Fixture Support Library

Table of Contents

This module (`incode-module-fixturesupport`) provides support for writing fixtures.

How to configure/use

Classpath

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

```
<dependency>
    <groupId>org.incode.module.fixturesupport</groupId>
    <artifactId>incode-module-fixturesupport-dom</artifactId>
</dependency>
```

Check for later releases by searching Maven Central Repo.

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

Utilities

DemoData

The `DemoData` interface, and supporting `DemoData.Util` utility class, is intended to allow type-safe datasets to be set up. It is defined as:

```
public interface DemoData<D extends Enum<D>, T> {
    T asDomainObject();
    T persistUsing(ServiceRegistry2 serviceRegistry);       ①
    T findUsing(ServiceRegistry2 serviceRegistry);
}
```

① From `ServiceRegistry` the class can either lookup the low-level `RepositoryService`, or can lookup a higher-level domain-specific service (eg `CustomerRepository`).

To use, assume we have a domain object such as:

```
public class DemoInvoice implements Comparable<DemoInvoice> {

    @lombok.Builder                                                 ①
    public DemoInvoice(
            int num,
            LocalDate dueBy,
            int numDays,
            String atPath) {
        this.num = num;
        this.dueBy = dueBy;
        this.numDays = numDays;
        this.atPath = atPath;
    }

    private int num;                                                ②
    private LocalDate dueBy;
    private int numDay;
    private String atPath;

    ...
}
```

① Lombok-generated builder

② corresponding fields (JDO annotations and Isis etc. not shown, for brevity)

We then define a corresponding "data" subclass as an enum, implementing `DemoData`. For example:

```
@lombok.AllArgsConstructor
@lombok.Getter
public enum DemoInvoiceData implements DemoData<DemoInvoiceData, DemoInvoice> {

    Invoice1(1, new LocalDate(2017,1,31), 30, "/"),                          ①
    Invoice2(2, new LocalDate(2017,1,20), 60, "/ITA"),
    Invoice3(3, new LocalDate(2017,1,25), 90, "/FRA"),
    ;

    private final int num;                                                   ②
    private final LocalDate dueBy;
    private final int numDay;
    private final String atPath;

    @Programmatic
    public DemoInvoice asDomainObject() {
        return DemoInvoice.builder()                                         ③
                    .num(num)
                    .dueBy(dueBy)
                    .numDays(numDay)
                    .atPath(atPath)
                    .build();

    @Programmatic
    public DemoInvoice persistUsing(ServiceRegistry2 serviceRegistry) {
        return Util.persist(this, serviceRegistry);                          ④
    }

    @Programmatic
    public DemoInvoice findUsing(ServiceRegistry2 serviceRegistry) {
        return Util.firstMatch(this, serviceRegistry);                       ⑤
    }
    ...
}
```

① the data sets to create

② mirror the fields in the domain object

③ using the `@Builder` provided by the domain object

④ delegates to `DemoData.Util` to create and persist an instance of the domain object

⑤ delegates to `DemoData.Util` to find an existing instance of the domain object

A fixture script can then be written by subclassing the supporting `DemoDataPersistAbstract` fixture script. We suggest this script is implemented as a nested static class,eg:

```
public enum DemoInvoiceData ... {
    public static class PersistScript
                extends DemoDataPersistAbstract<PersistScript, DemoInvoiceData,
DemoInvoice> {
        public PersistScript() {
            super(DemoInvoiceData.class);
        }
    }
}
```

The fixture script can now be used in the setup for tests, or used as within a larger composite fixture scripts:

```
final DemoInvoiceData.PersistScript fs = new DemoInvoiceData.PersistScript();
fixtureScripts.runFixtureScript(fs, null);
```

Optionally, the number of instances to create can be specified:

```
final DemoInvoiceData.PersistScript fs = new DemoInvoiceData.PersistScript().
setNumber(1);
fixtureScripts.runFixtureScript(fs, null);
```

Each data instance can also be used to find the corresponding domain object:

```
final DemoInvoice invoice1 = DemoInvoiceData.Invoice1.findUsing(serviceRegistry);
...
```

Known issues

None known at this time.

Dependencies

Maven can report modules dependencies using:

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

which, excluding Incode Platform and Apache Isis modules, returns no direct compile/runtime dependencies.

From the Incode Platform it uses:

- base library module