public class OfferRestService {

    @Action(semantics = SemanticsOf.IDEMPOTENT)
    public OfferTemplateSelectorForCustomer
        offerSelectorForCustomer(
            final String subscriberNumber,
            final OfferTemplateFilter filter) {
        return offerSelectorRepository.create(subscriberNumber, filter);
    }
    ...
}
```

I'm wondering how this could be achieved without custom rest service. Ideally the service consumer would post a kind of JSON structure where my view model OfferTemplateFilter would be created?

3.5.2. Possible Answer...

Rather than try to "upload" the `OfferTemplateFilter` view model as a parameter, instead treat it as a resource.

That is:

- have a new service to create an instance of the filter, and then
- update this filter (adding/removing from its two collections).
- When done, pass a reference to the filter to the original REST service, as a regular reference.

Obviously the URL passed in the last step will be rather long and messy, but that's not a problem per-se.

Chapter 4. DataNucleus Object Store

Hints and tips for working with persistent entities. Taken from the DataNucleus Object Store.

See also the official DataNucleus documentation, for much more on mappings and so forth.

4.1. Overriding JDO Annotations

The JDO Objectstore (or rather, the underlying DataNucleus implementation) builds its own persistence metamodel by reading both annotations on the class and also by searching for metadata in XML files. The metadata in the XML files takes precedence over the annotations, and so can be used to override metadata that is "hard-coded" in annotations.

In fact, JDO/DataNucleus provides two different XML files that have slightly different purposes and capabilities:

- first, a `.jdo` file can be provided which - if found - completely replaces the annotations.

  The idea here is simply to use XML as the means by which metadata is specified.

- second, an `.orm` file can be provided which - if found - provides individual overrides for a particular database vendor.

  The idea here is to accommodate for subtle differences in support for SQL between vendors. A good example is the default schema for a table: `dbo` for SQL Server, `public` for HSQLDB, `sys` for Oracle, and so on.

If you want to use the first approach (the `.jdo` file), you'll find that you can download the effective XML representation of domain entities using the downloadJdoMetadata mixin action available in prototyping mode. This then needs to be renamed and placed in the appropriate location on the classpath; see the DataNucleus documentation for details.

However, using this first approach does create a maintenance effort; if the domain entity's class structure changes over time, then the XML metadata file will need to be updated.

The second approach (using an `.orm` file) is therefore often more useful than the first, because the metadata provided overrides rather than replaces the annotations (and annotations not overridden continue to be honoured).

A typical use case is to change the database schema for an entity. For example, as of `1.9.0` the various (non-ASF) Incode Platform modules use schemas for each entity. For example, the `AuditEntry` entity in the (non-ASF) Incode Platform's audit module is annotated as:

```
@javax.jdo.annotations.PersistenceCapable(
        identityType=IdentityType.DATASTORE,
        schema = "IsisAddonsAudit",
        table="AuditEntry")
public class AuditEntry {
    ...
}
```

This will map the `AuditEntry` class to a table `"IsisAddonsAudit"."AuditEntry"`; that is using a custom schema to own the object.

Suppose though that for whatever reason we didn't want to use a custom schema but would rather use the default. Also suppose we are using SQL Server as our target database.

We can override the above annotation using a `AuditEntry-sqlserver.orm` file, placed in the same package as the `AuditEntry` entity. For example:

AuditEntry-sqlserver.orm

```
<?xml version="1.0" encoding="UTF-8" ?>
<orm xmlns="http://xmlns.jcp.org/xml/ns/jdo/orm"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/jdo/orm
        http://xmlns.jcp.org/xml/ns/jdo/orm_3_0.xsd">

    <package name="org.isisaddons.module.audit.dom">
        <class name="AuditEntry"
               schema="isisaudit"
               table="AuditEntry">
        </class>
    </package>
</orm>
```

It's also necessary to tell JDO/DataNucleus about which vendor is being used (`sqlserver` in the example above). This is done using a configuration property:

isis.properties

```
isis.persistor.datanucleus.impl.datanucleus.Mapping=sqlserver
```

4.2. Subtype not fully populated

Taken from this thread on the Apache Isis users mailing list...

If it seems that Apache Isis (or rather DataNucleus) isn't fully populating domain entities (ie leaving some properties as `null`), then check that your actions are not accessing the fields directly. Use getters instead.

Properties of domain entities should always be accessed using getters. The only code that should access to fields should be the getters themselves.

Why so? Because DataNucleus will potentially lazy load some properties, but to do this it needs to know that the field is being requested. This is the purpose of the enhancement phase: the bytecode of the original getter method is actually wrapped in code that does the lazy loading checking. But hitting the field directly means that the lazy loading code does not run.

This error can be subtle: sometimes "incorrect" code that accesses the fields will seem to work. But that will be because the field has been populated already, for whatever reason.

One case where you will find the issue highlighted is for subtype tables that have been mapped using an inheritance strategy of `NEW_TABLE`, eg:

```
@javax.jdo.annotations.PersistenceCapable
@javax.jdo.annotations.Inheritance(strategy = InheritanceStrategy.NEW_TABLE)
public class SupertypeEntity {
    ...
}
```

and then:

