Alias Subdomain

Table of Contents

Domain Model 2
Screenshots 3
How to configure/use 7
  Classpath 7
  Bootstrapping 7
  For each domain object... 7
  SPI services 9
UI Concerns 11
Other Services 13
Known issues 14
Dependencies 15

This module (`incode-module-alias`) provides the ability to attach `Alias` objects to arbitrary domain entities, where "alias" means an alternative identifier for the entity. Most commonly this is the identifier within some external system (in DDD terms: in some other bounded context).

The identity of the external system itself is represented in two ways: first by the "alias type" (general ledger, document management system" etc), and also by an "application tenancy path". The latter is the same concept as exists within the Security SPI module, and generally represents a multi-tenancy environment, eg to distinguish different countries. Thus, an `Alias` is the combination of the app tenancy, an alias type, and the alias reference itself (a simple string).

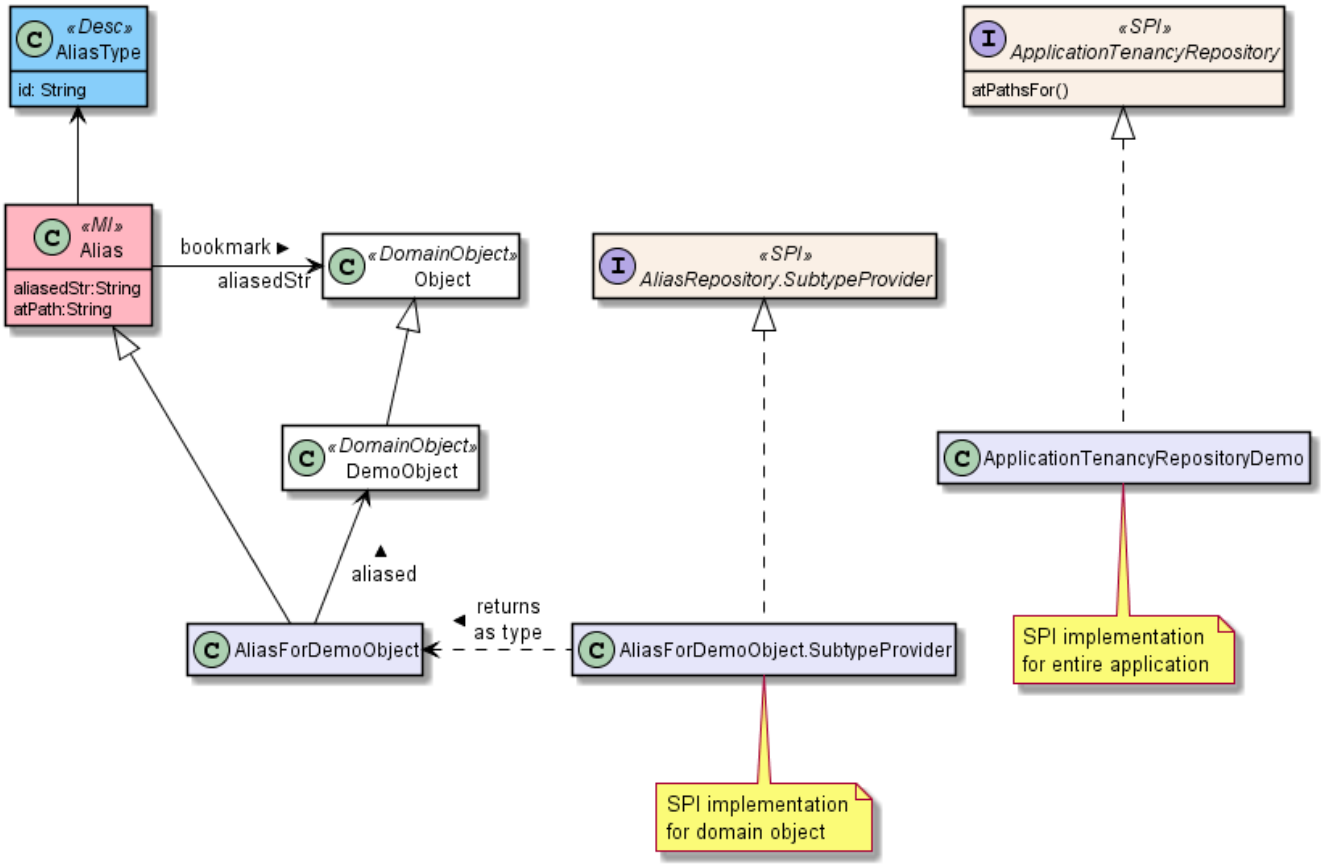
For example, a `Party` entity might have a different identifier within a general ledger system for France say, and another identifier again in centrally-managed document management system.

