DB Connection Factory 1.1 Component Specification

Additions will be in **RED** and updates in **BLUE**.

1. Design

The DB Connection Factory 1.1 component provides a simple but flexible framework allowing the clients to obtain the connections to a SQL database without providing any implementation details. The clients are no longer aware of an underlying implementation used to obtain the connections to the databases.
The component provides the clients with the configurable factory which allows obtaining either a default connection to a database or a configured connection referenced by name. Also the users can obtain connections authenticated with credentials in the configuration or specified through programming API. Such a factory may be configured both through programming API, configuration files(deprecated) or configuration objects.
The logic for obtaining the connections to a database is concentrated by a single ConnectionProducer interface. The implementations of this interface are responsible for producing the SQL connections which are returned by the factory to the clients. The factory manages the mapping from names to ConnectionProducer instances.
This design provides the implementations of ConnectionProducer interface which allows obtaining the connections either through a DataSource or by dynamically creating them via a JDBC URL.

1.1 Design Patterns

Factory Method: The createConnection methods in ConnectionProducer interface are factory methods responsible for obtaining the connections. From the users' view, those createConnection methods in DBConnectionFactory can also be deemed as factory methods.

Strategy: The ConnectionProducer instances are used as strategies in the DBConnectionFactoryImpl context.

1.2 Industry Standards

JDBC 1.0
JDBC 2.0
JNDI 1.0

1.3 Required Algorithms

No complex algorithms are used in the component, in this section we will just show where to find how to load configuration for DBConnectionFactoryImpl either from configuration files or from configuration objects.

1.3.1 Load DBConnectionFactoryImpl Configuration From Configuration Files

This process has been completed in version 1.0, see the real code of DBConnectionFactoryImpl(String namespace) for the details. Note that in version 1.1 to load the configuration in such a way is deprecated.

1.3.2 Load DBConnectionFactoryImpl Configuration From Configuration Objects

The component will rely on the structure of the ConfigurationObject to be consistent with that of the root configuration object produced by using the Configuration Persistence 1.0

component to load a Configuration Manager configuration file.
For the concrete steps to load the configuration, see the method document for DBConnectionFactoryImpl(configurationObject:ConfigurationObject, namespace:String) method.

1.4 Component Class Overview

DBConnectionFactory

This is an interface defining the factory API that enables the clients to obtain the connections to the specific databases. Such a factory may produce different Connection objects providing the connections to the different databases. The configured connections are mapped to the String names uniquely distinguishing the connection among other connections. The client may obtain a connection by specifying such a name or requesting a connection which is set as default for the factory. Also, the client can obtain connections authenticated with credentials either in the configuration or provided dynamically.

DBConnectionFactoryImpl

A default implementation of DBConnectionFactory interface.

This class implements a simple mapping from the names to the ConnectionProducers which are used to create the connections to the databases. Any of registered producers can be set to be used to create the connections by default.

This class provides a constructors allowing to instantiate and initialize the factory instance either programmatically, from the parameters specified by the configuration file(deprecated) or from the parameters specified by the configuration objects. A non-argument constructor may be used to create the empty factory which may be populated at runtime through appropriate methods. The constructor accepting a single String argument may be used to instantiate and initialize the factory from the parameters provided by a specified configuration namespace and this constructor becomes deprecated in version 1.1. Two extra constructors accepting a ConfigurationObject argument are provided to construct the factory from the parameters provided by configuration objects, either from the default namespace or from the namespace specified. Also, the client can obtain connections authenticated with credentials either in the configuration or provided dynamically.

ConnectionProducer

This is an interface specifying the API for the producers of the connections to the specific databases. Such a producer is responsible for creating the connections to a specific database.

As of DB Connection Factory 1.1 the ConnectionProducers are used by a DBConnectionFactoryImpl to configure and create the connections to the databases. The implementations which are expected to be used to configure the DBConnectionFactoryImpl through configuration file are required to provide a public constructor accepting a Property instance providing the configuration parameters to be used to initialize the producer. That constructor is deprecated in version 1.1. The implementations which are expected to be used to configure the DBConnectionFactoryImpl through configuration objects are required to provide a public constructor accepting a ConfigurationObject instance providing the configuration parameters to be used to initialize the producer.