```
@javax.jdo.annotations.PersistenceCapable
@javax.jdo.annotations.Inheritance(strategy = InheritanceStrategy.NEW_TABLE)
public class SubtypeEntity extends SupertypeEntity {
    ...
}
```

This will generate two tables in the database, with the primary key of the supertype table propagated as a foreign key (also primary key) of the subtype table (sometimes called "table per type" strategy). This means that DataNucleus might retrieve data from only the supertype table, and the lazily load the subtype fields only as required. This is preferable to doing a left outer join from the super- to the subtype tables to retrieve data that might not be needed.

On the other hand, if the `SUPERCLASS_TABLE` strategy (aka "table per hierarchy" or roll-up) or the `SUBCLASS_TABLE` strategy (roll-down) was used, then the problem is less likely to occur because DataNucleus would obtain all the data for any given instance from a single table.

Final note: references to other objects (either scalar references or in collections) in particular require that getters rather than fields to be used to obtain them: it's hopefully obvious that DataNucleus (like all ORMs) should not and will not resolve such references (otherwise, where to stop… and the whole database would be loaded into memory).

In summary, there's just one rule: **always use the getters, never the fields**.

4.3. Java8

DataNucleus 4.x supports Java 7, but can also be used with Java 8, eg for streams support against collections managed by DataNucleus.

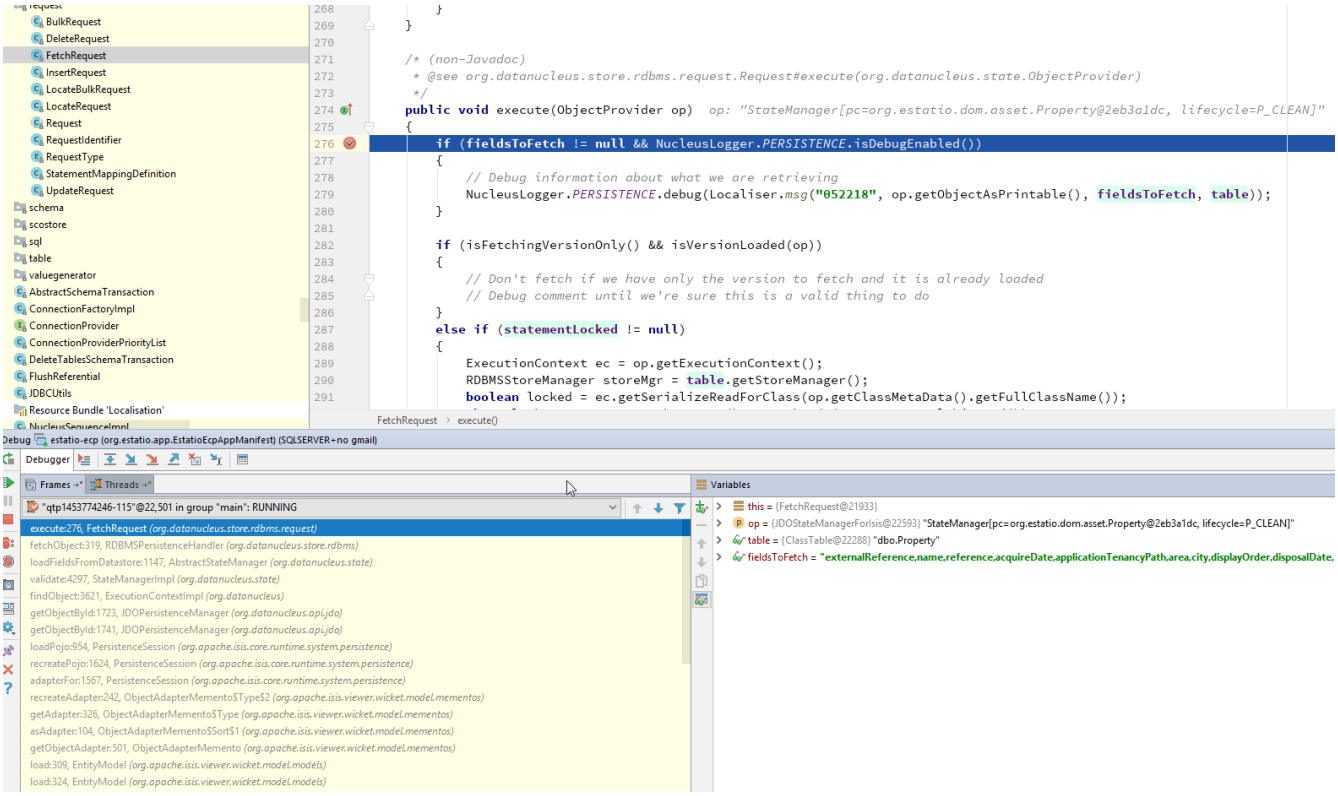
Just include within `<dependencies>` of your `dom` module's `pom.xml`:

```
<dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-java8</artifactId>
    <version>4.2.0-release</version>t
</dependency>
```

The DataNucleus website includes a page listing version compatibility of these extensions vis-a-vis the core DataNucleus platform.

4.4. Diagnosing n+1 Issues

(As of DN 4.1) set a break point in `FetchRequest#execute(…)`:



The "Variables" pane will tell you which field(s) are being loaded, and the stack trace should help explain why the field is required.

For example, it may be that an object is being loaded in a table and the initial query did not eagerly load that field. In such a case, consider using fetch groups in the initial repository query to bring the required data into memory with just one SQL call. See this hint/tip for further details.

4.5. Typesafe Queries and Fetch-groups

Fetch groups provide a means to hint to DataNucleus that it should perform a SQL join when querying. A common use case is to avoid the n+1 issue.

(So far as I could ascertain) it isn't possible to specify fetch group hints using JDOQL, but it is possible to specify them using the programmatic API or using typesafe queries.

For example, here's a JDOQL query:

```
@Query(
        name = "findCompletedOrLaterWithItemsByReportedDate", language = "JDOQL",
        value = "SELECT "
                + "FROM org.estatio.capex.dom.invoice.IncomingInvoice "
                + "WHERE items.contains(ii) "
                + "   && (ii.reportedDate == :reportedDate) "
                + "   && (approvalState != 'NEW' && approvalState != 'DISCARDED') "
                + "VARIABLES org.estatio.capex.dom.invoice.IncomingInvoiceItem ii "
),
public class IncomingInvoice ... { ... }
```

which normally would be used from a repository:

```
public List<IncomingInvoice> findCompletedOrLaterWithItemsByReportedDate(
        final LocalDate reportedDate) {
    return repositoryService.allMatches(
            new QueryDefault<>(
                    IncomingInvoice.class,
                    "findCompletedOrLaterWithItemsByReportedDate",
                    "reportedDate", reportedDate));
}
```

This can be re-written as a type-safe query as follows:

```
public List<IncomingInvoice> findCompletedOrLaterWithItemsByReportedDate(final
LocalDate reportedDate) {

    final QIncomingInvoice ii = QIncomingInvoice.candidate();
    final QIncomingInvoiceItem iii = QIncomingInvoiceItem.variable("iii");

    final TypesafeQuery<IncomingInvoice> q =
        isisJdoSupport.newTypesafeQuery(IncomingInvoice.class);

    q.filter(
            ii.items.contains(iii)
        .and(iii.reportedDate.eq(reportedDate))
        .and(ii.approvalState.ne(IncomingInvoiceApprovalState.NEW))
        .and(ii.approvalState.ne(IncomingInvoiceApprovalState.DISCARDED)));
    final List<IncomingInvoice> incomingInvoices = Lists.newArrayList(q.executeList()
);
    q.closeAll();
    return incomingInvoices;
}
```

Now the `IncomingInvoice` has four fields that require eager loading. This can be specified by defining a named fetch group:

```
@FetchGroup(
        name="seller_buyer_property_bankAccount",
        members={
                @Persistent(name="seller"),
                @Persistent(name="buyer"),
                @Persistent(name="property"),
                @Persistent(name="bankAccount")
        })
public class IncomingInvoice ... { ... }
```

This fetch group can then be used in the query using `q.getFetchPlan().addGroup(…)`. Putting this all together, we get:

