Entity Provider and Broker Usage

Information

This explains how to use the entity broker and entity provider system in Sakai

Usage of the Entity Broker

- EntityBroker gives you access to all entities in Sakai including the legacy ones. It also provides methods for searching for entities using things like properties and tags.
- Use the EntityBroker like any other Sakai component bean. You can inject it into any services and beans OR look it up manually using the Sakai ComponentManager.
    1. Inject it into a spring bean by creating a setter and adding the property to your bean (normal Spring)

```
<bean id="yourBean" class="org.place.YourClass">
      <property name="entityBroker" ref="org.sakaiproject.entitybroker.EntityBroker" />
</bean>
```

    2. Get it from the Sakai component manager using a call to the ComponentManager cover

```
import org.sakaiproject.component.cover.ComponentManager;
...
EntityBroker entityBroker = (EntityBroker) ComponentManager.get(EntityBroker.class);
```

- Add the needed maven dependendencies to the your maven files
    - Maven 1 users will need soemthing like this in the project.xml:

```
<dependency>
        <groupId>sakaiproject</groupId>
        <artifactId>sakai-entitybroker-api</artifactId>
        <version>${sakai.version}</version>
</dependency>
```

    - Maven 2 users will need something like this in the pom.xml:

```
<dependency>
    <groupId>org.sakaiproject.entitybroker</groupId>
    <artifactId>entitybroker-api</artifactId>
    <version>1.3.9</version>
    <scope>provided</scope>
</dependency>
```

- Now you can access the various methods on the EntityBroker like:

```
User user = (User) entityBroker.fetchEntity("/user/admin");
Site site = (Site) entityBroker.fetchEntity("/site/DG73G3GE33-3FJ3HR4RF-F3FJF3JF4H-F3R4FR4HH");
```

Defining EntityProviders

- Explains how to define your own entity providers

Helper code

DeveloperHelperService

- Makes it easier to work with Sakai in general and think of everything as entities

Access the DeveloperHelperService service

- You can use Spring Framework to inject the service or use the cover

1. Using Spring to get the service for your class (e.g. YourAppClass) *(recommended)*
    a. Add the DeveloperHelperService bean to the bean for YourAppClass

```
<bean id="org.sakaiproject.yourapp.logic.YourAppClass"
                class="org.sakaiproject.yourapp.logic.impl.YourAppClassImpl">
        <property name="developerHelperService"
                ref="org.sakaiproject.entitybroker.DeveloperHelperService" />
</bean>
```

    b. Add a variable and setter to YourAppClass to use the service in like so:

```
private DeveloperHelperService developerHelperService;
public void setDeveloperHelperService(DeveloperHelperService developerHelperService) {
        this.developerHelperService = developerHelperService;
}
```

2. Using the Component Manager to get the service
    - **Note:** This is not the recommended method, you should be using Spring to inject the service
    a. Use the CM cover to get the service

```
import org.sakaiproject.component.cover.ComponentManager;
import org.sakaiproject.entitybroker.DeveloperHelperService;
...
  private DeveloperHelperService developerHelperService;
...
    developerHelperService = (DeveloperHelperService) ComponentManager.get(DeveloperHelperService.
class);
```

BeanCollector

- Allows developers to look up beans by an interface type and have them injected without having to do manual lookups

ReloadableComponentProxy

- Allows for exposing a service from a webapp to the Sakai component manager/central Spring context