JDBCConnectionProducer

A ConnectionProducer obtaining the connections to a database by the means specified by JDBC 1.0 specification, namely a DriverManager provided with a JDBC-compliant URL string and a connection properties is used to obtain a connection to a database.

This class provides a set of convenient constructors allowing to instantiate and initialize the JDBCConnectionProducer either programmatically, by providing the properties from configuration file(deprecated) or by providing the parameters from configuration objects.

This class requires only a JDBC-compliant URL string to be provided. Such an URL is used to obtain the connection to a database from a DriverManager. Optionally the client may specify the JDBC driver specific parameters to be passed to the JDBC driver matching the specified JDBC URL. Those parameters may include the username/password pair to be used to obtain the connections to a database.

The client can obtain connections authenticated with credentials either in the configuration or provided dynamically.

DataSourceConnectionProducer

This is an implementation of ConnectionProducer obtaining the connections from a DataSource.

In order to operate properly this class requires the DataSource to be provided. Optionally a username/password pair may be provided to specify a user on whose behalf the connection is to be made.

This is a base class for the connection producers obtaining the connections to a database from a DataSource, optionally providing the username and password. The subclasses are responsible only for proper initialization of a DataSource, username and password.

This class provides a set of constructors allowing to instantiate and initialize the instance either programmatically, through the configuration properties(deprecated) or by configuration objects. The public constructors are intended to be called both by the clients and the subclasses while the protected constructors are intended to be called by subclasses only.

The client can obtain connections authenticated with credentials either in the configuration or provided dynamically.

JNDIConnectionProducer

This is a DataSourceConnectionProducer obtaining the DataSource from a JNDI context using the provided JNDI name.

This class provides only a set of convenient constructors allowing to instantiate and initialize the JNDIConnectionProducer either programmatically, by providing the properties from configuration file(deprecated) or by providing the parameters from configuration objects.

This class requires only a JNDI name to be provided. Such a name is used to locate the DataSource within the JNDI context. Optionally the client may specify the environment properties to be used to initialize the JNDI context along with the username/password pair to be used to obtain the connections from a DataSource.

ReflectingConnectionProducer

This is a DataSourceConnectionProducer instantiating the DataSource through Java Reflection API being provided with the fully-qualified name of class implementing the DataSource interface.

The main purpose of this class is to provide a facility to obtain a DataSource without requiring the JNDI context (for example, for testing purposes). This class is more suitable to be instantiated and initialized from the properties provided by configuration file.

This class provides only a set of convenient constructors allowing to instantiate and initialize the ReflectingConnectionProducer either programmatically, by providing the properties from configuration file(deprecated) or by providing the parameters from configuration objects.

This class requires only a fully-qualified name of class implementing the DataSource interface to be provided. Such a name is used to create the DataSource using Java Reflection API. Optionally the client may specify the username/password pair to be used to obtain the connections from a DataSource.

The class expects the DataSource implementation classes to provide a public non-argument constructor in order to be instantiated through Java Reflection API.

1.5 Component Exception Definitions

com.topcoder.db.connectionfactory.ConfigurationException

An exception to be thrown by DBConnectionFactoryImpl being initialized from the parameters provided by configuration files or by configuration objects if any error occurs while reading the configuration parameters, instantiating and initializing the ConnectionProducers. Usually this exception will wrap the original exception which caused the operation to fail.

This exception is thrown from DBConnectionFactoryImpl(String) constructor, DBConnectionFactoryImpl(ConfigurationObject) constructor or DBConnectionFactoryImpl(ConfigurationObject, String) constructor.

com.topcoder.db.connectionfactory.DBConnectionException

This is an exception to be thrown by DBConnectionFactory or ConnectionProducer methods if a requested connection can not be created for some reason. Usually this exception will wrap the original, implementation-specific exception which caused the operation to fail.

