Template Loader 1.0 Component Specification

1. Design

The purpose of the `Template Loader 1.0` component is to load the details for the top-level template hierarchies from the persistent data store into a runtime memory on demand. The component provides an option to cache the template hierarchies which have been recently loaded from the persistent data store.

The clients must note that adding, deleting and updating the templates and template hierarchies in persistent data store is out of scope of this component.

This component declares the `TemplateHierarchy` and `Template` classes which are an object oriented representations of template hierarchy and template entities existing within the persistent data store.

The component utilizes the `Strategy` design pattern to provide the clients with facility to change the underlying persistence without affecting the behavior of the component. The current implementation of this strategy implements loading of template hierarchies from the `JDBC` compliant databases whose schema is initialized as provided by this component.

1.1 Design Patterns

`Strategy`

The `TemplateHierarchyPersistence` interface and `JDBCPersistence` class implement this pattern.

`Null Object`

The `NullCache` class implements this pattern.

`Composite`

The `TemplateHierarchy` class implements this pattern.

1.2 Industry Standards

`JDBC 2.0`

1.3 Required Algorithms

Initializing TemplateLoader with configuration properties provided by namespace

- If the `"caching"` configuration property is provided then:
  - If the `"timeout"` sub-property is provided then use the value of that property as cache timeout. Otherwise use the `SimpleCache.NO_TIMEOUT` as cache timeout.
  - If the `"size"` sub-property is provided then use the value of that property as cache maximum size. Otherwise use the `SimpleCache.NO_MAX_SIZE` as cache timeout.

    - Initialize the 'cache' instance field with instance of SimpleCache class constructed with selected timeout, max cache size and LRUCacheEvictionStrategy.
- Otherwise initialize the 'cache' instance field with instance of NullCache class.
- If the "persistence" property is not provided then throw ConfigurationException
- Otherwise:
    - Get the value of "class" sub-property
    - If the "config" sub-property is provided then instantiate the instance of TemplateHierarchyPersistence via Java Reflection API using the constructor accepting a single Property parameter.
    - Otherwise instantiate the instance of TemplateHierarchyPersistence via Java Reflection API using the constructor taking no parameters.
- If the "file_server" property is not provided then throw ConfigurationException
- Otherwise initialize the 'fileServerUri' instance field with the value of this property.
- Any exception which may be thrown when performing the steps above must be caught, wrapped with ConfigurationException and re-thrown.

Loading requested top-level template hierarchy by JDBCPersistence

1. Obtain a connection to database from DB Connection Factory.
2. Prepare SQL statement which is used to load the details for requested top-level template hierarchy. The following SQL statement template may be used for these purposes:

```
SELECT temp_hier_id, temp_hier_name

FROM temp_hier

WHERE temp_hier_name = ?

AND   temp_hier_id = parent_temp_hier_id
```

3. Execute the SQL statement and construct a TemplateHierarchy instance with obtained data.
4. Prepare SQL statement which is used to load the details for templates directly nested within specified template hierarchy. The following SQL statement template may be used for these purposes:

```
SELECT t.template_id, t.template_name, t.description,  t.uri,

       t.dest_filename

FROM templates t, temp_hier_mapping m

WHERE t.template_id = m.template_id

AND   m.temp_hier_id = ?
```

5. Execute the SQL statement and construct a Template instance for each record from the returned result set and add each template to template hierarchy.
6. Create a Map mapping the Long objects representing the IDs of template hierarchies to TemplateHierarchy objects. And put the loaded top-level template hierarchy to that map.

7. Create a list holding the `ID` of a top-level template hierarchy. This list will contain the IDs of template hierarchies which have been loaded from the database and for which the nested template hierarchies are to be loaded from the database.
8. Start the loop which must finish when the list created at step 7 becomes empty. Within the loop perform the following steps:
    1. Prepare SQL statement which is used to locate the list of template hierarchies directly nested within the template hierarchies referenced by `IDs` contained within the list created at step 7 . The following SQL statement template may be used for these purposes:

```
SELECT parent_temp_hier_id, temp_hier_id, temp_hier_name

FROM temp_hier

WHERE parent_temp_hier_id IN (?, ?, ...)

ORDER BY parent_temp_hier_id
```

    2. Execute the SQL statement and construct a `TemplateHierarchy` instance for each record from the returned result set, load the templates for each hierarchy and add each template hierarchy to template hierarchy corresponding to a parent `ID` (located by `ID` in map created at step 6).
    3. Initialize the list created at step 7 with `IDs` of template hierarchies just loaded from the database.

