SDL Your Content Their Language

HowTo: Extending Content Broker Storage

An example of storing comments using the new extensible storage model

Table of contents

1 Introduction

For SDL Tridion 2011, Content Delivery has received an overhaul of the storage layer. This whitepaper explains how you can utilize this new layer to store your own proprietary content for use on your website using the example of storing user-generated comments on your published content. For this article, you will need to have knowledge of *Hibernate* and the *Java Persistence API* and we recommend you read up on those topics before reading this article.

1.1 Why a new data storage model?

Storing content in a way it can be retrieved and reused is the essential corner stone for any website; comments, product information or stock information are all types of content that go into making your website usable for your visitors. Technically, it has always been possible to include multiple data sources into one website but often this requires multiple Data Access Layers that all have the potential to go wrong. The new storage layer from SDL Tridion 2011's Content Delivery allowing seamless integration both your Content Broker and your proprietary storage in one layer making it easier to extend your data sources into multiple distinct storage environments all accessed through the same API.

2 New storage layer

The new storage layer is based upon the Java Persistence API (JPA) and its concrete implementation, Hibernate. JPA uses entities to represent data persisted in a relational database, similar to the concepts of Enterprise Java Beans, and these entities are accessible using the Java Persistence Query Language (JPQL). JPQL makes queries against entities stored in a relational database. Queries resemble SQL queries in syntax, but operate against entity objects rather than directly with database tables.

2.1 Storage Configuration

The storage configuration (cd_storage_conf.xml) is the replacement to the, now deprecated, Broker configuration (cd_broker_conf.xml). The storage configuration now allows you fine grain and flexible configuration of where each item type, or entity, can be persisted into storage. This is a move away from the old model of all content in either the file system or a database (or a combination of the two) but never two different databases or two different file locations. Such a mapping looks as follows:

```
<Item typeMapping="LinkInfo" cached="true" storageId="defaultFile"/>
```

From the configuration example you are able to see what storage is used for a given item type and whether the results are cached or not. Note that the example can be used at global level (to define default storage for a given item type) or at publication level (to define storage for a given item type and publication).

Storage itself is defined, in much the same way as in 2009; the only major difference is the definition of the file system storage (which is implicit in the 2009 configuration).

It is worth noting that for some item types you are now able to define different storage locations for different extensions. For instance, for Binary data you could split JPEG images to a file system and PDFs to a database. For example:

```
<Item typeMapping="Binary" itemExtension=".pdf" storageId="defaultdb"/>

<Item typeMapping="Binary" itemExtension=".jpeg" storageId="defaultFile"/>
```

JPA Data Access Objects

Underneath the covers of the Broker APIs and the storage configuration, JPA & Hibernate are used to persist entities to and from storage. The JPA Data Access Objects (or DAO) are the front line in this work and these define how such entities are to be operated on; persisting, removing, finding etc. are all defined in the DAO.

Underneath the DAO are both Entitles and Database tables;

Entities

Persistence entities are the code representations of database table(s) and entity instances correspond to rows in the table(s).

Database Tables

One or more database tables are the physical storage for the entities

3 How to store your own information

The Storage Layer is designed to be extensible and because of this, it is now possible to store other information using the same mechanism as regular Broker content. . To do this, you will need to define and implement the following:

- Database tables and their entity definition
- A DAO Bundle
- A custom DAO

3.1 Creating an Entity

Your entity effectively defines your data and typically represents a database table and therefore you must first define the database table you wish to use and create that in a database. Your entity would then be defined as follows:

```
package com.tridion.storage;

import javax.persistence.Entity;
```

```
import javax.persistence.Table;
import javax.persistence.Transient;
 @Entity
@Table(name="TABLE_NAME")
public class Comment extends BaseEntity {
    // constructors, fields, methods and custom logic
    @Override
    @Transient
    public Object getPK() {
        return internalId;
    }
 }
```

The class should extend com.tridion.storage.BaseEntity, should be in the package com.tridion.storage and should follow the JPA annotation specifications as the example shows. The getPK method, which should be Transient, defines the primary key and can be either simple or composite. In this entity class, you will need to continue to define the getters and setters that match the columns in your table. For example:

```
@Column(name="comment", nullable=false)
public String getComment() {
      return comment;
}

public void setComment(String comment) {
      this.comment = comment;
}
```

3.2 Creating a custom DAO

The custom DAO must implement a custom interface, which should be placed in the package *com.tridion.storage.dao*. It must extend the *JPABaseDAO* classes for accessing persistence storage. Your custom DAO would then look something like the following:

```
package com.tridion.storage.dao;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

import com.tridion.storage.dao.CommentDAO;

import com.tridion.storage.persistence.JPABaseDAO;


@Component("JPACommentDAOImpl")

@Scope("prototype")

public class JPACommentDAOImpl extends JPABaseDAO implements CommentDAO {

      public JPACommentDAOImpl(String storageId, EntityManagerFactory
      entityManagerFactory, String storageName) {

            super(storageId, entityManagerFactory, storageName);

      }

// class continues…
```