This class is a base class for all implementation-specific exceptions which are thrown if a requested connection can not be created for whatever reason.

This exception is thrown from createConnection methods declared by DBConnectionFactory and ConnectionProducer interfaces and their implementations.

com.topcoder.db.connectionfactory.NoDefaultConnectionException

An exception to be thrown by if the default connection is not configured within the factory.

This exception is thrown from createConnection() method or createConnection(

username:String, password:String) method from DBConnectionFactory interface.

com.topcoder.db.connectionfactory.UnknownConnectionException

An exception to be thrown if the connection corresponding to specified name is not configured within the factory.

This exception is thrown from createConnection(String) method or createConnection(String, String, String) method declared by DBConnectionFactory interface and implemented by DBConnectionFactoryImpl class. Also the setDefault(String) method, DBConnectionFactoryImpl(String) constructor, DBConnectionFactoryImpl(ConfigurationObject) constructor and DBConnectionFactoryImpl(ConfigurationObject, String) constructor may throw such an exception.

com.topcoder.configuration.ConfigurationAccessException

This exception is thrown in the constructors of Connection Producer classes with a single ConfigurationObject parameter if any error occurs while reading the configuration.

javax.naming.NamingException

This exception is thrown in JNDIConnectionProducer's constructors if a naming error occurs while getting the initial JNDI context and performing a lookup for requested data source.

java.lang.NullPointerException

~~This exception is thrown when a parameter provided to a method is NULL and the method prohibits the NULL parameters to be passed.~~ This exception is discarded in version 1.1.

java.lang.IllegalArgumentException

This exception is thrown by producer classes or factory implementation if specified argument is invalid. For example, an empty name passed to factory is empty or the Property/ConfigurationObject passed to constructors of ConnectionProducer implementations does not contain the required property. All the NullPointerExceptions thrown in version 1.0 are replaced by this exception.

java.lang.ClassNotFoundException

java.lang.SecurityException

java.lang.IllegalAccessException

java.lang.InstantiationException

java.lang.ClassCastException

These exceptions are thrown from constructors of ReflectingConnectionProducer when it attempts to create the DataSource instance using Java Reflection API. For example, if a specified class could not be located, there is no permission to create the instances of that class, the class does not provide public constructor, the class does not implement the DataSource interface or is an interface or an abstract class.

java. lang. LinkageError
java. lang. ExceptionInInitializerError

1) These error happen when loading the jdbc driver from JDBCConnectionProducer(ConfigurationObject) or JDBCConnectionProducer(Property)
2) The ReflectingConnectionProducer alos throw it when using reflection to create the needed DataSource
3) The constructor of DBConnectionFactoryImpl which use reflection to create producers wrap those error into ConfigurationException

1.6 Thread Safety

This component is thread safe. The implementations of DBConnectionFactory and ConnectionProducer interfaces are required to be thread safe. For the current implementations, synchronization is performed where necessary to keep the class thread safe, see the class document for the details that how each class achieves this requirement.
A clarification should be made that although we assure there will be no thread safety issues when calling the methods defined in this component at the same time from different threads, some extra attentions should be paid for the correct usage of the component. First, the ConfigurationObject instance used in the construction may not be thread safe, if the contents of that object are modified during the configuration from other threads, the constructed factories and producers may have unexpected configuration. Second, if the users want to traverse the producers names in a thread safe way using the iterator returned by listConnectionProducerNames method in DBConnectionFactoryImpl, they must perform the synchronization on the factory instance during the whole traverse.

2. Environment Requirements

2.1 Environment

Development language: Java 1.4
Compile target: Java 1.4, Java 5.0
Note that Java 1.3 is no longer need to be supported.

QA Environment:
Solaris 9, 10
RedHat Enterprise Linux 4
Windows XP, 2003, Vista

2.2 TopCoder Software Components

Base Exception 1.0:

The custom exceptions defined by this component derive from the base exceptions provided by that component

Configuration Manager 2.1.5(deprecated):

This component is used to read the configuration from specified configuration file. In version 1.1 this usage is deprecated.

