String Interpolator (OGNL) Library

Table of Contents

This module (`isis-module-stringinterpolator`) provides a mechanism to interpolate string templates with either Isis system properties or values obtained from a domain object. It uses OGNL under the covers.

One use case for this service is in building URLs based on an object's state, parameterized by environment (prod/test/dev etc). These URLs could be anything; for example, to a reporting service:

```
${property['reportServerBase']}/ReportViewer.aspx?/InvoicesDue&propertyId=${this.property.id}
```

where:

- `${property['reportServerBase']}` is an Isis system property
- `${this.property.id}` is an expression that is evaluated in the context of a domain object (`this`), returning `this.getProperty().getId()`

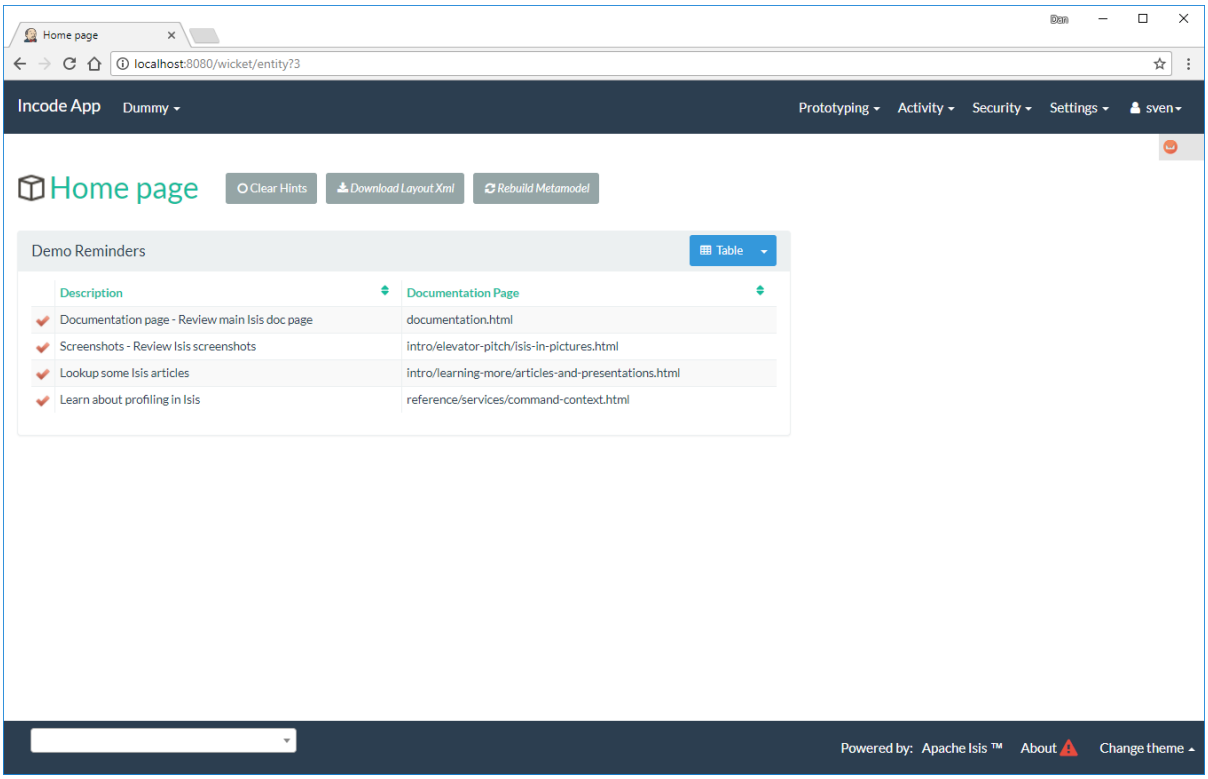
Apache Isis system properties are exposed as the `properties` map, while the target object is exposed as the `this` object.