The implementation of that interface defines the actual methods that will store, retrieve, update and remove our entities to and from the persistence layer. Your class must be in a package under *com.tridion.storage.dao*, should extend *JPABaseDAO* and should implement your custom defined interface; the *JPABaseDAO* contains a set of methods that help you manage you entities to and from the persistence layer:

```
public class JPACommentDAOImpl extends JPABaseDAO {
```

And it should also have a constructor with three parameters to comply with the internal definition of the DAOs:

```
public JPACommentDAOImpl(String storageId, EntityManagerFactory
entityManagerFactory, String storageName) {
```

```
        super(storageId, entityManagerFactory, storageName);

    }
```

For each of the methods defined in your interface, you will need to implement the method (or its stub). For example, you could define a *getCommentsForPost* method and use JPQL to get a list of your comments.

```
public List<Comment> getCommentsForPost(int publicationId, int commentId)
throws StorageException {

        StringBuffer queryBuilder = new StringBuffer();

        queryBuilder.append("from BlogComment where blogId= :blogId and
        blogPublicationId= :publicationId");

        Map<String, Object> queryParams = new HashMap<String, Object>();

        queryParams.put("commentId", commentId);

        queryParams.put("publicationId", publicationId);

        return executeQueryListResult(queryBuilder.toString(),
        queryParams);

    }
```

Or to store a new comment, in which case an example method would be as follows:

```
public void storeComment(BlogComment blogComment) throws StorageException
{

        create(blogComment);

    }
```

3.3 Configuring a custom DAO Bundle

To be able to use the DAO and therefore the comments, you must configure Comments in the Tridion storage configuration. This configuration is the DAO bundle, is stored as a separate configuration file and looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<StorageDAOBundles>

   <StorageDAOBundle type="persistence">
```

```
        <StorageDAO typeMapping="Comment" class="
        com.tridion.storage.dao.JPACommentDAOImpl"/>

        </StorageDAOBundle>

    </StorageDAOBundles>
```

This bundle is an XML file and defines the *typeMapping* and the class implementing your DAO.

With this, your DAO is complete and you can now use it with Tridion Content Delivery.

4 Making it all work

To use your new DAO you will need to:

- Configure the Content Delivery storage
- Write an application to create, retrieve, update and remove content

4.1 Configuration

The storage configuration, in the file *cd_storage_conf.xml,* must be configured with custom storage bindings. To do this you must add your custom DAO bundle, which specifes what the custom typeMappings as well as where Content Delivery can find the definitions of the DAO, to the storage configuration within the *Storages* element:

```
<StorageBindings>
      <Bundle src="myCustomDAOBundle.xml"/>
</StorageBindings>
```

Then you can configure the type mappings for your *comment* type and indicate the storage, in this case a database defined in the storage configuration as demoDB.

```
<Item typeMapping="Comment" storageId="demoDB"/>
```

Add this type mapping to the same section as all other type mappings or as part of a publication's configuration

4.2 Accessing the data

To access the data you will simply need to call the methods defined in your implementation of your DAO. For example:

```
CommentDAO commentDAO = (CommentDAO)
StorageManagerFactory.getDefaultDAO("Comment");

List<Comment> comments =
commentDAO.getCommentsForPost(parsedPostURI.getPublicationId(),
parsedPostURI.getItemId());
```

5 Learning more

To learn more about the technologies used in the new storage layer, check the links below for more information:

JPA: http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

Hibernate: http://www.hibernate.org/

JPQL: http://en.wikipedia.org/wiki/Java_Persistence_Query_Language

5.1 About the Author

Julian Wraith

Principal Consultant, SDL WCM Solutions Division

Julian Wraith is a Technical Account Manager with SDL Tridion and has worked with the company for 8 years. He specializes in infrastructure related matters and is a Certified Consultant. Next to helping customers with their WCM needs, Julian is instrumental in many of the Knowledge Sharing activities at SDL Tridion. In the past, Julian has arranged the quarterly knowledge sharing between customers, partners, certified consultants and SDL Tridion. He was also a SDL Tridion MVP in 2010 and is the recipient of a SDL Tridion Community Builder Award for 2011. You can follow him on Twitter and via his personal blog.

6 About SDL

SDL is the leader in Global Information Management solutions, which provide increased business agility to enterprises by accelerating the delivery of high-quality multilingual content to global markets. The company's integrated Web Content Management, eCommerce, Structured Content and Language Technologies, combined with its Language Services drive down the cost of content creation, management, translation and publishing. SDL solutions increase conversion ratios and customer satisfaction through targeted information that reaches multiple audiences around the world through different channels.

Global industry leaders who rely on SDL include ABN-Amro, Bosch, Canon, CNH, FICO, Hewlett-Packard, KLM, Microsoft, NetApp, Philips, SAP, Sony and Virgin Atlantic. SDL has over 1000 enterprise customers, has deployed over 170,000 software licenses and provides access to on-demand portals for 10 million customers per month. It has a global infrastructure of more than 50 offices in 32 countries. For more information, visit www.sdl.com.