Configuration API 1.0:

This component is used to read the configuration from specified configuration object.

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

JAXP 1.3: Not used directly. Configuration Manager 2.1.5 makes use of it.

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.db.connectionfactory
com.topcoder.db.connectionfactory.producers

3.2 Configuration Parameters

The formats used for configuration object and configuration file(deprecated) are the same logically. Configuration Persistence 1.0 component can just transform the configuration file to the root configuration object used. The configuration for the connection factory will be put into a certain namespace of the root object.

DBConnectionFactoryImpl:

Parameter	**Description**	**Values**
connections	This property is a container for nested properties providing the details for configured connections. Required.	N/A
connections.default	This property provides a name of the configured connection to be used by default. Optional.	Should match the name of any other property directly nested within ‘connections’ property. Non-null, non-empty String.
connections.*<connection name>*	This property is a container for nested properties providing the configuration parameters for a particular connection. The name of this property represents the name of configured connection and should be unique across entire configuration. Optional.	N/A
connections.*<connection name>*.producer	This property provides a fully-qualified name of class implementing ConnectionProducer interface. Required.	Non-null, non-empty String.
connections.*<connection*	This property is a container for	N/A

name>.parameters	the nested properties providing the parameters to be passed to a ConnectionProducer. Consult the documentation for configured ConnectionProducer to determine the list of parameters necessary to configure the particular producer. Required.	

JDBCConnectionProducer's parameters:

Parameter	Description	Values
jdbc_url	The JDBC-compliant URL to be provided to a DriverManager to obtain a connection to a database. Required.	Non-null, non-empty String.
jdbc_driver	The name of a configuration property providing the fully-qualified name of a JDBC driver class to be loaded to obtain the connections to a database. Optional.	Non-null, non-empty String.
<connection properties>	The configuration parameters to be provided to a DriverManager to obtain a connection to a database. Optional.	Each value will be a non-null String.

DataSourceConnectionProducer's parameters:

Parameter	Description	Values
username	The username to be used to obtain a connection to a database. Optional.	Non-null String.
password	The password to be used to obtain a connection to a database. Optional.	Non-null String.

JNDIConnectionProducer's parameters(Besides those defined in DataSourceConnectionProducer):

Parameter	Description	Values
jndi_name	The JNDI name to be used to locate the DataSource to be used to obtain the connections to a database. Required.	Non-null, non-empty String.
<environment properties>	The environment properties for the context. Optional.	Each value will be a non-null String.

ReflectingConnectionProducer’s parameters(Besides those defined in DataSourceConnectionProducer):

Parameter	**Description**	**Values**
datasource_class	The fully-qualified name of a class implementing DataSource interface to be used by this producer to obtain a connection to a database. Required.	Non-null, non-empty String.

3.3 Dependencies Configuration

Follow the configuration for Configuration Manager 2.1.5 component.(Actually the usage is deprecated).

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

The most likely usage scenario for this component could look as follows:

1) For DBConnectionFactory initialized from the parameters specified by configuration object:

   Load the configuration object from configuration files using Configuration Persistence 1.0 component.

   Instantiate the DBConnectionFactoryImpl providing the configuration object.

   Use a factory to produce a connection to a database.

2) For DBConnectionFactory initialized from the parameters specified by configuration file(deprecated):

   Create a configuration file providing the parameters for connection factory.

   Instantiate the DBConnectionFactoryImpl providing the configuration namespace.

   Use a factory to produce a connection to a database.

3) For DBConnectionFactory initialized programmatically:

   Instantiate an empty DBConnectionFactoryImpl.

   Instantiate a ConnectionProducer to be registered to factory.

   Register the newly created producer to a factory under unique name.

   Register as many producers as needed.

   Specify the producer to be used by default.

   Use a factory to produce a connection to databases.

Optionally:

i. Remove the producers which are no longer needed.
ii. Remove all producers from factory.
iii. Get the name of default producer.
iv. Iterate over the names of registered producers.
v. Get the required producer by name.

