Togglz Extension

Table of Contents

This module (`isis-module-togglz`) provides an integration with Togglz to provide a feature toggle capability.

Courtesy of Togglz, this integration has an embedded console and has support for integration testing through a custom JUnit rule.

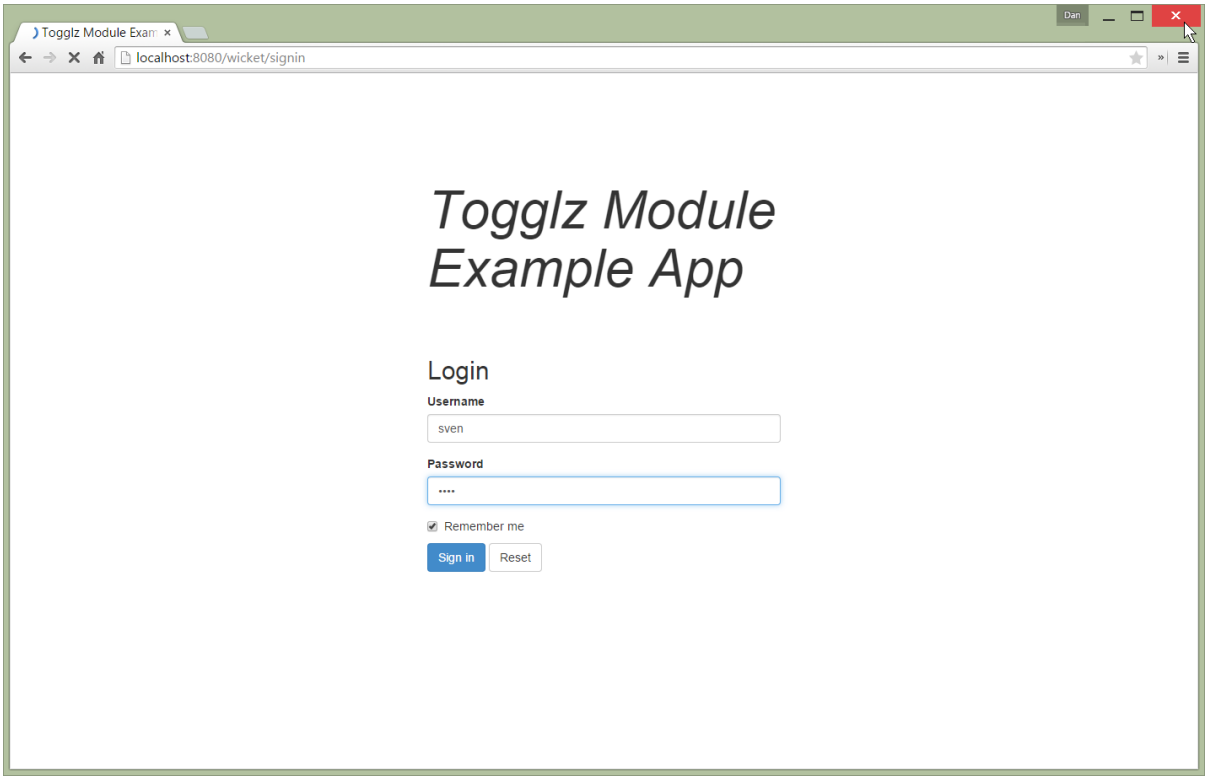
The module integrates both Togglz and uses the settings subdomain for feature persistence.

Screenshots

The screenshots below **do** demonstrate the functionality of this module, but are out of date in that they are taken from the original isisaddons/incodehq module (prior to being amalgamated into the incode-platform).