1.4 Component Class Overview

Package `com.topcoder.buildutility.template`

`TemplateLoader`

This is a main class of the component which must be used for locating and loading the top-level template hierarchies. The clients must use this class to obtain the template hierarchies loaded from the persistent data store. The `TemplateLoader` class provides an ability to specify whether the template hierarchies loaded from the persistent data store on demand must be cached or not.

The instances of this class may be initialized with parameters passed at run-time or using the configuration properties provided by the specified configuration namespace.

`TemplateHierarchy`

This class is an object representation of the template hierarchies which are used as groups of the template hierarchies and templates. This class does not implement any complex logic other than storing the values of the attributes of the template hierarchy, returning them via simple accessor methods and maintaining the lists of template hierarchies and templates directly nested within template hierarchy.

`Template`

This class is an object representation of the templates which are grouped into the template hierarchies. This class does not implement any complex logic other than storing the values of the attributes of the template and returning them via simple accessor methods.

`NullCache`

This class is a "dummy" implementation of `Cache` interface which is used by the `TemplateLoader` if it is instructed not to use any caching of template hierarchies loaded from persistent data store into a runtime memory. This implementation does nothing and always behaves like an empty cache thus causing the TemplateLoader always to load the template hierarchies from the persistent data store. The introduction of this class results in a simpler implementation of `TemplateLoader` class which is not required to check whether the caching is used or not when loading the template hierarchies.

`TemplateHierarchyPersistence`

This is an interface specifying the contract for the pluggable persistent data store which is used by the `TemplateLoader` to load the requested template hierarchies into runtime memory.

The implementation classes are required to provide a public constructor accepting a single `Property` argument if the implementation class is specified by the `"persistence.class"` configuration property and the `"persistence.config"` configuration property is also provided. The implementation classes are required to provide a public constructor taking no arguments if the implementation class is specified by the "persistence.class" configuration property and the `"persistence.config"` configuration property is not provided.

Package `com.topcoder.buildutility.template.persistence`

`JDBCPersistence`

An implementation of `TemplateHierarchyPersistence` interface which may be used to load the top-level template hierarchies from the JDBC-compliant database with schema initialized as specified by the Component Specification.

This class utilizes the `DB Connection Factory` component to obtain the connections to the target database. Therefore a name of such a pre-configured connection must be provided. If such a name is omitted then a default pre-configured connections is used.

This class provides a set of constructors which could be used to instantiate the instances of this class via `Java Reflection API` as specified by the contract for `TemplateHierarchyPersistence` interface. Also a name of an optional configuration property which must provide the name of pre-configured DB connection is provided by this class.

1.5 Component Exception Definitions

This component defines the following custom exceptions:

`com.topcoder.buildutility.template.ConfigurationException`

This is a checked exception to be thrown by `TemplateLoader` to indicate that the unexpected error occurs while reading the configuration properties from the specified configuration namespace and using them for initializing the state of the `TemplateLoader`. This exception may be (but not limited to) caused by absence of any of required properties, inability to locate the specified class implementing the `TemplateHierarchyPersistence` interface, negative values of cache timeout and maximum size, etc.

com.topcoder.buildutility.template.DuplicateObjectException

This is an unchecked exception to be thrown by the `TemplateHierarchy` to indicate that an attempt to add an object (template/template hierarchy) with existing ID is made. This exception will provide an ID of duplicate object.

com.topcoder.buildutility.template.UnknownTemplateHierarchyException

This is an unchecked exception to be thrown to indicate that the requested top-level template hierarchy could not be located (does not exist). Actually, this exception is thrown by the `TemplateHierarchyPersistence` and is propagated to the clients by the TemplateLoader.

com.topcoder.buildutility.template.PersistenceException

This is a checked exception to be thrown by the implementations of `TemplateHierarchyPersistence` implementations to indicate that an unrecoverable error preventing the loading of the details for the requested template hierarchy has occurred. Actually, this exception is thrown by the `TemplateHierarchyPersistence` and is propagated to the clients by the `TemplateLoader`.