There are *no* requirements for domain objects with aliases implement any interfaces. Instead, `Alias` is an abstract class; the consuming application provides a subclass of `Alias` and related classes to act as the "glue" between the `Alias` and the "aliased" domain object. In total about 50 lines of boilerplate are required, details below.

This module places no constraints on either the alias type nor on the format of the application tenancy path. The alias type is represented by the `AliasType`; to use the module the application must implement the mandatory `AliasTypeRepository` SPI service which identifies the available `AliasType`s. Similarly, the available application tenancy paths are provided to the module by the mandatory `ApplicationTenancyRepository` SPI service. In both cases the list of alias types and application tenancy paths is per aliased domain object; not every aliased object will have an alias in every external system.

Domain Model

The following class diagram highlights the main concepts:

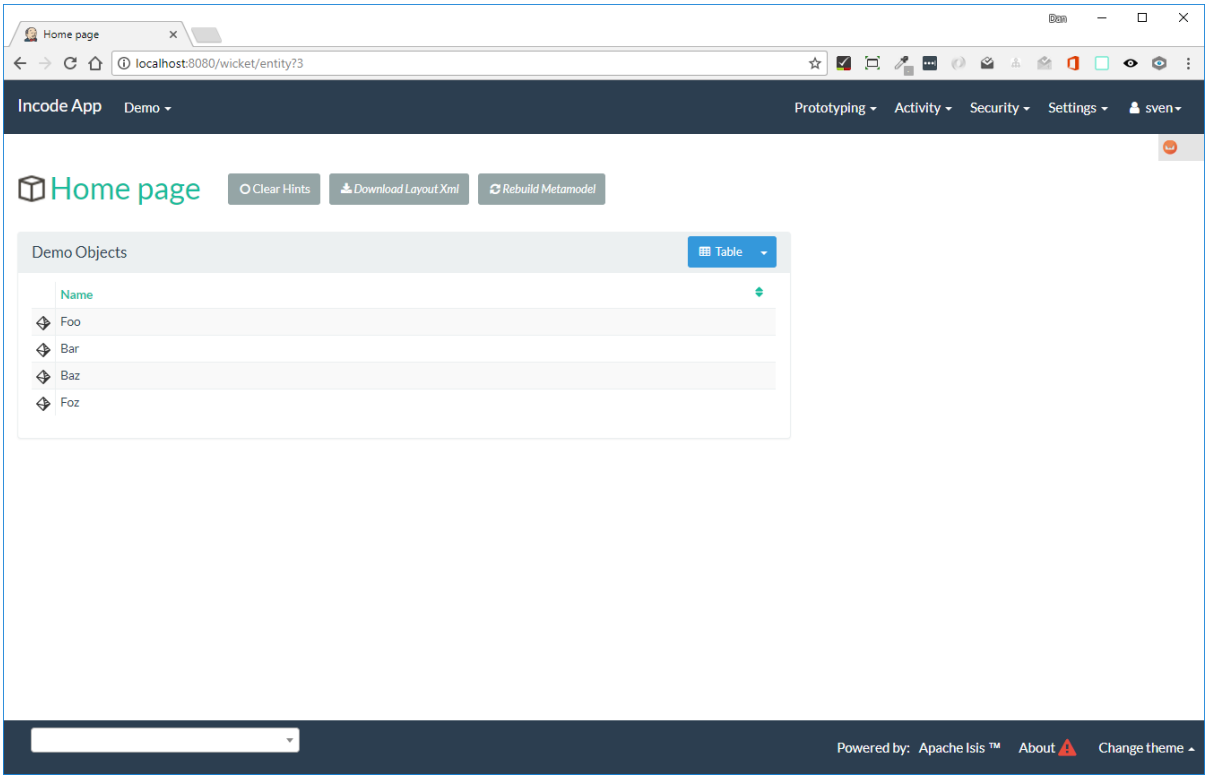


(The colours used in the diagram are - approximately - from Object Modeling in Color).