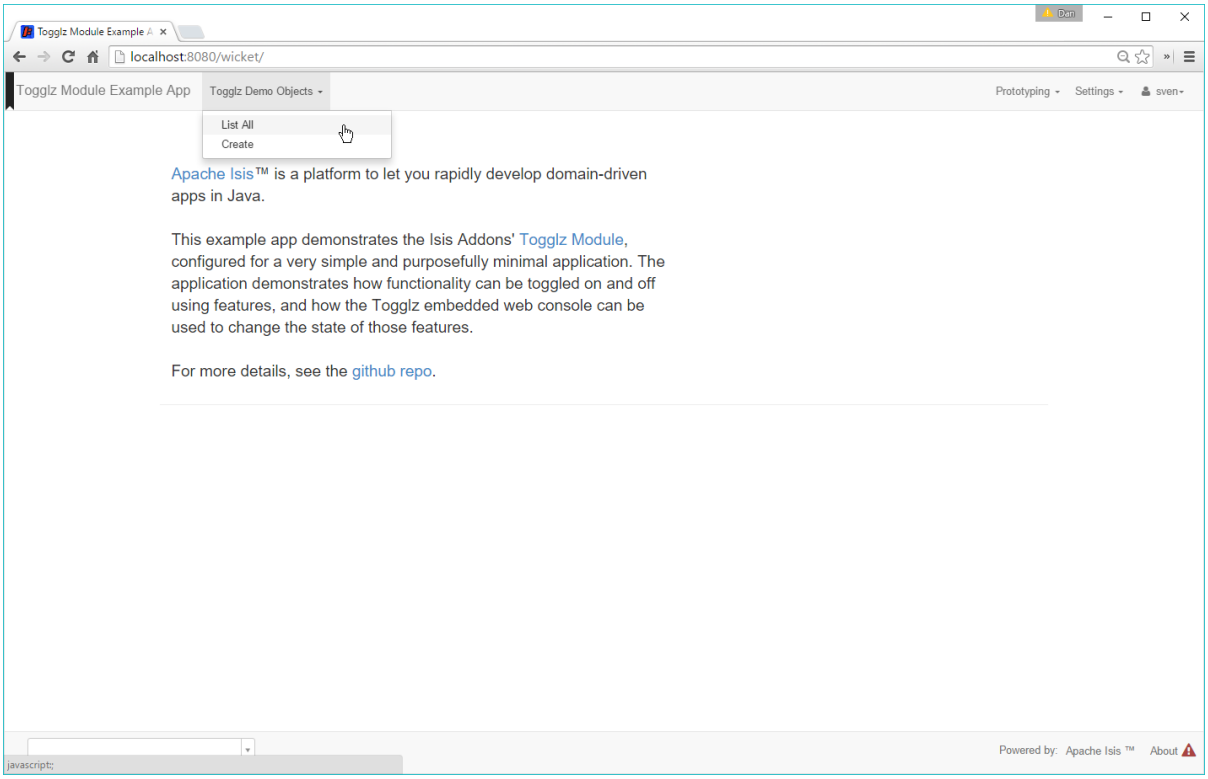
The following screenshots show an example app's usage of the module.