The following system exceptions are declared to be thrown by the classes provided with this component:

java.lang.NullPointerException

This exception is thrown when almost any of the specified parameters is `null`.

java.lang.IllegalArgumentException

This exception is thrown when almost any of the specified String parameters is an empty String or the specified caching parameters are invalid. This exception is also thrown by TemplateHierarchy.addNestedHierarchy() when if the added template hierarchy is a top-level hierarchy or this hierarchy is not a parent of added hierarchy, or the added hierarchy causes a loop.

java.io.FileNotFoundException

This exception is thrown from `Template.getData()` method if the file providing the content of template could not be located.

java.io. IllegalStateException

This exception is thrown by Template.getData() method if the method is called before 'fileServerUri' is initialized.

1.6 Thread Safety

The component is thread safe.

Package `com.topcoder.buildutility.template`

`TemplateLoader`

This class is thread safe. The private state of `TemplateLoader` is never changed after instantiation and the only method affecting the state of the internal cache is synchronized.

`TemplateHierarchy`

This class is thread safe. The mutator methods affecting the state of lists of templates and template hierarchies are synchronized. The accessor methods accessing the `'templates'` and `'childHierarchies'` instance variables are synchronized. No need to synchronize accessor methods for `'id'` and `'name'` instance variables since these variables are not changed after instantiation.

`Template`

This class is thread safe. The private state of the instances of this class is never changed after instantiation.

`NullCache`

This class is thread safe. It does not maintain any private state.

Package `com.topcoder.buildutility.template.persistence`

`JDBCPersistence`

This class is thread safe. The private state is never changed after the instantiation.

2. Environment Requirements

2.1 Environment

- At minimum, Java1.4 is required for compilation and executing test cases.
- Java 1.2 or higher can be used for Basic and Log4j logging implementations.
- Java 1.4 or higher must be used for Java 1.4 built in logging.

2.2 TopCoder Software Components

- `Base Exception 1.0`

The custom exceptions defined by `Template Loader` component derive from the base exception classes declared by this component.

- `Config Manager 2.1.3`

The `TemplateLoader` class uses this component to read the configuration properties and initialize its state.

- `DB Connection Factory 1.0`

The `JDBCPersistence` uses this component to obtain the connections to target database.

- `Simple Cache 1.1`

  The `TemplateLoader` class uses this component to cache the template hierarchies loaded on demand from the persistent data store.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

`com.topcoder.buildutility.template`

`com.topcoder.buildutility.template.persistence`

3.2 Configuration Parameters

Parameter	Description	Values
file_server	A required property providing the URI referencing the root of a file server which the template URIs are resolved relatively to.	C:\TopCoder
caching	An optional container property providing the parameters to be used to configure the cache to be used by a TemplateLoader instance. If such a property is provided by the configuration namespace then the TemplateLoader instance will cache the template hierarchies loaded on demand. If such a property is missing then the TemplateLoader will not use any caching.	N/A
caching.timeout	An optional property which must be nested within "caching" property. This property (if present) must provide a positive timeout value to be used for configuring the cache used by the TemplateLoader instance.	10000
caching.size	An optional property which must be nested within "caching" property. This property (if present) must provide a positive value specifying the maximum size of the cache to be used for configuring the cache used by the TemplateLoader instance.	100
persistence	A required container property providing the parameters necessary to configure the TemplateHierarchyPersistence to be used by the TemplateLoader instance to load the requested template hierarchies from the persistent data store.	N/A
persistence.class	A required property which must be nested within the "persistence" property. This property must provide the fully-qualified name of class implementing the TemplateHierarchyPersistence interface which must be used by the TemplateLoader.	com.topcoder.buildutility.template.persistence.JDBCPersistence
persistence.config	An optional container property which must be nested within the "persistence" property. This property (if present) must provide the configuration parameters necessary to configure the instance of class implementing the TemplateHierarchyPersistence	N/A
persistence.config.db_connection	An optional property which (if present) must provide a name of pre-configured DB connection to be used by new instance of JDBCPersistence. If such a property is missing then a default DB connection will be used	InformixDB