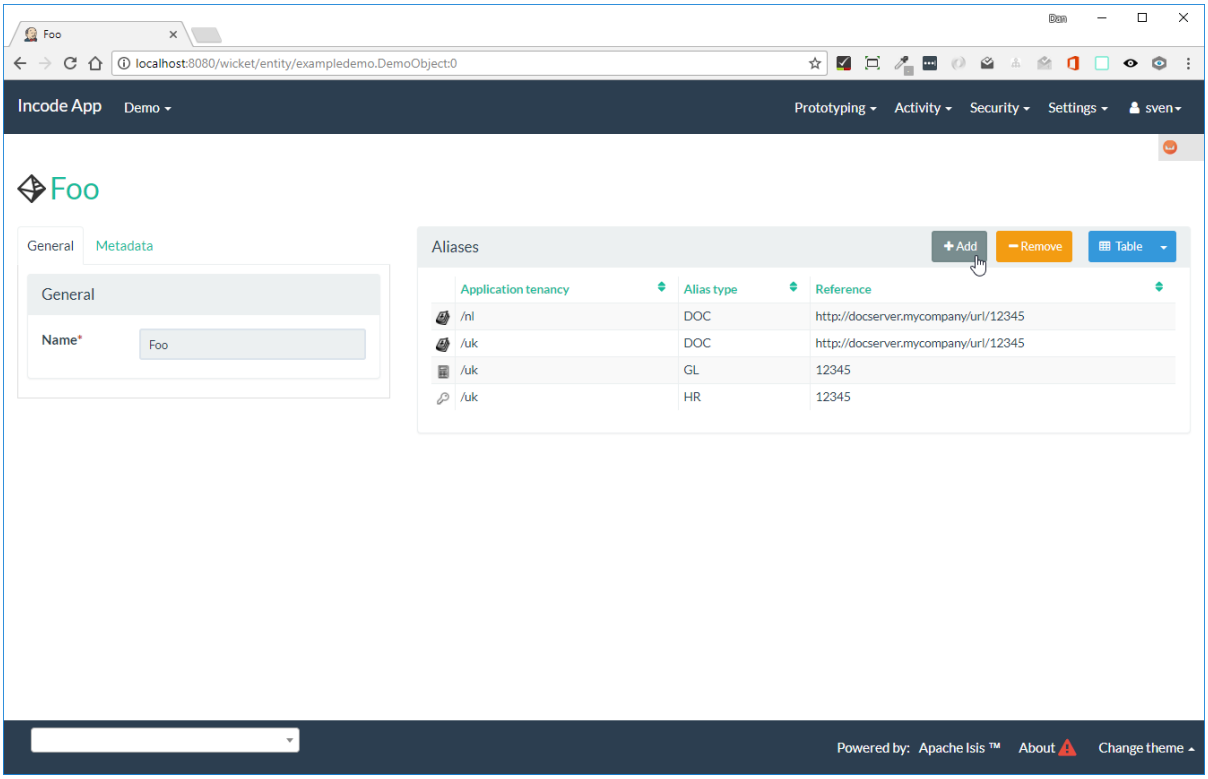
Screenshots

The module's functionality can be explored by running the quickstart with example usage using the `org.incode.domainapp.example.app.modules.ExampleDomDomAliasAppManifest`.

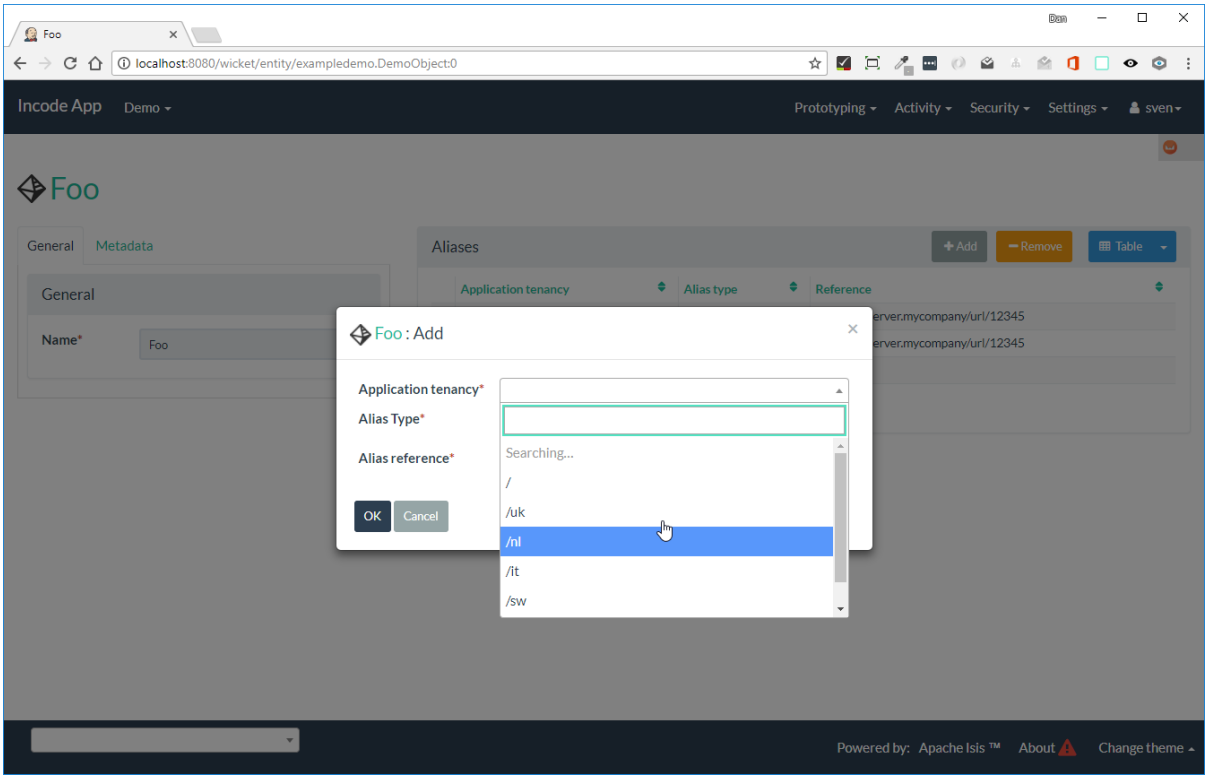
A home page is displayed when the app is run this way:



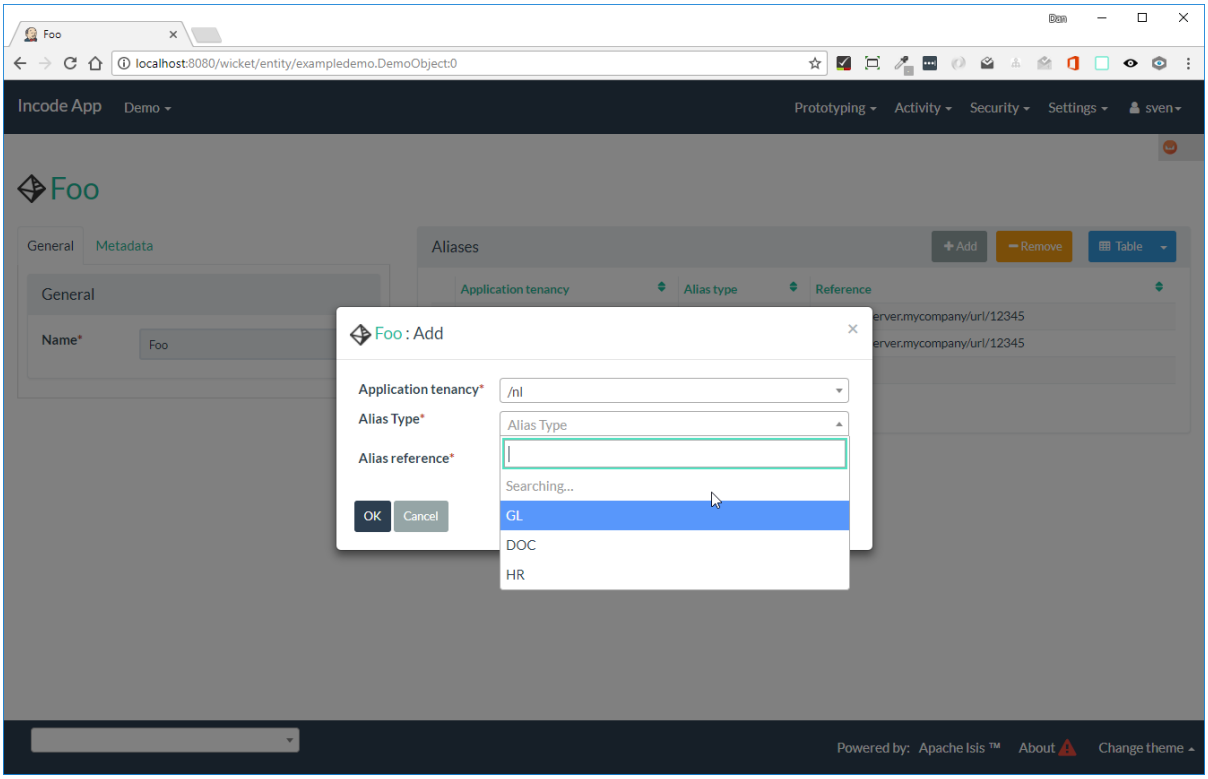
These "aliasable" demo objects have a (contributed) `aliases` collection, and we can also add new aliases using a (contributed) `addAlias(…)` action:



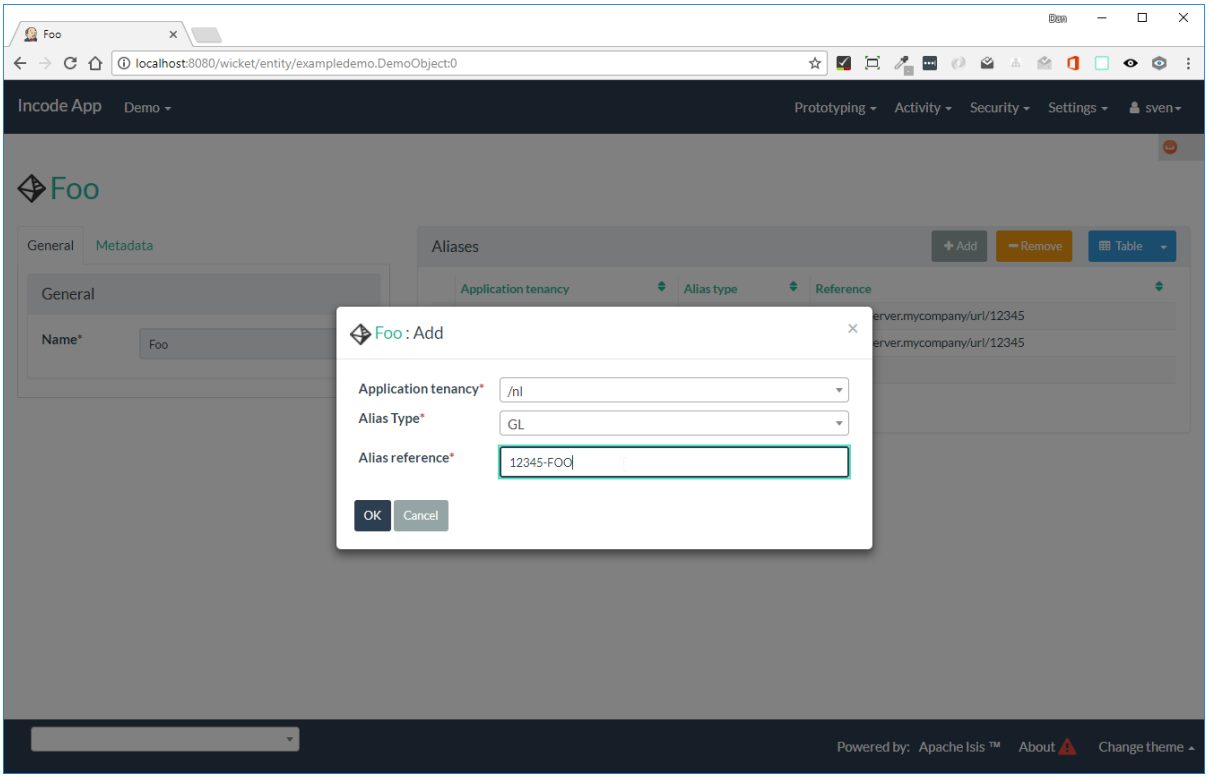
The action requires the application tenancy of the alias (as returned from the `ApplicationTenancyRepository` SPI service) to be specified:



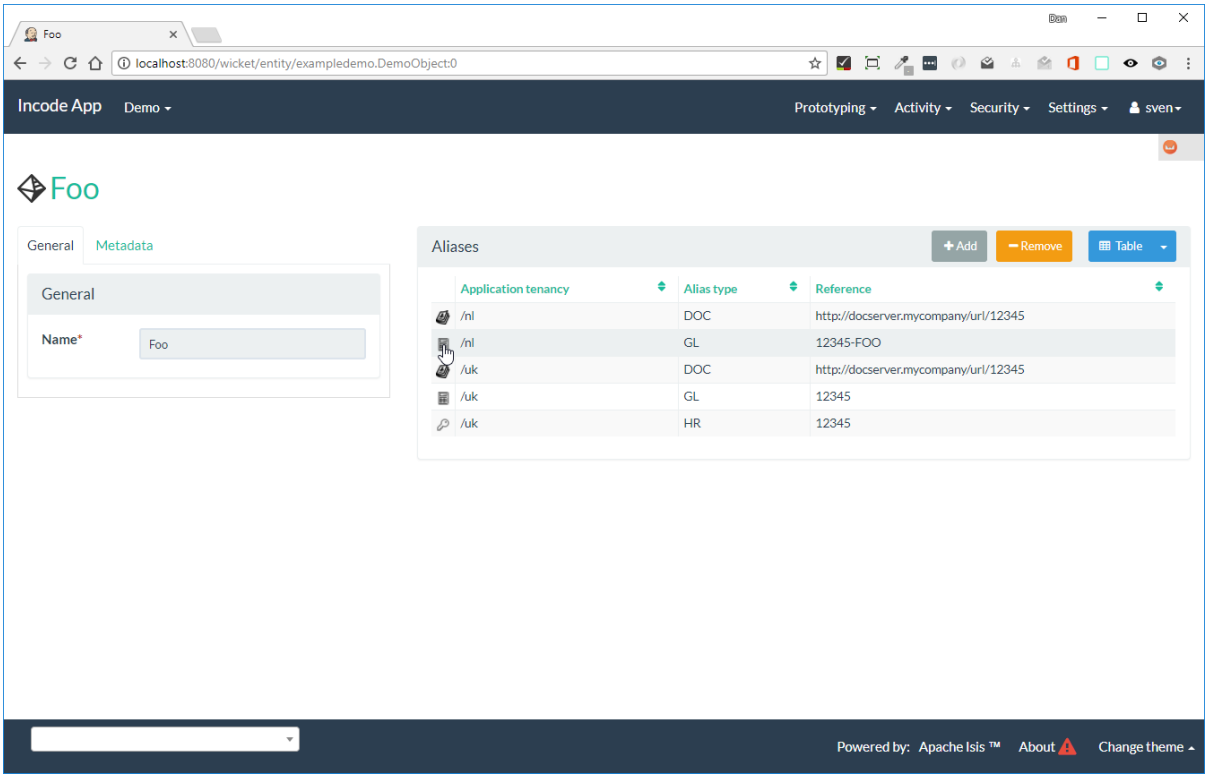
and also the alias type (as returned from the `AliasTypeRepository` SPI service) to be specified:



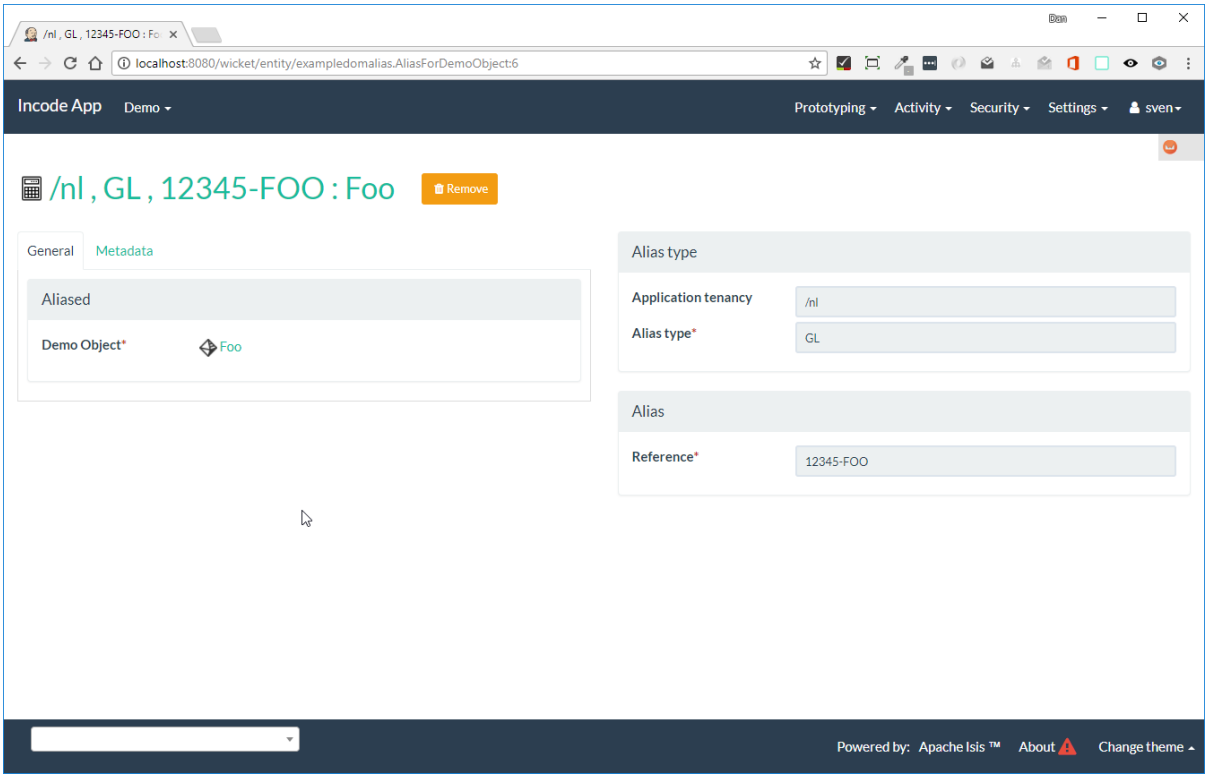
and finally the external alias reference itself must be specified also:



The aliases for the `Alias` domain object is added to:



Each `Alias` can also be viewed:


/nl , GL , 12345-FOO : Foo
localhost:8080/wicket/entity/exampledomalias.AliasForDemoObject:6
Incode App
Demo
Prototyping
Activity
Security
Settings
sven
/nl , GL , 12345-FOO : Foo
Remove
General
Metadata
Aliased
Demo Object*
Foo
Alias type
Application tenancy
/nl
Alias type*
GL
Alias
Reference*
12345-FOO
Powered by: Apache Isis ™
About
Change theme

How to configure/use

Classpath

Update your classpath by adding this dependency in your dom project's `pom.xml`:

```
<dependency>
    <groupId>org.incode.module.alias</groupId>
    <artifactId>incode-module-alias-dom</artifactId>
    <version>1.15.0</version>
</dependency>
```

Check for later releases by searching Maven Central Repo.

For instructions on how to use the latest `-SNAPSHOT`, see the contributors guide.

Bootstrapping

In the `AppManifest`, update its `getModules()` method, eg:

```
@Override
public List<Class<?>> getModules() {
    return Arrays.asList(
            ...
            org.incode.module.alias.dom.AliasModule.class,
    );
}
```

For each domain object...

In order to be able to add/remove aliases to a domain object, you need to:

- implement a subclass of `Alias` for the domain object's type.

  This is the object that will be polymorphically attached to the "aliased" domain object; the subtype provides the type-safe association (a foreign key within the RDBMS).

- implement the `SubtypeProvider` SPI interface:

  ```
  public interface SubtypeProvider {
      Class<? extends Alias> subtypeFor(Class<?> domainObject);
  }
  ```

  This tells the module which subclass of `Alias` to use to attach to the "aliased" domain object. The `SubtypeProviderAbstract` adapter can be used to remove some boilerplate.

- subclass `T_addAlias`, `T_removeAlias` and `T_aliases` (abstract) mixin classes for the domain object.

  These contribute the "aliases" collection and actions to add and remove `Alias`es.

Typically the SPI implementations and the mixin classes are nested static classes of the `Alias` subtype.

For example, in the demo app the `DemoObject` domain object can have aliases by virtue of the `AliasForDemoObject` subclass:

```
@javax.jdo.annotations.PersistenceCapable(identityType= IdentityType.DATASTORE,
schema="incodeAliasDemo")
@javax.jdo.annotations.Inheritance(strategy = InheritanceStrategy.NEW_TABLE)
@DomainObject(objectType = "incodeAliasDemo.AliasForDemoObject")
public class AliasForDemoObject extends Alias {
①

    private DemoObject demoObject;
    @Column(allowsNull = "false", name = "demoObjectId")
    @Property(editing = Editing.DISABLED)
    public AliasDemoObject getDemoObject() {
②
        return demoObject;
    }
    public void setDemoObject(final AliasDemoObject demoObject) {
        this.demoObject = demoObject;
    }

    public Object getAliased() {
③
        return getDemoObject();
    }
    protected void setAliased(final Object aliased) {
        setDemoObject((AliasDemoObject) aliased);
    }

    @DomainService(nature = NatureOfService.DOMAIN)
    public static class SubtypeProvider extends AliasRepository
.SubtypeProviderAbstract {   ④
        public LinkProvider() {
            super(DemoObject.class, AliasForDemoObject.class);
        }
    }

    @Mixin
    public static class _aliases extends T_aliases<DemoObject> {
⑤
        public _aliases(final AliasDemoObject aliased) {
            super(aliased);
        }
```

```
    }
    @Mixin
    public static class _addAlias extends T_addAlias<DemoObject> {
        public _addAlias(final AliasObject aliased) {
            super(aliased);
        }
    }
    @Mixin
    public static class _removeAlias extends T_removeAlias<DemoObject> {
        public _removeAlias(final DemoObject aliased) {
            super(aliased);
        }
    }
}
```

① extend from `Alias`

② the type-safe reference property to the "aliased" domain object (in this case `DemoObject`). In the RDBMS this will correspond to a regular foreign key with referential integrity constraints correctly applied.