Login as administrator



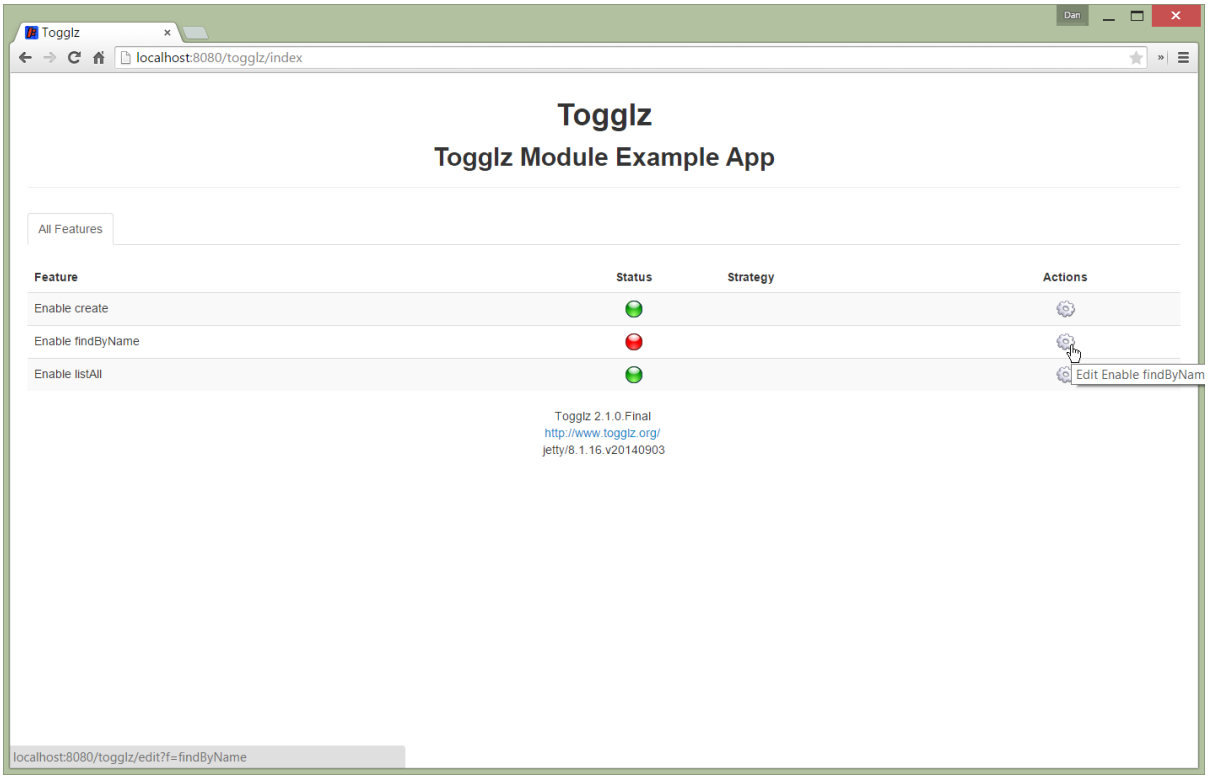
Feature disabled

In the demo app the "Togglz Demo Objects" service has three actions, all of which are protected behind features. Two of these (for "create" and "listAll") are enabled by default, but one (for "findByName") is disabled by default, meaning that the action is suppressed from the UI:

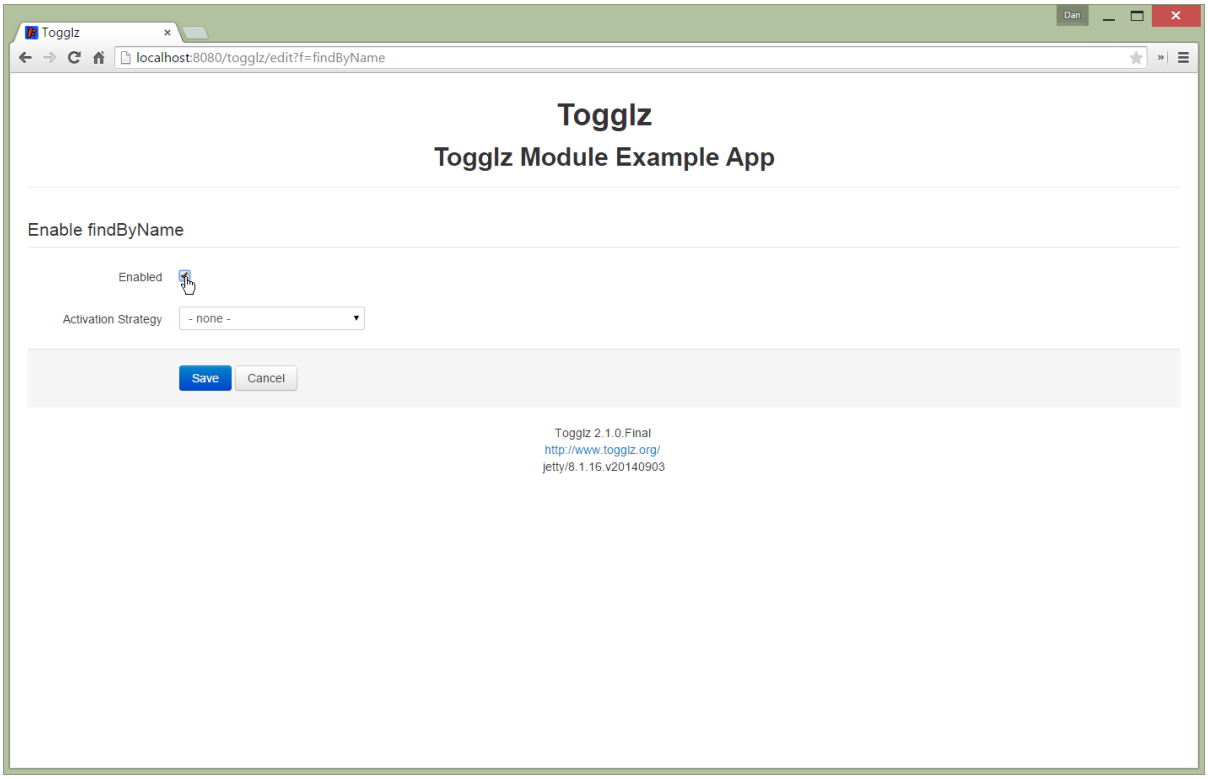


Togglz Console

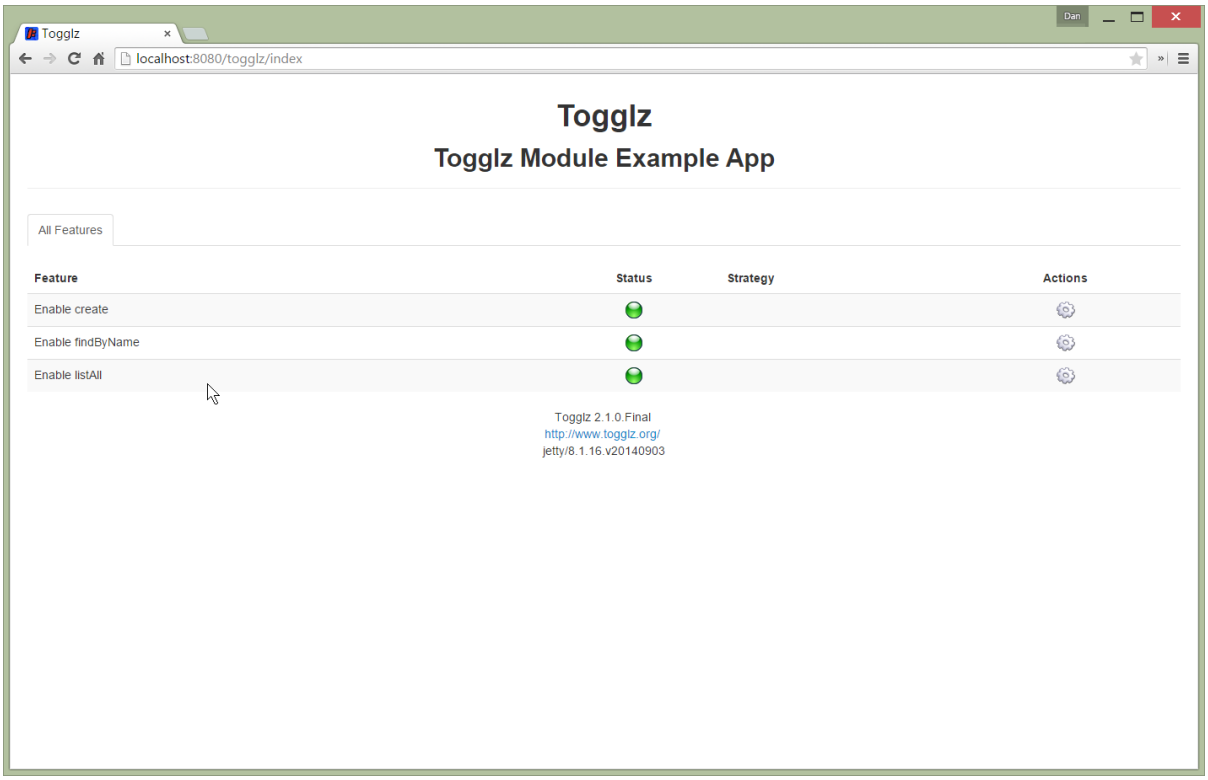
Users with the appropriate role (`isis-module-togglz-admin`) can access the Togglz console, which lists all features:



Using the console, we can edit the feature:

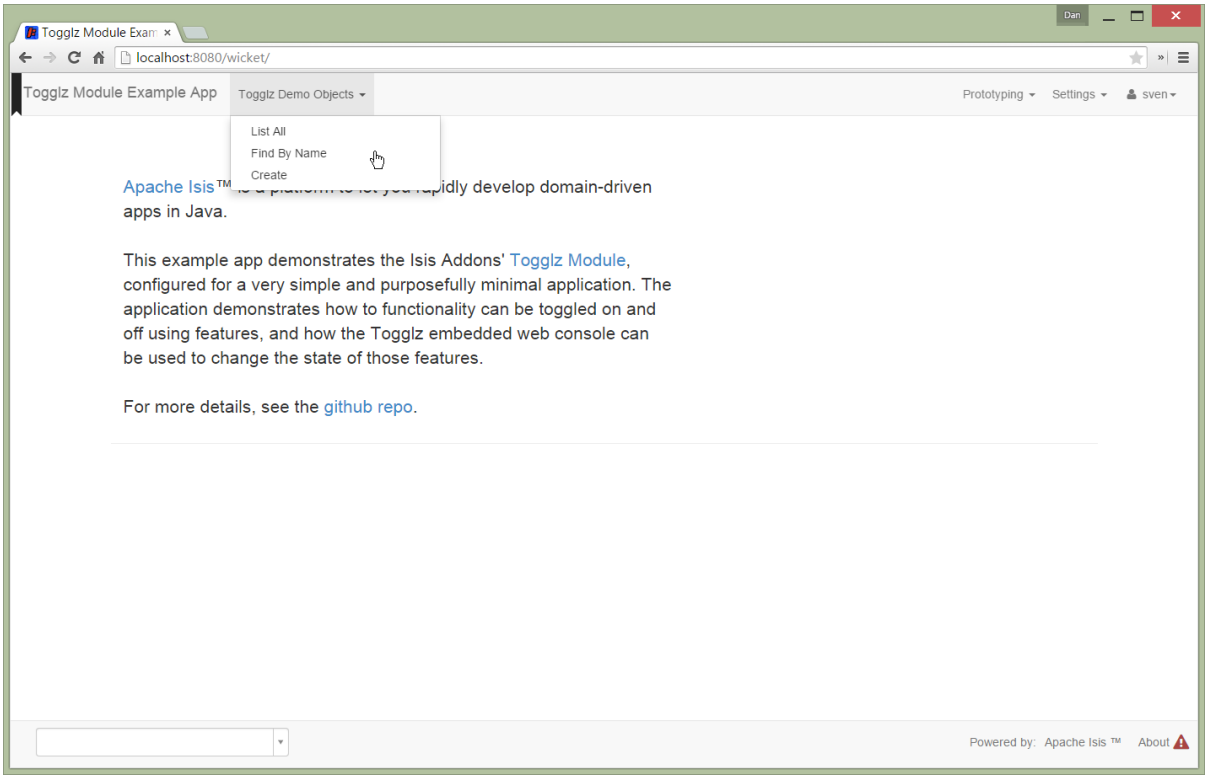


so it is now enabled:



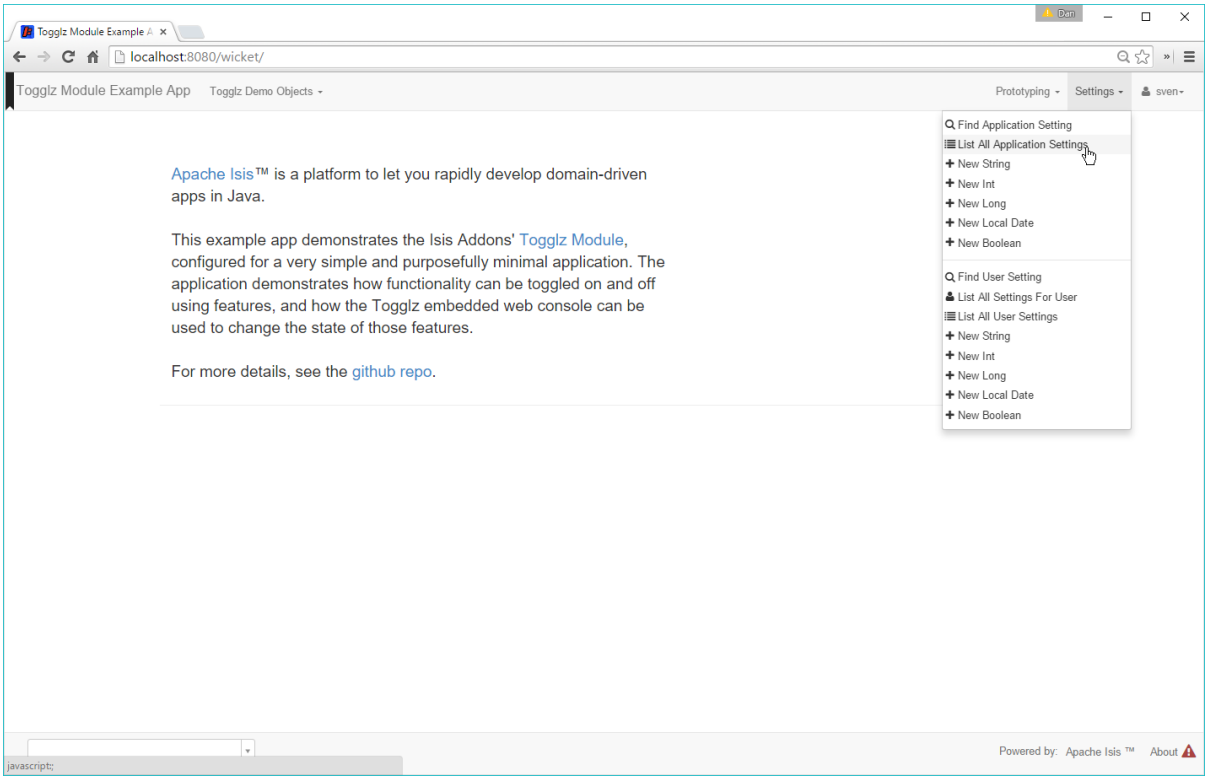
Feature enabled

Back in the demo app the feature ("findByName") is now visible:

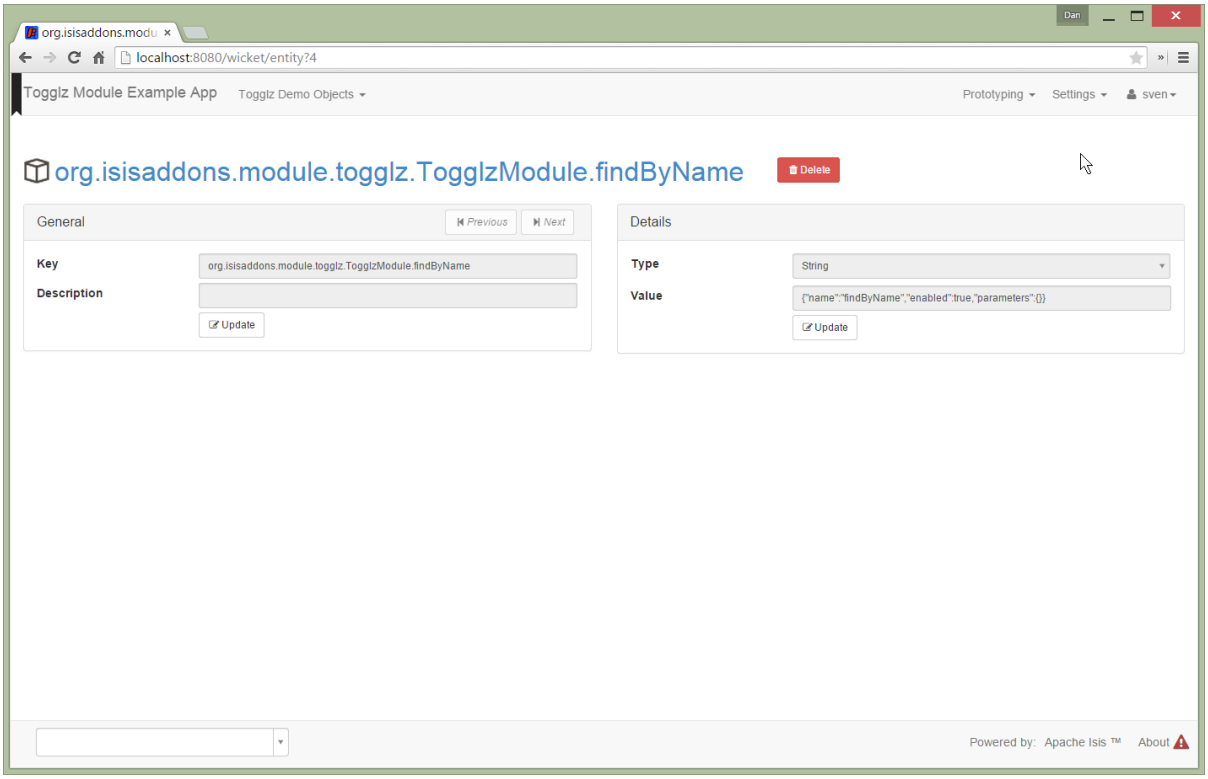


Feature persistence

The module uses settings subdomain module for feature persistence.