Screenshots

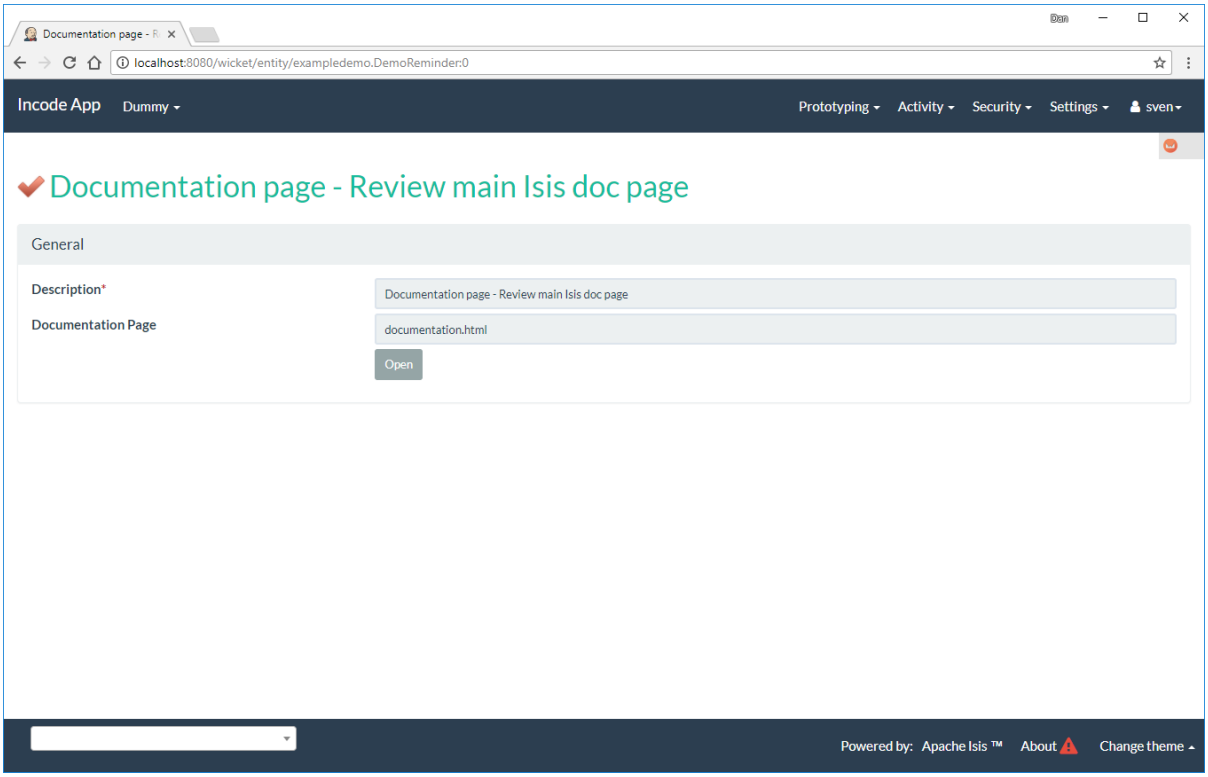
The module's functionality can be explored by running the quickstart with example usage using the `org.incode.domainapp.example.app.modules.ExampleDomLibStringInterpolatorAppManifest`.

Install example fixtures

A home page is displayed when the app is run:



... returning an example entity:



The `Open` (contributed) action is:

```
public URL open(ToDoItem toDoItem) throws MalformedURLException {
    String urlStr = stringInterpolatorService.interpolate(
        toDoItem, "${properties['isis.website']}/${this.documentationPage}");
    return new URL(urlStr);
}
```

where the configuration property "isis.website" has been set (eg in `WEB-INF/isis.properties`):

```
isis.website=http://isis.apache.org
```

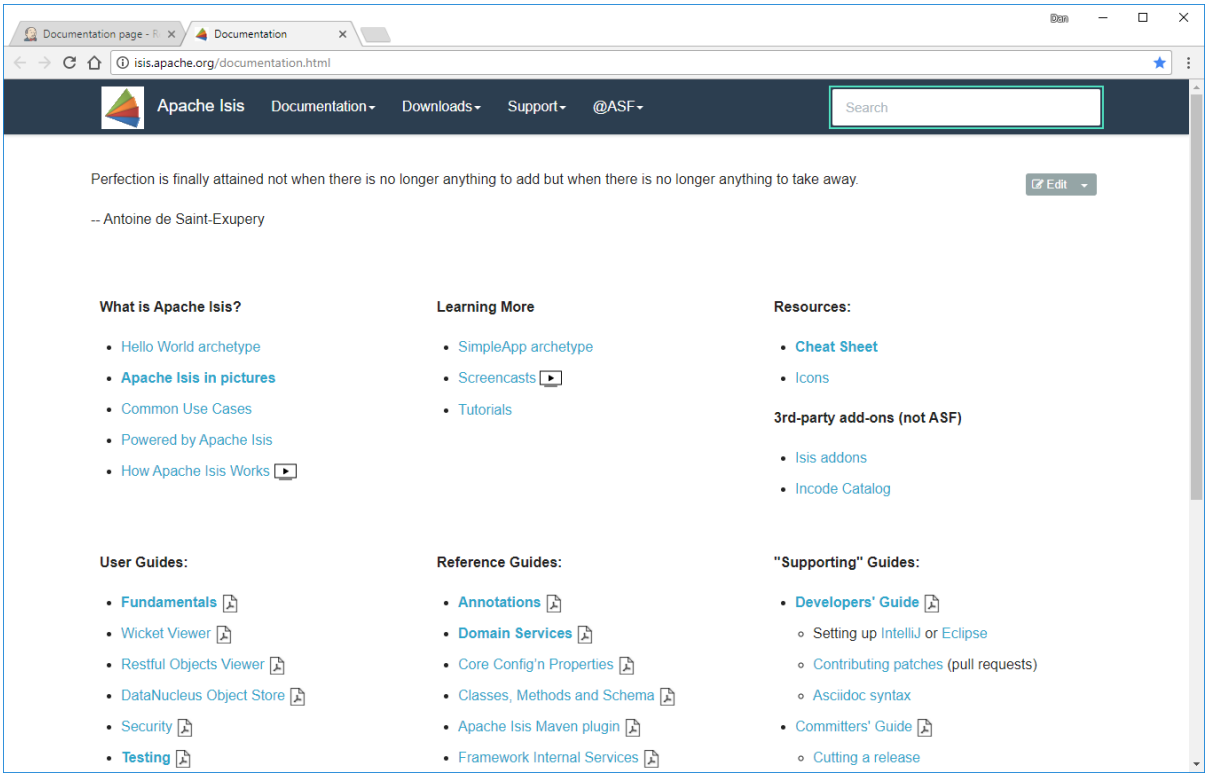
and where (as the screenshot shows) `ToDoItem` entity has the structure:

```
public class ToDoItem ... {

    private String description;
    private String documentationPage;

    // getters and setters omitted
}
```