3.3 Dependencies Configuration

No special settings are required to configure the `Base Exception` and `Simple Cache` components. Consult the documentation for `Config Manager` and `DB Connection Factory` components to learn the steps necessary to configure these components.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

1. Create the `TemplateLoader` instance.

   There are two possible usage scenarios for initializing this component:

   1. Using the configuration properties provided by the specified configuration namespace.

      In this case the following steps should be performed:

      - The configuration file must be created.
      - The configuration file must be loaded under specified namespace to `ConfigManager`.
      - The `TemplateLoader` must be instantiated using the constructor accepting a single `String` argument.

   2. Instantiate the `TemplateLoader` using the runtime parameters. In this case the following steps should be performed:

      - Create the instance of `TemplateHierarchyPersistence`.
      - Determine whether the caching must be used and which parameters must be used to configure the caching strategy.
      - Instantiate the `TemplateLoader` class using any of the constructors accepting the `TemplateHierarchyPersistence` argument.

2. Load the details for target template hierarchy.
3. Query the details for loaded target template hierarchy.

4.3 Demo

4.3.1 The general usage of this component.

```
// we could create TemplateLoader instance with a namespace
TemplateLoader loader = new TemplateLoader(UNITTEST_NAMESPACE);

// we can also create TemplateLoader instance using the specified
```

```
// TemplateHierarchyPersistence and the file server uri, no caching
loader = new TemplateLoader(new MockPersistence(), "test_files/");

// we can also create TemplateLoader instance using the specified cache
loader = new TemplateLoader(new MockPersistence(), "test_files/", 1000, 10);

// new we can load the desired TemplateHierarchy
TemplateHierarchy root = loader.loadTemplateHierarchy("foo");
assertEquals("the name is incorrect", "foo", root.getName());

// get a Template of root node
Template template = root.getTemplates()[0];

// get an InputStream containg the data of the template
InputStream input = template.getData();
input.close();
```

4.3.2 The general usage of Template and TemplateHierarchy.

```
// create a top-level TemplateHierarchy
TemplateHierarchy root = new TemplateHierarchy(1, "root", 1);

// create another TemplateHierarchy
TemplateHierarchy child = new TemplateHierarchy(2, "child", 1);

// get the id and parent id
assertEquals("id is incorrect", 1, root.getId());
assertEquals("parent id is incorrect", root.getId(), child.getParentId());

// add the child node to root node
root.addNestedHierarchy(child);

// see whether the root has children
assertTrue("true should be returned", root.hasChildren());

// get all children of root
TemplateHierarchy[] children = root.getChildren();
assertEquals("the length is incorrect", 1, children.length);

// create a Template
Template template = new Template(1, "name", "des", "file", "uri");

// get the attributes
assertEquals("id is incorrect", 1, template.getId());
assertEquals("name is incorrect", "name", template.getName());
assertEquals("description is incorrect", "des", template.getDescription());
assertEquals("file name is incorrect", "file", template.getFileName());
assertEquals("uri is incorrect", "uri", template.getUri());

// add the template to child node
child.addTemplate(template);

// see whether the root has templates
assertTrue("true should be returned", child.hasTemplates());

// get all templates of child
Template[] templates = child.getTemplates();
assertEquals("the length is incorrect", 1, templates.length);
```

4.3.3 The general usage of JDBCPersistence

```
Property property = configManager.getPropertyObject(
   UNITTEST_NAMESPACE, "persistence.config");

// create a JDBCPersistence using the given configuration property
TemplateHierarchyPersistence persistence = new JDBCPersistence(property);

// create a JDBCPersistence using the given DB factory and connection name
persistence = new JDBCPersistence(
   "InformixConnection", new DBConnectionFactoryImpl(UNITTEST_NAMESPACE));

// we can also create a JDBCPersistence instance which use the defualte DB
// factory namespace and connection name
persistence = new JDBCPersistence();
```

5. Future Enhancements

The API provided by the component could be extended to provide a functionality supporting the full lifetime cycle of template hierarchies (creation, deletion, modification).