Each feature's state is serialized to/from JSON:


org.isisaddons.modu
localhost:8080/wicket/entity?4
Dan
Togglz Module Example App
Togglz Demo Objects
Prototyping
Settings
sven
org.isisaddons.module.togglz.TogglzModule.findByName
Delete
General
Previous
Next
Key
org.isisaddons.module.togglz.TogglzModule.findByName
Description
Update
Details
Type
String
Value
{"name":"findByName","enabled":true,"parameters":{}}
Update
Powered by: Apache Isis ™
About

Service SPIs

The module defines the following SPI service that must be implemented:

```
public interface FeatureStateRepository {
    FeatureState find(String key);
    FeatureState create(String key);
}
```

where `FeatureState` is just a wrapper around a string:

```
public interface FeatureState {
    String getValue();
    void setValue(String value);
}
```

This is used to persist the feature state.

How to configure/use

Classpath

Update the classpath in your project's `dom` module `pom.xml` to reference the togglz library:

```
<properties>
    <togglz.version>2.1.0.Final</togglz.version>
</properties>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-core</artifactId>
    <version>${togglz.version}</version>
</dependency>
```

+ * as described in the [Togglz documentation](http://www.togglz.org/documentation/overview.html), create a "feature enum" class that enumerates your features. This should extend from `org.togglz.core.Feature`.

+ For example, the demo app's feature enum class is:

+

```
public enum TogglzDemoFeature implements org.togglz.core.Feature {

    @Label("Enable create")
    @EnabledByDefault
    create,

    @Label("Enable findByName")
    findByName,

    @Label("Enable listAll")
    @EnabledByDefault
    listAll;

    public boolean isActive() {
      return FeatureContext.getFeatureManager().isActive(this);
    }
}
```

- use your feature class in your app as required.

  For example, the demo app uses its feature enum to selectively hide actions of the `TogglzDemoObjects` domain service:

```
public class TogglzDemoObjects {
    ...
    public List<TogglzDemoObject> listAll() { ... }
    public boolean hideListAll() {
      return !TogglzDemoFeature.listAll.isActive();
    }
}
```

- in your `integtests` module, update the `pom.xml` for togglz's JUnit support:

```
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-junit</artifactId>
    <scope>test</scope>
</dependency>
```

- also in your `integtests` module, make sure that the `TogglzRule` (documented here on the togglz website) is enabled for any tests that depend on features.

  In the demo app, this means adding the following to `TogglzModuleIntegTest` base class:

```
@Rule
public TogglzRule togglzRule = TogglzRule.allEnabled(TogglzDemoFeature.class);
```

- update your classpath by adding this dependency in your project's `fixture` module's `pom.xml`:

```
<dependency>
    <groupId>org.isisaddons.module.togglz</groupId>
    <artifactId>isis-module-togglz-glue</artifactId>
    <version>1.15.0</version>
</dependency>
<dependency>
    <groupId>org.isisaddons.module.security</groupId>
    <artifactId>isis-module-security-dom</artifactId>
    <version>1.15.0</version>                                    ①
</dependency>
```

  ① or which ever is the latest version

- in your project's `app` module, write a subclass of `TogglzModuleFeatureManagerProviderAbstract` (provided by this module) that registers your feature enum:

```
public class CustomTogglzModuleFeatureManagerProvider
      extends TogglzModuleFeatureManagerProviderAbstract {
    protected CustomTogglzModuleFeatureManagerProvider() {
      super(TogglzDemoFeature.class);
    }
}
```

- also in your project's `app` module, in `src/main/resources`, register the provider by creating a file `META-INF/services/org.togglz.core.spi.FeatureManagerProvider`. Its contents is the fully qualified class name of your feature manager provider implementation.

  For example, the demo app's file consists of:

```
org.isisaddons.module.togglz.webapp.CustomTogglzModuleFeatureManagerProvider
```

- also in your project's `app` module, write an implementation of the `FeatureStateRepository` SPI service (defined by this module). This SPI service is designed to be easy to be implemented using the settings subdomain module (though you can of course use some other persistence mechanism if you wish). For example:

```
@DomainService(nature = NatureOfService.DOMAIN)
public class FeatureStateRepositoryForApplicationSettingsJdo implements
FeatureStateRepository {
    public FeatureState find(final String key) {
        final ApplicationSetting applicationSetting =
                                    applicationSettingsService.find(key);
        return FeatureStateForApplicationSettingJdo.from(applicationSetting);
    }
    public FeatureState create(final String key) {
        final ApplicationSetting applicationSetting =
                                    applicationSettingsService.newString(key, "",
"");
        return FeatureStateForApplicationSettingJdo.from(applicationSetting);
    }
    @Inject
    ApplicationSettingsServiceRW applicationSettingsService;
}
```