Invoking the `Open` action computes the `urlStr` local variable, and then (because the action returns a `URL`), results in the browser opening the appropriate web page:


Documentation page - R
Documentation
isis.apache.org/documentation.html
Apache Isis
Documentation
Downloads
Support
@ASF
Search
Perfection is finally attained not when there is no longer anything to add but when there is no longer anything to take away.
-- Antoine de Saint-Exupery
Edit
What is Apache Isis?
Hello World archetype
Apache Isis in pictures
Common Use Cases
Powered by Apache Isis
How Apache Isis Works
Learning More
SimpleApp archetype
Screencasts
Tutorials
Resources:
Cheat Sheet
Icons
3rd-party add-ons (not ASF)
Isis addons
Incode Catalog
User Guides:
Fundamentals
Wicket Viewer
Restful Objects Viewer
DataNucleus Object Store
Security
Testing
Reference Guides:
Annotations
Domain Services
Core Config'n Properties
Classes, Methods and Schema
Apache Isis Maven plugin
Framework Internal Services
"Supporting" Guides:
Developers' Guide
Setting up IntelliJ or Eclipse
Contributing patches (pull requests)
Asciidoc syntax
Committers' Guide
Cutting a release

How to configure/use

Classpath

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

```
<dependency>
    <groupId>org.isisaddons.module.stringinterpolator</groupId>
    <artifactId>isis-module-stringinterpolator-dom</artifactId>
    <version>1.16.1</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.isisaddons.module.stringinterpolator.StringInterpolatorModule.class,
            ...
    );
}
```

API and Usage

The module consists of a single domain service, `StringInterpolatorService`.

The interpolation replaces each occurrence of `${⋯}` with its interpolated value. The expression in within the braces is interpreted using OGNL.

Object graph interpolation

The main API exposed by this service provides object-graph interpolation:

```
public class StringInterpolatorService {

    @PostConstruct
    public void init(                                       ①
        Map<String,String> properties) { ... }

    public String interpolate(                              ②
        Object domainObject, String template) { ... }

    ...
}
```

① called by Isis (which passes in all Isis properties)

② public API

Using this API makes `domainObject` available as `this` in the template.

For example, assuming an instance of the `Customer` class:

```
public class Customer {
    private String firstName;
    private String lastName;
    private Address address;
    private Address billingAddress;

    // getters and setters omitted
}
```

... that in turn has relationships to the `Address` class

```
public class Address {
    private int houseNumber;
    private String town;
    private String postalCode;

    // getters and setters omitted
}
```

then the following are valid expressions:

- `${this.firstName}`
- `${this.lastName != null? this.lastName : ''}`
- `${this.address.houseNumber}`

Object graph interpolation (using the lower-level API)

The service also offers a lower-level API which allows multiple objects to be made accessible from the context:

```
public class StringInterpolatorService {

    public static class Root {
        ...
        public Root(final Object context) {
            this._this = context;
        }
        public Object getThis() { return _this; }
        ...
    }

    // public API
    public String interpolate(Root root, String template) { ... }

    ...
}
```

The `Root` class can be extended as necessary.

For example, create a custom subclass of the `Root` class:

```
final class CustomRoot extends StringInterpolatorService.Root {
    private Customer customer;
    public CustomRoot(Object context, Customer customer) {
        super(context);
        this.customer = customer;
    }
    public Customer getCustomer() {
        return customer;
    }
}
```

The example above exposes the `customer` property. This can then be used in the template, eg:

```
@Test
public void simple() throws Exception {
    String interpolated = service.interpolate(
        new CustomRoot(null, customer), "${customer.firstName}");
    assertThat(interpolated, is("Fred"));
}
```

Strict Mode (applies to both APIs)

By default, any expression that cannot be parsed or would generate an exception (eg null pointer exception) is instead returned unchanged in the interpolated string.

The service also provides a "strict" mode, which is useful for testing expressions:

```
StringInterpolatorService service = new StringInterpolatorService().withStrict(true);
```

If enabled, then an exception is thrown instead.

Known issues

None known at this time.

Dependencies

In addition to Apache Isis, this module depends on:

- `ognl:ognl` (ASL v2.0 License)

Related Modules/Services

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/lib/stringinterpolator/impl -D
excludeTransitive=true
```

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

```
org.javassist:javassist:jar:3.19.0-GA
ognl:ognl:jar:3.0.8
```

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

- OGNL
- Javassist