```
public List<IncomingInvoice> findCompletedOrLaterWithItemsByReportedDate(final
LocalDate reportedDate) {

    final QIncomingInvoice ii = QIncomingInvoice.candidate();
    final QIncomingInvoiceItem iii = QIncomingInvoiceItem.variable("iii");

    final TypesafeQuery<IncomingInvoice> q =
        isisJdoSupport.newTypesafeQuery(IncomingInvoice.class);

    q.getFetchPlan().addGroup("seller_buyer_property_bankAccount");     ①

    q.filter(
            ii.items.contains(iii)
        .and(iii.reportedDate.eq(reportedDate))
        .and(ii.approvalState.ne(IncomingInvoiceApprovalState.NEW))
        .and(ii.approvalState.ne(IncomingInvoiceApprovalState.DISCARDED)));
    final List<IncomingInvoice> incomingInvoices = Lists.newArrayList(q.executeList()
);
    q.closeAll();
    return incomingInvoices;
}
```

① specify the fetch group to use.

Chapter 5. Security

Hints and tips for working with security module. Taken from the Security guide.

5.1. Bypassing security

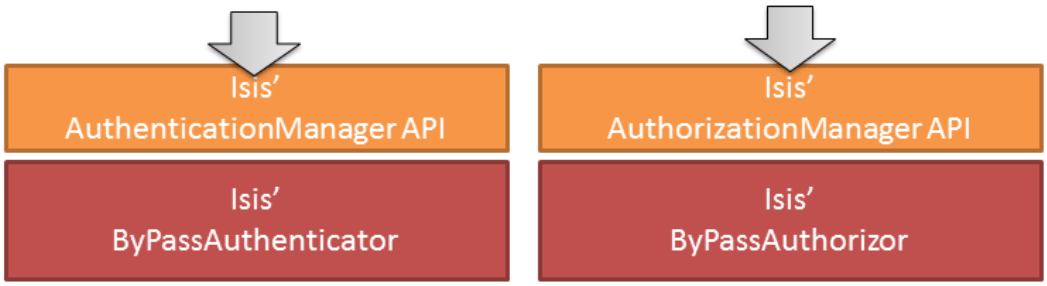
The bypass security component consists of an implementation of both the `AuthenticationManager` and `AuthorizationManager` APIs, and are intended for prototyping use only.

The authentication manager allows access with any credentials (in a sense, "bypassing" authentication), while the authorization manager provides access to all class members (in a sense, "bypassing" authorization).

To tell Apache Isis to bypass security, just update the `WEB-INF/isis.properties` file:

```
isis.authentication=bypass
isis.authorization=bypass
```

This installs the appropriate no-op implementations for both authentication and authorization:



5.2. Run-as

This hint shows how to temporarily change the current user as reported by Shiro. This can be useful to support "Run As", for example.

The heavy lifting is done in `ShiroService`:

```
@DomainService(nature = NatureOfService.DOMAIN)
public class ShiroService {

    public void runAs(String userName) {
        SimplePrincipalCollection principals =
            new SimplePrincipalCollection(userName, "jdbcRealm");
①
        getSubject().runAs(principals);
    }

    public String releaseRunAs() {
        final PrincipalCollection principals = getSubject().releaseRunAs();
        String username = (String)principals.asList().get(0);
        return username;
    }

    public String getUsername() {
②
        String principalAsString = ((String)getSubject().getPrincipal());
        return principalAsString.toLowerCase();
    }

    public String getRealUsername() {
③
        return userService.getUser().getName().toLowerCase();
    }

    public boolean isRunAs() {
        return getSubject().isRunAs();
    }

    private static Subject getSubject() {
        return org.apache.shiro.SecurityUtils.getSubject();
    }

    @Inject
    private UserService userService;
}
```

① "jdbcRealm" is realm as configured in Shiro config (shiro.ini). Might want to look this up from `ConfigurationService`.

② The username of the currently logged in user (by which permissions are determined). This could be the user name the real user is running as.

③ The username of the real currently logged in user.

This could be exposed in the UI using a simple `RunAsService`, for example:

```
@DomainService(nature = NatureOfService.VIEW_MENU_ONLY)
@DomainServiceLayout(menuBar = DomainServiceLayout.MenuBar.TERTIARY)
public class RunAsService {

    public Dashboard runAs(User user) {
        shiroService.runAs(user.getUsername());
        return dashboardService.openDashboard();                    ①
    }
    public List<User> choices0RunAs() {
        return ...                                                  ②
    }
    public boolean hideRunAs() {
        return shiroService.isRunAs();
    }



    public User releaseRunAs() {
        String username = shiroService.releaseRunAs();
        return usersRepository.findByUsername(username);
    }
    public boolean hideReleaseRunAs() {
        return !shiroService.isRunAs();
    }


    @Inject
    private ShiroService shiroService;
    @Inject
    private UsersRepository usersRepository;
    @Inject
    private DashboardService dashboardService;                      ①
}
```

① go to the home page (application-specific)

② return a list of users to run as

Credits: adapted from this gist.

5.3. Caching and other Shiro Features

We don't want to repeat the entire Shiro documentation set here, but we should flag a number of other features that are worth checking out.

5.3.1. Caching

To ensure that security operations does not impede performance, Shiro supports caching. For example, this sets up a simple memory-based cache manager:

```
memoryCacheManager = org.apache.shiro.cache.MemoryConstrainedCacheManager
securityManager.cacheManager = $memoryCacheManager
```

Other implementations can be plugged in; see the Shiro documentation for further details.

5.3.2. Further Reading

- Shiro's documentation page can be found here.
- community-contributed articles can be found here.

  These include for instance this interesting article describing how to perform certificate-based authentication (ie login using Google or Facebook credentials).

Chapter 6. Beyond the Basics

Miscellaneous hints and tips, either advanced or just obscure. Taken from the Beyond the Basics guide.

6.1. 'Are you sure?' idiom

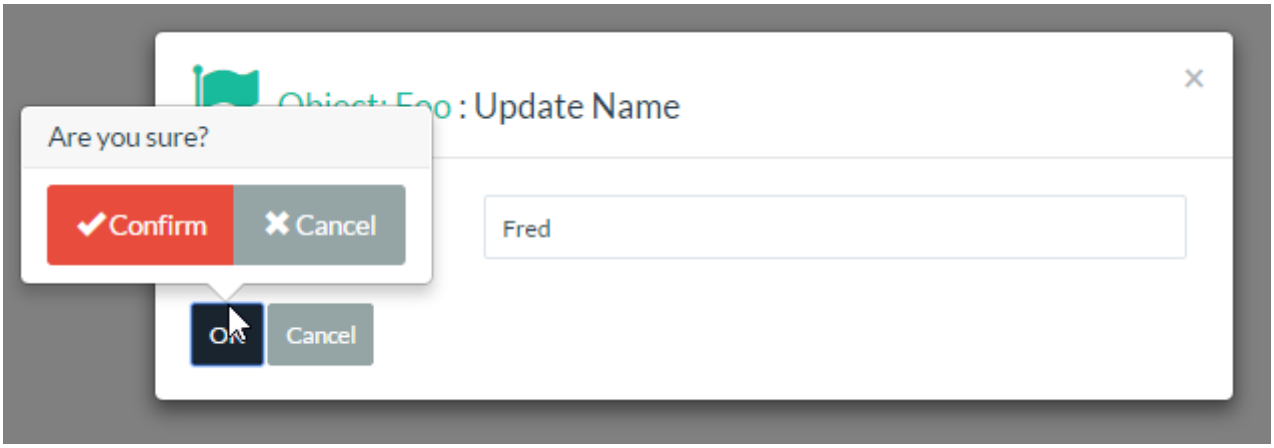
Sometimes an action might perform irreversible changes. In such a case it's probably a good idea for the UI to require that the end-user explicitly confirms that they intended to invoke the action.

6.1.1. Using action semantics

One way to meet this requirement is using the framework's built-in `@Action#semantics()` attribute:

```
@Action(
        semantics = SemanticsOf.IDEMPOTENT_ARE_YOU_SURE
)
public SimpleObject updateName(
        @Parameter(maxLength = NAME_LENGTH)
        @ParameterLayout(named = "New name")
        final String name) {
    setName(name);
    return this;
}
```

This will render as:



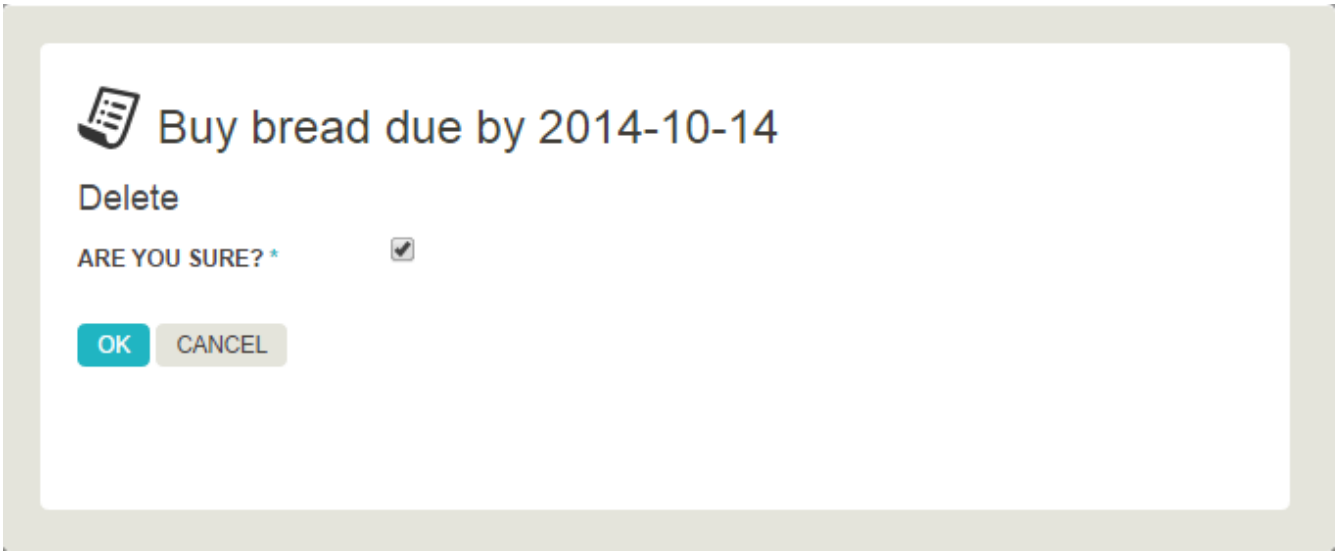
6.1.2. Using a checkbox

An alternative approach (for all versions of the framework) is to require the end-user to check a dummy checkbox parameter (and prevent the action from being invoked if the user hasn't checked that parameter).

For example:

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

If the user checks the box:



then the action will complete.

However, if the user fails to check the box, then a validation message is shown:

The code for this is pretty simple:

```
public List<ToDoItem> delete(@Named("Are you sure?") boolean areYouSure) {
    container.removeIfNotAlready(this);
    container.informUser("Deleted " + container.titleOf(this));
    return toDoItems.notYetComplete();          ①
}
public String validateDelete(boolean areYouSure) {
    return areYouSure? null: "Please confirm you are sure";
}
```

① invalid to return `this` (cannot render a deleted object)

Note that the action itself does not use the boolean parameter, it is only used by the supporting `validate…()` method.

6.2. Overriding Default Service Implns

The framework provides default implementations for many of the domain services. This is convenient, but sometimes you will want to replace the default implementation with your own service implementation.

For example, suppose you wanted to provide your own implementation of `LocaleProvider`. The trick is to use the `@DomainService#menuOrder()` attribute, specifying a low number (typically `"1"`).

Here's how:

```
@DomainService(
        nature = NatureOfService.DOMAIN,
        menuOrder = "1"                                 ①
)
public class MyLocaleProvider implements LocaleProvider {
    @Override
    public Locale getLocale() {
        return ...
    }
}
```

① takes precedence over the default implementation.

The framework uses the value of the `menuOrder` attribute to determine priority; lowest number wins.

Thus, if a single instance is to be injected, eg:

```
@javax.inject.Inject
LocalProvider localeProvider;
```

then the custom implementation will be used in preference to the framework's default implementation.

If multiple instances are injected, eg:

```
@javax.inject.Inject
List<LocalProvider> localeProviders;
```

then all implementations will be present in the list, ordered by priority; your custom implementation will be first in the list.

(As of `1.15.0` onwards), there is no need to specify the `menuOrder` attribute: its default value is now set to a lower value (specifically: `Integer.MAX_VALUE - 100`). Thus, the code simplifies to:

```
@DomainService(nature = NatureOfService.DOMAIN)
public class MyLocaleProvider implements LocaleProvider {
    @Override
    public Locale getLocale() {
        return ...
    }
}
```

It is also possible to use `@DomainServiceLayout#menuOrder()` attribute, rather than `@DomainService#menuOrder()`. The general convention is to use the former only for services that are rendered in the UI, the latter for programmatic services.

If both attributes are present, then the value of `@DomainServiceLayout#menuOrder()` is used.

6.2.1. Decorating existing implementations

It's also quite common to want to decorate the existing implementation (ie have your own implementation delegate to the default); this is also possible and quite easy:

```
@DomainService(
        nature = NatureOfService.DOMAIN
)
@DomainServiceLayout(
        menuOrder = "1"
①
)
public class MyLocaleProvider implements LocaleProvider {
    @Override
    public Locale getLocale() {
        return getDelegateLocaleProvider().getLocale();
②
    }
    Optional<LocaleProvider> delegateLocaleProvider;
③
    private LocaleProvider getDelegateLocaleProvider() {
        if(delegateLocaleProvider == null) {
            delegateLocaleProvider = Iterables.tryFind(localeProviders, input -> input
!= this);    ④
        }
        return delegateLocaleProvider.orNull();
    }
    @Inject
    List<LocaleProvider> localeProviders;
⑤
}
```

① takes precedence over the default implementation when injected elsewhere.

② this implementation merely delegates to the default implementation

③ lazily populated

④ delegate to the first implementation that isn't *this* implementation (else infinite loop!)

⑤ Injects all implementations, including this implemenation

6.3. Vetoing Visibility

FIXME - a write-up of the "vetoing subscriber" design pattern, eg as described in the `BookmarkService`

eg if included an addon such as auditing or security.

solution is to write a domain event subscriber that vetoes the visibility

All the addons actions inherit from common base classes so this can be as broad-brush or fine-grained as required

6.4. Transactions and Errors

In Apache Isis, every interaction (action invocation or property edit) is automatically wrapped in a transaction, and any repository query automatically does a flush before hand.

What that means is that there's no need to explicitly start or commit transactions in Apache Isis; this will be done for you. Indeed, if you do try to manage transactions (eg by reaching into the JDO `PersistenceManager` exposed by the IsisJdoSupport domain service, then you are likely to confuse the framework and get a stack trace for your trouble.

However, you can complete a given transaction and start a new one. This is sometimes useful if writing a fixture script which is going to perform some sort of bulk migration of data from an old system. For this use case, use the `TransactionService`.

For example:

```
public class SomeLongRunningFixtureScript extends FixtureScript

    protected void execute(final ExecutionContext executionContext) {
        // do some work
        transactionService.nextTransaction();
        // do some work
        transactionService.nextTransaction();
        // do yet more work
    }

    @javax.inject.Inject
    TransactionService transactionService;
}
```

You get the idea.

6.4.1. Raise message in the UI

The framework provides the `MessageService` as a means to return an out-of-band message to the end-user. In the Wicket viewer these are shown as "toast" pop-ups; the Restful Objects viewer

returns an HTTP header.

The `UserService` provides three APIs, for different:

- `informUser()` - an informational message. In the Wicket viewer these are short-lived pop-ups that disappear after a short time.
- `warnUser()` - a warning. In the Wicket viewer these do not auto-close; they must be acknowledged.
- `raiseError()` - an error. In the Wicket viewer these do not auto-close; they must be acknowledged.

Each pop-up has a different background colour indicating its severity.

None of these messages/errors has any influence on the transaction; any changes to objects will be committed.

6.4.2. Aborting transactions

If you want to abort Apache Isis' transaction, this can be done by throwing an exception. The exception message is displayed to the user on the error page (if Wicket viewer) or a 500 status error code (if the Restful Objects viewer).

If the exception thrown is because of an unexpected error (eg a `NullPointerException` in the domain app itself), then the error page will include a stack trace. If however you want to indicate that the exception is in some sense "expected", then throw a `RecoverableException` (or any subclass, eg `ApplicationException`); the stack trace will then be suppressed from the error page.

Another way in which exceptions might be considered "expected" could be as the result of attempting to persist an object which then violates some type of database constraint. Even if the domain application checks beforehand, it could be that another user operating on the object at the same moment of time might result in the conflict.

To handle this the `ExceptionRecognizer` SPI can be used. The framework provides a number of implementations out-of-the-box; whenever an exception is thrown it is passed to each known recognizer implementation to see if it recognizes the exception and can return a user-meaningful error message. For example, `ExceptionRecognizerForSQLIntegrityConstraintViolationUniqueOrIndexException` checks if the exception inherits from `java.sql.SQLIntegrityConstraintViolationException`, and if so, constructs a suitable message.

6.5. Persisted Title

Normally the title of an object is not persisted to the database, rather it is recomputed automatically from underlying properties. On occasion though you might want the title to also be persisted; either to make things easier for the DBA, or for an integration scenario, or some other purpose.

We can implement this feature by leveraging the JDO lifecycle. In the design we discuss here we make it a responsibility of the entities to persist the title as a property, by implementing a `ObjectWithPersistedTitle` interface:

```
public interface ObjectWithPersistedTitle {
    @PropertyLayout(hidden = Where.EVERYWHERE)  ①
    String getTitle();
    void setTitle(final String title);
}
```

① we don't want to expose this in the UI because the title is already prominently displayed.

We can then define a subscribing domain service that leverage this.

```
@DomainService(nature = NatureOfService.DOMAIN)
public class TitlingService extends AbstractSubscriber {
    @Subscribe
    public void on(final ObjectPersistingEvent ev) {
        handle(ev.getSource());
    }
    @Subscribe
    public void on(final ObjectUpdatingEvent ev) {
        handle(ev.getSource());
    }
    private void handle(final Object persistentInstance) {
        if(persistentInstance instanceof ObjectWithPersistedTitle) {
            final ObjectWithPersistedTitle objectWithPersistedTitle =
                (ObjectWithPersistedTitle) persistentInstance;
            objectWithPersistedTitle.setTitle(container.titleOf
(objectWithPersistedTitle));
        }
    }
    @Inject
    private DomainObjectContainer container;
}
```

Prior to 1.10.0 (when lifecycle events were introduced), this could also be be done by accessing the JDO API directly:

```
@RequestScoped
@DomainService(nature = NatureOfService.DOMAIN)
public class TitlingService {
    @PostConstruct
    public void init() {
        isisJdoSupport.getJdoPersistenceManager().addInstanceLifecycleListener(
            new StoreLifecycleListener() {
                @Override
                public void preStore(final InstanceLifecycleEvent event) {
                    final Object persistentInstance = event.getPersistentInstance();
                    if(persistentInstance instanceof ObjectWithPersistedTitle) {
                        final ObjectWithPersistedTitle objectWithPersistedTitle =
                            (ObjectWithPersistedTitle) persistentInstance;
                        objectWithPersistedTitle.setTitle(container.titleOf
(objectWithPersistedTitle));
                    }
                }
                @Override
                public void postStore(final InstanceLifecycleEvent event) {
                }
            }, null);
    }
    @Inject
    private IsisJdoSupport isisJdoSupport;
    @Inject
    private DomainObjectContainer container;
}
```

The above is probably the easiest and most straightforward design. One could imagine other designs where the persisted title is stored elsewhere. It could even be stored off into an Apache Lucene (or similar) database to allow for free-text searches.

6.6. View Model Instantiation

With view models, some care must be taken in how they are instantiated. Specifically, it's important that the framework doesn't "know" about the view model until its state is "sufficiently" populated to distinguish from other view models.

In practical terms, this means that view models should be instantiated using a constructor, and then injecting services (if required) using the `ServiceRegistry` service:

```
CustomerViewModel viewModel = new CustomerViewModel("Joe", "Bloggs");
serviceRegistry.injectServicesInto(viewModel);
```

What will most likely **fail** is to use the `FactoryService`:

```
// DON'T DO THIS WITH VIEW MODELS
CustomerViewModel viewModel = factoryService.instantiate(CustomerViewModel.class);

viewModel.setFirstName("Joe");
viewModel.setLastName("Bloggs");
serviceRegistry.injectServicesInto(viewModel);
```

That's because the internal "OID" identifier that the framework creates to handle this view model will be incomplete. Although the framework can handle changes to the OID (when the corresponding view model's state changes) once created, this isn't the case during initial instantiation process.

6.6.1. Example

To explain further, here's an implementation using `FactoryService` that fails:

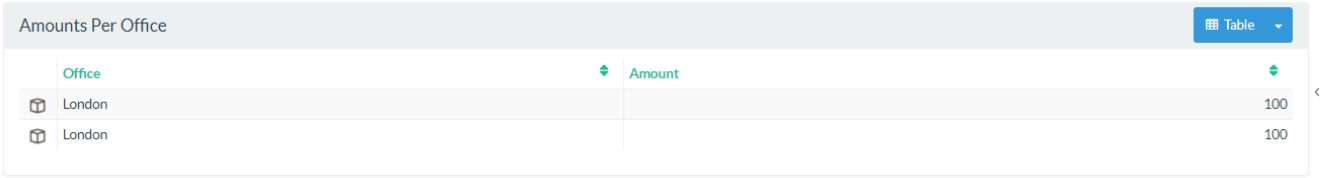
```
@XmlRootElement(name = "yearSummary")
@XmlType( propOrder = { ... } )
@XmlAccessorType(XmlAccessType.FIELD)
public class YearSummary {                                                    ①
    ...
    @XmlTransient
    @CollectionLayout(defaultView = "table")
    public List<OfficeOptionViewModel> getAmountsPerOffice() {
        List<OfficeOptionViewModel> amountsPerOffice = new ArrayList<>();

        OfficeOptionViewModel office1 =                                       ②
            factoryService.instantiate(OfficeOptionViewModel.class);
        office1.setOffice("Amsterdam");                                       ③
        office1.setAmount(200);
        amountsPerOffice.add(office1);

        OfficeOptionViewModel office2 =                                       ②
            factoryService.instantiate(OfficeOptionViewModel.class);
        office2.setOffice("London");                                          ③
        office2.setAmount(100);
        amountsPerOffice.add(office2);

        return amountsPerOffice;
    }
}
```

① Parent view model

② Using `FactoryService`, incorrectly.

③ Hard-coded just for demo purposes

This collection, is, confusing, rendered as:



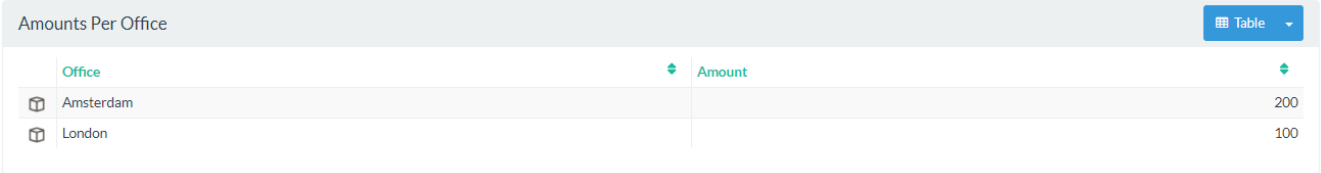
Even though the `amountsPerOffice` collection of view models is correctly populated, the framework/viewer maps these to their corresponding OIDs before they are rendered. Because the "Amsterdam" pojo and "London" pojo each mapped to the same OID, when fetching out the results the viewer obtains the London pojo both times.

The following implementation, on the other hand, succeeds:

```
@XmlRootElement(name = "yearSummary")
@XmlType( propOrder = { ... } )
@XmlAccessorType(XmlAccessType.FIELD)
public class YearSummary {
    ...
    @XmlTransient
    @CollectionLayout(defaultView = "table")
    public List<OfficeOptionViewModel> getAmountsPerOffice() {
        List<OfficeOptionViewModel> amountsPerOffice = new ArrayList<>();

        OfficeOptionViewModel office1 = new OfficeOptionViewModel("Amsterdam", 200);
①
        serviceRegistry.injectServicesInto(office1);
        amountsPerOffice.add(office1);

        OfficeOptionViewModel office2 = new OfficeOptionViewModel("London", 100);
①
        serviceRegistry.injectServicesInto(office2);
        amountsPerOffice.add(office2);

        return amountsPerOffice;
    }
}
```

① Just instantiate with constructor. The framework "sees" the domain object when services are injected into it.

As can be seen, this renders just fine:



To complicate matters a little, note though that following "incorrect" implementation using `FactoryService` does also work correctly:

```
@XmlRootElement(name = "yearSummary")
@XmlType( propOrder = { ..., "amountsPerOffice" } )                    ①
@XmlAccessorType(XmlAccessType.FIELD)
public class YearSummary {
    ...

    void init() {
        amountsPerOffice = calculateAmountsPerOffice();
    }

    @XmlElementWrapper
    @XmlElement(name = "officeOption")
    @CollectionLayout(defaultView = "table")
    @Getter @Setter
    private List<OfficeOptionViewModel> amountsPerOffice = Lists.newArrayList();

    @XmlTransient
    @CollectionLayout(defaultView = "table")
    public List<OfficeOptionViewModel> calculateAmountsPerOffice() {
        List<OfficeOptionViewModel> amountsPerOffice = new ArrayList<>();