  and:

```
class FeatureStateForApplicationSettingJdo implements FeatureState {
    static FeatureState from(final ApplicationSetting applicationSetting) {
        return applicationSetting != null ?
                    new FeatureStateForApplicationSettingJdo(applicationSetting) :
null;
    }
    private final ApplicationSettingForJdo applicationSetting;
    private FeatureStateForApplicationSettingJdo(final ApplicationSetting
applicationSetting) {
        this.applicationSetting = (ApplicationSettingForJdo) applicationSetting;
    }
    public String getValue() {
        return applicationSetting.valueAsString();
    }
    public void setValue(final String value) {
        applicationSetting.updateAsString(value);
    }
}
```

- in your `AppManifest`, update its `getModules()` method.

```
@Override
public List<Class<?>> getModules() {
    return Arrays.asList(
            ...
            org.isisaddons.module.security.SecurityModule.class,
            org.isisaddons.module.settings.SettingsModule.class,
            org.isisaddons.module.togglz.TogglzModule.class,
            ...
    );
}
```

- in your project's `webapp` module, update your `WEB-INF/web.xml`, after the Shiro configuration but before Isis' configuration (so that the filters are applied in the order Shiro -> Togglz -> Isis):

```
<!-- bootstrap Togglz -->
<context-param>
    <param-name>org.togglz.FEATURE_MANAGER_PROVIDED</param-name>
    <!-- prevent the filter from bootstrapping
          so is done lazily later once Isis has itself bootstrapped -->
    <param-value>true</param-value>
</context-param>
<filter>
    <filter-name>TogglzFilter</filter-name>
    <filter-class>org.togglz.servlet.TogglzFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>TogglzFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- optional: if you want to install the Togglz console, then in your project's `webapp` module, update your `WEB-INF/web.xml`:

```
<!-- enable the togglz console (for FeatureToggleService) -->
<servlet>
    <servlet-name>TogglzConsoleServlet</servlet-name>
    <servlet-class>org.togglz.console.TogglzConsoleServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>TogglzConsoleServlet</servlet-name>
    <url-pattern>/togglz/*</url-pattern>
</servlet-mapping>
```

  The togglz console will be available at http://localhost:8080/togglz

- if you have configured the Togglz console (above), then you'll also need to setup users to have `isis-module-togglz-admin` role.

  The demo app uses simple Shiro-based configuration, which means updating the `WEB-INF/shiro.ini` file, eg:

```
sven = pass, admin_role, isis-module-togglz-admin
```

- if you have configured the Togglz console (above), then you can optionally configure its URL and also whether to hide the menu action provided to access the console from the main Wicket application:

  in `isis.properties` (or in `AppManifest#getConfigurationProperties()`):

```
isis.services.togglz.FeatureToggleConsoleAccessor.consoleUrl=http:///togglz     ①
isis.services.togglz.FeatureToggleConsoleAccessor.hideAction=false               ②
```

① URL that hosts the togglz console

② whether to hide the action that can be used to access the URL.

If you are using some other security mechanism, eg Isis addons security spi module, then define a role with the same name and grant to users. You can use the `TogglzModuleAdminRole` to setup fixture/seed data for the security module.

- Check for later releases by searching Maven Central Repo).
- Make sure the `togglz.version` defined in your `dom` module matches the one used in the version of the `isis-module-togglz-glue` module (currently `2.1.0.Final`).

Check for later releases by searching [Maven Central Repo](http://search.maven.org/#search|ga|1|isis-module-togglz-glue).

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

Known issues

None known at this time.

Dependencies

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/ext/togglz/impl -D excludeTransitive=true
```

which, excluding Apache Isis itself, returns these compile/runtime dependencies:

```
com.google.code.gson:gson:jar:2.3.1:compile
org.apache.geronimo.specs:geronimo-servlet_3.0_spec:jar:1.0
org.togglz:togglz-servlet:jar:2.1.0.Final
org.togglz:togglz-console:jar:2.1.0.Final
org.togglz:togglz-core:jar:2.1.0.Final
```

For further details on 3rd-party dependencies, see:

- Togglz

The quickstart app uses settings subdomain module for feature persistence.