③ implement the hook `setAliased(…)` method to allow the type-safe reference property to the "aliased" (in this case `DemoObject`) to be set. Also implemented `getAliased()` similarly

④ implementation of the `SubtypeProvider` SPI domain service, telling the module which subclass of `Alias` to instantiate to attach to the "aliased" domain object

⑤ mixins for the collections and actions contributed to the "aliased" domain object

SPI services

There are two further mandatory SPI domain services that must be implemented:

- First, the `ApplicationTenancyRepository` returns the application tenancy (path)s that are available to locate alias types for a given aliased:

  ```
  public interface ApplicationTenancyRepository {
      Collection<String> atPathsFor(final Object domainObjectToAlias);
  }
  ```

  Note that this isn't (necessarily) the same as the application tenancy path of the object being aliased; rather it is the list of the paths available (eg: countries/regions) for which there is an alias type (eg an external system) that may contain an alias (external system identifier).

- Second, the `AliasTypeRepository` interface returns the available alias types for a given application tenancy path and aliased:

```
public interface AliasTypeRepository {
    Collection<AliasType> aliasTypesFor(final Object aliased, final String atPath);
}
```

where `AliasType` is defined as the interface:

```
public interface AliasType {
    String getId();
}
```

Typically `AliasType` will be implemented as an entity or perhaps a view model. The "id" is used as a column in the database tables, but in the UI the end-user sees the title of the object that implements the interface.

The Apache Isis framework curently (as of v1.14.0) does not support enums implementing interfaces; the example app shows how a view model can be used as a work-around.

Note that there can be multiple implementations of either of these interfaces. This is to support the use case that different unrelated entities in the application may have aliases; each such aliased object can have its own supporting implementations of these SPI interfaces.

UI Concerns

The attached `Alias` objects are shown in two contexts: as a table of `Alias` objects for the "aliased" domain object, and then as the actual subtype when the alias object itself is shown (eg `AliasForDemoObject` in the demo app).

In the former case (as a table) the `Alias` will be rendered according to the `Alias.layout.xml` provided by the module. In the latter (as an object) the alias will be rendered according to the layout provided by the consuming app, offering full control of the layout. The layout provided in the demo app (ie `AliasForDemoObject.layout.xml`) is a good starting point.

The example `AliasForDemoObject.layout.xml` uses a little bit of custom CSS to adjust the right-hand column down a number of pixels. This resides in `application.css`:

```
.entityPage.org-incode-module-alias-fixture-dom-alias-
AliasForDemoObject .alias-col {
    padding-top: 41px;
}
```

The module also allows the title, icon and CSS for `Alias` objects to be customised. By default the values for these are obtained using default subscribers, namely - `Alias.TitleSubscriber`, `Alias.IconSubscriber` and `Alias.CssClassSubscriber`. The consuming module can override these values simply by providing alternative implementations.

For example, the demo app has this demo implementation:

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

    @Subscribe
    public void on(Alias.TitleUiEvent ev) {
        Alias alias = ev.getSource();
        if(isType(alias, AliasTypeDemoEnum.DOCUMENT_MANAGEMENT)) {
            ev.setTitle("DocMgmt  [" + alias.getAliasTypeId() + "] " + alias
.getReference());
        }
    }

    @Subscribe
    public void on(Alias.IconUiEvent ev) {
        Alias alias = ev.getSource();
        if(isType(alias, AliasTypeDemoEnum.DOCUMENT_MANAGEMENT)) {
            ev.setIconName("Alias-docMgmt");
        } else if (isType(alias, AliasTypeDemoEnum.GENERAL_LEDGER)) {
            ev.setIconName("Alias-GL");
        }
    }

    @Subscribe
    public void on(Alias.CssClassUiEvent ev) {
        Alias alias = ev.getSource();
        ev.setCssClass("Alias" + alias.getAtPath().replace("/", "-"));
    }

    private static boolean isType(final Alias alias, final AliasTypeDemoEnum
aliasType) {
        return alias.getAliasTypeId().equals(aliasType.getId());
    }
}
```

which returns a different title, icon and alias.

The custom png icons are picked up from `org.incode.module.alias.dom.impl` package (in this case, `Alias-docMgmt.png` and `Alias-GL.png`. The custom CSS is supplied in the `application.css` of the demo app:

```
tr.Alias-nl {
    color: blueviolet;
}
tr.Alias-uk {
    color: chocolate;
}
```

Other Services

The module provides one further domain service, namely `AliasRepository`. This can be used for finding the aliases attached to an "aliased" object.

Known issues

(As noted above), as of v1.14.0 the Apache Isis framework does not support enums implementing interfaces; the example app shows how a view model can be used as a work-around.

Dependencies

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/dom/alias/impl -D excludeTransitive=true
```

which, excluding the Apache Isis modules, returns no direct compile/runtime dependencies.

The module *does* use icons from icons8.