        OfficeOptionViewModel office1 = factoryService.instantiate
(OfficeOptionViewModel.class);
        office1.setOffice("Amsterdam");
        office1.setAmount(200);

        amountsPerOffice.add(office1);

        OfficeOptionViewModel office2 = factoryService.instantiate
(OfficeOptionViewModel.class);
        office2.setOffice("London");
        office2.setAmount(100);

        amountsPerOffice.add(office2);

        return amountsPerOffice;
    }
}
```

① "amountsPerOffice" is part of the state of the parent view model

In this case the `amountsPerOffice` collection is part of the state of the parent view model and so in this particular case everything works with either `FactoryService#instantiate` or using `ServiceRegistry`.

6.7. Collections of values

Although in Apache Isis you can have properties of either values (string, number, date etc) or of (references to other) entities, with collections the framework (currently) only supports collections

of (references to) entities. That is, collections of values (a bag of numbers, say) are not supported.

However, it is possible to simulate a bag of numbers using view models.

6.7.1. View Model

FIXME

6.7.2. Persistence Concerns

FIXME - easiest to simply store using DataNucleus' support for collections, marked as `@Programmatic` so that it is ignored by Apache Isis. Alternatively can store as json/xml in a varchar(4000) or clob and manually unpack.

6.8. How to handle void/null results

From this thread on the Apache Isis users mailing list:

- *When using a `void` action, let's say a remove action, the user is redirected to a page "no results". When clicking the back button in the browser the user sees "Object not found" (since you've just deleted this object).*
- *You can return a list for example to prevent the user from being redirect to a "No results" page, but I think it's not the responsibility of the controllers in the domain model.*
- *A solution could be that wicket viewer goes back one page when encountering a deleted object. And refresh the current page when receiving a null response or invoking a void action. But how to implement this?*

One way to implement this idea is to provide a custom implementation of the `RoutingService` SPI domain service. The default implementation will either return the current object (if not null), else the home page (as defined by `@HomePage`) if one exists.

The following custom implementation refines this to use the breadcrumbs (available in the Wicket viewer) to return the first non-deleted domain object found in the list of breadcrumbs:

```
@DomainService(nature = NatureOfService.DOMAIN)
@DomainServiceLayout(menuOrder = "1")                                    ①
public class RoutingServiceUsingBreadcrumbs extends RoutingServiceDefault {
    @Override
    public Object route(final Object original) {
        if(original != null) {                                           ②
            return original;
        }
        container.flush();                                               ③

        final BreadcrumbModelProvider wicketSession =                    ④
            (BreadcrumbModelProvider) AuthenticatedWebSession.get();
        final BreadcrumbModel breadcrumbModel =
            wicketSession.getBreadcrumbModel();
        final List<EntityModel> breadcrumbs = breadcrumbModel.getList();

        final Optional<Object> firstViewModelOrNonDeletedPojoIfAny =
                breadcrumbs.stream()                                     ⑤
                .filter(entityModel -> entityModel != null)
                .map(EntityModel::getObject)                             ⑥
                .filter(objectAdapter -> objectAdapter != null)
                .map(ObjectAdapter::getObject)                           ⑦
                .filter(pojo -> !(pojo instanceof Persistable) ||
                                !((Persistable)pojo).dnIsDeleted())      ⑧
                .findFirst();

        return firstViewModelOrNonDeletedPojoIfAny.orElse(homePage());   ⑨
    }
    private Object homePage() {
        return homePageProviderService.homePage();
    }
    @Inject
    HomePageProviderService homePageProviderService;
    @Inject
    DomainObjectContainer container;
}
```

① override the default imlpementation

② if a non-null object was returned, then return this

③ ensure that any persisted objects have been deleted.

④ reach inside the Wicket viewer's internals to obtain the list of breadcrumbs.

⑤ loop over all breadcrumbs

⑥ unwrap the Wicket viewer's serializable representation of each domain object (`EntityModel`) to the Isis runtime's representation (`ObjectAdapter`)

⑦ unwrap the Isis runtime's representation of each domain object (`ObjectAdapter`) to the domain object pojo itself

⑧

if object is persistable (not a view model) then make sure it is not deleted

⑨ return the first object if any, otherwise the home page object (if any).

Note that the above implementation uses Java 8, so if you are using Java 7 then you'll need to backport accordingly.

6.9. Multi-tenancy

Of the various modules provided by the (non-ASF) Incode Platform, the security module has the most features. One significant feature is the ability to associate users and objects with a "tenancy".

For more details, see the Incode Platform's security module README.

6.10. Subclass properties in tables

Suppose you have a hierarchy of classes where a property is derived and abstract in the superclass, concrete implementations in the subclasses. For example:

```
public abstract class LeaseTerm {
    public abstract BigDecimal getEffectiveValue();
    ...
}

public class LeaseTermForIndexableTerm extends LeaseTerm {
    public BigDecimal getEffectveValue() { ... }
    ...
}
```

Currently the Wicket viewer will not render the property in tables (though the property is correctly rendered in views).

For more background on this workaround, see ISIS-582.

The work-around is simple enough; make the method concrete in the superclass and return a dummy implementation, eg:

```
public abstract class LeaseTerm {
    public BigDecimal getEffectiveValue() {
        return null;        // workaround for ISIS-582
    }
    ...
}
```

Alternatively the implementation could throw a `RuntimeException`, eg

```
throw new RuntimeException("never called; workaround for ISIS-582");
```

6.11. Pushing Changes (deprecated)

This technique is much less powerful than using ../ugfun/ugfun.adoc#_ugfun_building-blocks_events_domain-events[the event bus] or an SPI service. We present it mostly for completeness.

6.11.1. When a property is changed

If you want to invoke functionality whenever a property is changed by the user, then you can create a supporting `modifyXxx()` method and include the functionality within that. The syntax is:

```
public void modifyPropertyName(PropertyType param) { ... }
```

Why not just put this functionality in the setter? Well, the setter is used by the object store to recreate the state of an already persisted object. Putting additional behaviour in the setter would cause it to be triggered incorrectly.

For example:

```
public class Order() {
    public Integer getAmount() { ... }
    public void setAmount(Integer amount) { ... }
    public void modifyAmount(Integer amount) { ①
        setAmount(amount);  ③
        addToTotal(amount); ②
    }
    ...
}
```

① The `modifyAmount()` method calls ...

② ... the `addToTotal()` (not shown) to maintain some running total.

We don't want this `addToCall()` method to be called when pulling the object back from the object store, so we put it into the modify, not the setter.

You may optionally also specify a `clearXxx()` which works the same way as modify `modify Xxx()` but is called when the property is cleared by the user (i.e. the current value replaced by nothing). The syntax is:

```
public void clearPropertyName() { ... }
```

To extend the above example:

```
public class Order() {
    public Integer getAmount() { ... }
    public void setAmount(Integer amount) { ... }
    public void modifyAmount(Integer amount) { ... }
    public void clearAmount() {
        removeFromTotal(this.amount);
        setAmount(null);
    }
    ...
}
```

6.11.2. When a collection is modified

A collection may have a corresponding `addToXxx()` and/or `removeFromXxx()` method. If present, and direct manipulation of the contents of the connection has not been disabled (see ?), then they will be called (instead of adding/removing an object directly to the collection returned by the accessor).

The reason for this behaviour is to allow other behaviour to be triggered when the contents of the collection is altered. That is, it is directly equivalent to the supporting `modifyXxx()` and `clearXxx()` methods for properties (see ?).

The syntax is:

```
public void addTo<CollectionName>(EntityType param) { ... }
```

and

```
public void removeFromCollectionName(EntityType param) { ... }
```

where `EntityType` is the same type as the generic collection type.

For example:

```
public class Employee { ... }

public class Department {

    private int numMaleEmployees;                              ①
    private int numFemaleEmployees;                            ②

    private Set<Employee> employees = new TreeSet<Employee>();
    public Set<Employee> getEmployees() {
        return employees;
    }
    private void setEmployees(Set<Employee> employees) {
        this.employees = employees;
    }
    public void addToEmployees(Employee employee) {            ③
        numMaleEmployees += countOneMale(employee);
        numFemaleEmployees += countOneFemale(employee);
        employees.add(employee);
    }
    public void removeFromEmployees(Employee employee) {       ④
        numMaleEmployees -= countOneMale(employee);
        numFemaleEmployees -= countOneFemale(employee);
        employees.remove(employee);
    }
    private int countOneMale(Employee employee) { return employee.isMale()?1:0; }
    private int countOneFemale(Employee employee) { return employee.isFemale()?1:0; }

    ...
}
```

① maintain a count of the number of male ...

② ... and female employees (getters and setters omitted)

③ the `addTo···()` method increments the derived properties

④ the `removeFrom···()` method similarly decrements the derived properties

6.12. How to implement a spellchecker?

From this thread on the Apache Isis users mailing list:

- *What is the easiest way to add a spell checker to the text written in a field in a domain object, for instance to check English syntax?*

One way to implement is to use the event bus:

- Set up a domain event subscriber that can veto the changes.
- if the change is made through an action, you can use `@Action#domainEvent()`.

if if the change is made through an edit, you can use `@Property#domainEvent()`.

You'll need some way to know which fields should be spell checked. Two ways spring to mind:

- either look at the domain event's identifier
- or subclass the domain event (recommended anyway) and have those subclass events implement some sort of marker interface, eg a `SpellCheckEvent`.

And you'll (obviously) also need some sort of spell checker implementation to call.

Chapter 7. Developers' Guide

7.1. Datanucleus Enhancer

DataNucleus is the reference implementation of the JDO (Java data objects) spec, and Apache Isis integrates with DataNucleus as its persistence layer. Datanucleus is a very powerful library, allowing domain entities to be mapped not only to relational database tables, but also to NoSQL stores such as Neo4J, MongoDB and Apache Cassandra.

With such power comes a little bit of complexity to the development environment: all domain entities must be enhanced through the DataNucleus enhancer.

Bytecode enhancement is actually a requirement of the JDO spec; the process is described in outline here.

What this means is that the enhancer — available as both a Maven plugin and as an Eclipse plugin — must, one way or another, be integrated into your development environment.

If working from the Maven command line, JDO enhancement is done using the `maven-datanucleus-plugin`.

Both the HelloWorld and SimpleApp Maven archetypes generate applications that have this plugin pre-configured.

7.1.1. META-INF/persistence.xml

It's also a good idea to ensure that every domain module(s) containing entities has a JDO `META-INF/persistence.xml` file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

    <persistence-unit name="simple"> ①
    </persistence-unit>
</persistence>
```

① change as required; typically is the name of the domain module.

Again, the applications generated by both the HelloWorld and Simpleapp Maven archetypes do this.

If running on Windows, then there's a good chance you'll hit the maximum path length limit. In this case the `persistence.xml` file is mandatory rather than optional.

This file is also required if you are using developing in Eclipse and relying on the DataNucleus plugin for Eclipse rather than the DataNucleus plugin for Maven. More information can be found here.

7.2. Enabling Logging

Sometimes you just need to see what is going on. There are various ways in which logging can be enabled, here are the ones we tend to use.

- In Apache Isis

  Modify `WEB-INF/logging.properties` (a log4j config file)

- In DataNucleus

  As per the DN logging page

- In the JDBC Driver

  Configure `log4jdbc` JDBC rather than the vanilla driver (see `WEB-INF/persistor_datanucleus.properties`) and configure log4j logging (see `WEB-INF/logging.properties`). There are examples of both in the SimpleApp archetype.

- In the database

  Details below.

Database logging can be configured:

- for HSQLDB

  by adding `;sqllog=3` to the end of the JDBC URL.

- for PostgreSQL:

  Can change `postgresql\9.2\data\postgresql.conf`; see this article for details.

- for MS SQL Server Logging:

  We like to use the excellent SQL Profiler tool.

7.3. Enhance only (IntelliJ)

From the Apache Isis mailing list is:

- *Is there a simple way to make a run configuration in IntelliJ for running the datanucleus enhancer*

before running integration test?

Yes, you can; here's one way:

- Duplicate your run configuration for running the webapp
  - the one where the main class is `org.apache.isis.WebServer`
  - there's a button for this on the run configurations dialog.
- then, on your copy change the main class to `org.apache.isis.Dummy`

Or, you could just write a small shell script and run from the command line:

enhance.sh

```
mvn -pl dom datanucleus:enhance -o
```

7.4. How run fixtures on startup?

From this thread on the Apache Isis users mailing list:

- *my fixtures have grown into a couple of files the application needs to read in when it starts the first time (and possibly later on when the files content change). What is the right way to do this? Hook up into the webapp start? Use events?*

The standard approach is to use fixture scripts. These can be run in on start-up typically by being specified in the `AppManifest`, see for example the SimpleApp archetype.

Alternatively just set `isis.fixtures` and `isis.persistor.datanucleus.install-fixtures` properties.

In terms of implementations, you might also want to check out the (non-ASF) Incode Platform's excel module, by using `ExcelFixture` and overriding `ExcelFixtureRowHandler`.

An example can be found in this (non ASF) contactapp, see `ContactRowHandler`.