4.3 Demo

Suppose the following configuration file has been provided:

```
<CMConfig>
  <Config name="com.topcoder.db.connectionfactory.DBConnectionFactoryImpl">
    <Property name="connections">
      <!--
        The "default" property refers to a configured connection.
      -->
      <Property name="default">
        <Value>MySqlJDBCConnection</Value>
      </Property>

      <!--
        The following property configures the ConnectionProducer obtaining the
Connections
        from a JDBC URL
      -->
      <Property name="MySqlJDBCConnection">
        <Property name="producer">

<Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer</V
alue>
        </Property>
        <Property name="parameters">
          <Property name="jdbc_driver">
            <Value>com.mysql.jdbc.Driver</Value>
          </Property>
          <Property name="jdbc_url">
            <Value>jdbc:mysql://localhost:3306/tcs</Value>
            </Property>
            <Property name="user">
                <Value>root</Value>
            </Property>
            <Property name="password">
                <Value></Value>
            </Property>
        </Property>
    </Property>


    <!--
        The following property configures the ConnectionProducer obtaining the
Connections
        from a JDBC URL. The JDBC driver is customized being provided driver-
specific properties.
    -->
<!--
    <Property name="CustomizedOracleJDBCConnection">
        <Property name="producer">

<Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer</V
alue>
        </Property>
        <Property name="parameters">
            <Property name="jdbc_driver">
                <Value>oracle.jdbc.driver.OracleDriver</Value>
            </Property>
            <Property name="jdbc_url">
                <Value>java:oracle:thin:@localhost:1521:TEST</Value>
            </Property>
            <Property name="user">
                <Value>scott</Value>
            </Property>
            <Property name="password">
                <Value>tiger</Value>
            </Property>
            <Property name="defaultRowPrefetch">
```

```
                <Value>15</Value>
            </Property>
            <Property name="defaultExecuteBatch">
                <Value>30</Value>
            </Property>
            <Property name="processEscapes">
                <Value>false</Value>
            </Property>
        </Property>
    </Property>
-->
    <!--
        The following property configures the ConnectionProducer obtaining the Connections from a DataSource
        object located through JNDI. This producer is provided only with a JNDI name to be used to lookup for the
        DataSource. Therefore a default JNDI context is used to locate the DataSource.
    -->
    <Property name="DefaultJNDI">
        <Property name="producer">
<Value>com.topcoder.db.connectionfactory.producers.JNDIConnectionProducer</Value>
        </Property>
        <Property name="parameters">
            <Property name="jndi_name">
                <Value>java:comp/env/jdbc/tcs</Value>
            </Property>
        </Property>
    </Property>

    <!--
         The following property configures a custom ReflectingConnectionProducer.
    -->
    <Property name="CustomReflectingConnectionProducer">
         <Property name="producer">
```

```
<Value>com.topcoder.db.connectionfactory.producers.ReflectingConnectionProduce
r</Value>
            </Property>
            <Property name="parameters">
              <Property name="datasource_class">
                <Value>customPackage.customClass</Value>
              </Property>
            </Property>
          </Property>
          <!--
            The following property configures the ConnectionProducer obtaining the
Connections from a DataSource
            object located through JNDI. This producer is provided with a JNDI name to
be used to lookup for the
            DataSource along with a set of initial parameters to be used to configure the
JNDI context to be used to
            locate the requested DataSource. The names of such initial parameters
represent the names of context
            parameters specified by the JNDI specification having the '.' character
replaced with '_' character. This is
            caused by a restriction applied by Configuration Manager for the property
names.
          -->
<!--
          <Property name="CustomJNDI">
            <Property name="producer">

<Value>com.topcoder.db.connectionfactory.producers.JNDIConnectionProducer</Va
lue>
            </Property>
            <Property name="parameters">
              <Property name="jndi_name">
                <Value>java:comp/env/jdbc/tcs</Value>
              </Property>
              <Property name="java_naming_factory_initial">
                <Value>foo.bar.ContextFactory</Value>
              </Property>
              <Property name="java_naming_provider_url">
```

```
                    <Value>protocol://localhost:port</Value>
                </Property>
            </Property>
        </Property>
-->
    </Property>
</Config>
</CMConfig>
```

// Add an entry for the above file to com/topcoder/util/config/ConfigManager.properties file:

com.topcoder.db.connectionfactory = <a path to above file>

// Also add an entry for the above file to com/topcoder/configuration/persistence/ConfigFileManager.properties file:

root_namespace= <a path to above file>

The demo program is as follows:

// Instantiate the DBConnectionFactoryImpl providing the configuration namespace. This usage is deprecated in version 1.1.

```
DBConnectionFactory factory = new DBConnectionFactoryImpl(
                    “com.topcoder.db.connectionfactory.DBConnectionFactoryImpl”);


 // In version 1.1 the common usage is to create the factory based on configuration object.
ConfigurationFileManager manager = new ConfigurationFileManager("test_files" + File.separator
                     + "demopreLoad.properties");
ConfigurationObject rootObject = manager.getConfiguration("demo");

// Create DBConnectionFactoryImpl using DBConnectionFactoryImpl.DEFAULT_NAMESPACE
factory = new DBConnectionFactoryImpl(rootObject);
                // Also we can specify the namespace manually to load configuration from the object.
                factory = new DBConnectionFactoryImpl(rootObject,
                   "com.topcoder.db.connectionfactory.DBConnectionFactoryImpl");

                ////////////////////////////////////////////////////////////////////////
                /// Second part: demonstration of the operations                      ///
                ///////////////////////////////////////////////////////////////////////
                // Obtain a default connection(It will obtain the default connection from
                // MySqlJDBCConnection since that’s the default producer)
                Connection con = factory.createConnection();
```

```
// Obtain a named connection
con = factory.createConnection("DefaultJNDI");
// Obtain a connection from the default(MySqlJDBCConnection) producer with user-provided
// credentials
con = factory.createConnection(username, password);
// Obtain a named connection with user-provided credentials
con = factory.createConnection("DefaultJNDI", "admin", "welcome");

// Create ReflectingConnectionProducer instance
ReflectingConnectionProducer pro = new ReflectingConnectionProducer(
   "com.topcoder.db.connectionfactory.producers.DataSourceImpl", "test_user", "test_pswd");

// Get the default connection from producer directly.
con = pro.createConnection();

// Get the connection with user-provided credentials from producer directly.
con = pro.createConnection("guest", "hello");

// Instantiate an empty factory
DBConnectionFactoryImpl factoryImpl = new DBConnectionFactoryImpl();

// Populate the factory with connection producers
factoryImpl.add("test", new JNDIConnectionProducer(jndi_name));

factoryImpl.add("production", new JDBCConnectionProducer(jdbcUrl, username, password));

factoryImpl.add("stock", new JNDIConnectionProducer(jndi_name, username, password));

// Set the default producer.
factoryImpl.setDefault("production");

// Obtain a default connection with configured credentials.
con = factoryImpl.createConnection();

// Obtain a named connection with specified credentials.
con = factoryImpl.createConnection("stock", "boss", "custom_password");

// Remove the "test" producer.
factoryImpl.remove("test");
```

```
// Check whether that producer is still in the factory
factoryImpl.contains("test"); // Should return false

// Set the "stock" producer to be the default one.
factoryImpl.setDefault("stock");

// Get the name for the default producer.
String defaultProducerName = factoryImpl.getDefault();

// Get the "production" producer
ConnectionProducer producer = factoryImpl.get("production");

// Now we want to traverse the producers contained in the factory.
Iterator iterator = factoryImpl.listConnectionProducerNames();
while (iterator.hasNext()) {
    producer = factoryImpl.get((String) iterator.next());
}
// Remove all the producers in the factory at last.
factoryImpl.clear();
```

5. Future Enhancements

The component could be extended to provide the methods to obtain the connections to be used in distributed environment (XAConnection, PooledConnection).