Pass-through of External Context Objects to ODA Data Providers – Project Features Specification

BPS #35

Version 1.~~3~~4: November ~~1~~10, 2005

Abstract

The Open Data Access (ODA) runtime API is currently a self-encapsulating framework. However, when used as an embedded component in an application server, it would be useful to provide the capability to pass in external objects through the BIRT engine and into a custom ODA data provider (driver). This document describes this feature enhancement and intended scope of support in BIRT 2.0 and Data Tools Platform (DTP) ODA version 3.0.

Document Revisions

Version	Date	Primary Author(s)	Description of Changes
Version 1.4	11/10/2005	Linda Chan	Updated section 2.1 to add a recommended practice on what to use for a context Map's key value. Updated section 2.3 to clarify how BIRT uses the ODA setAppContext methods.
Version 1.3	11/1/2005	Linda Chan	Updated sections 2.1 and 2.2 to reflect the Report Engine API changes to require a Map object for the application context object.
Version 1.2	10/7/2005	Linda Chan	Renamed BPS35 proposed methods from setContext to setAppContext to clarify that the context object is provided by an application and is opaque to the BIRT and ODA framework.
Draft 1.1	9/13/2005	Linda Chan	Revised the proposed changes in DtE and ODA API.
Draft 1.0	9/11/2005	Linda Chan	Initial Draft

Contents

1. Project Description and Use Cases

The BIRT engine is often embedded as part of a middle-tier application server, where the BIRT components are added to the mix of various J2EE components. During report run-time, some of these other components may instantiate context objects, which are served to the BIRT engine for its custom data source to use.

The BIRT engine would thus need to pass through such external Java object(s) into the BIRT data source components for consumption. The embedding application is expected to customize the underlying BIRT data source provider, such as an Open Data Access (ODA) run-time driver or BIRT Scripting data source. The custom data source provider would then know how to process the external object(s) passed to its context.

These context objects are assumed to be dynamically instantiated at report run-time, and are not statically defined in a report design. The scope of this project thus focus on data source specific run-time Java objects that are instantiated external of BIRT, and are passed through to a custom data source provider plugged in the BIRT engine.

The primary use cases for passing through external context objects to a BIRT data source provider are described below.

1.1 Use Case 1: Re-use Connection Handles

A middle-tier application manages its own data source connections, such as using its own security model, or provides load balancing. It has established an open connection handle that it wants to pass through to its custom ODA run-time driver plugged in the BIRT Engine for re-use. Examples of such external connection handle are:

- A pooled connection object that participates in connection pool management, and represents a physical connection to a data source.
- A JDBC connection object that is already open, and at a state ready for preparing and executing a statement for data retrieval.

The BIRT JDBC ODA run-time driver can then be extended and customized to re-use the external connection object in its ODA IDriver or IConnection implementation.

1.2 Use Case 2: Generated User Authorization Keys

An application manages its own user profiles and access control. It generates a user authorization key object at run-time that should then be used by BIRT as a connection property for access to the custom data source.

The authorization key should pass through the BIRT engine to the customized ODA run-time driver. The customized ODA IConnection implementation would then take this authorization key to open a physical connection to its data source.

1.3 Use Case 3: External Query Parameter Objects

An application collects relevant parameter values through its own user interface or API. It then in turn passes the run-time parameter values as Java objects to its customized ODA run-time driver in the BIRT engine. The customized implementation of the ODA IQuery interface would then apply these external parameter objects in its execution and data retrieval.

2. Proposed Solution

As illustrated in the various use cases, the content of the external context objects could vary depending on the nature of individual applications and their custom data source providers. To provide a generalized solution, the BIRT engine supports passing through context information as a java.lang.Object instance. A BIRT consumer application is free to define any types of object as appropriate. For example, a Map can be used to pass through multiple objects in the context. The underlying custom data source provider, such as an ODA run-time driver, should then be customized to process this context object.

An application that embeds the BIRT engine should specify the context object for each report execution task via the BIRT engine task API. The BIRT report engine would then pass the context object to the Data Engine as the context for accessing **all** the data sources and data sets defined in that report. The BIRT Data Engine would in turn pass the context object to its underlying custom data source providers, which are implemented by the embedding application.

Below sections describe the specific BIRT and ODA public APIs involved in the pass-through runtime operation.

2.1 Report Engine API

The BIRT Report Engine public API has an ~~existing~~ interface method that allows an application, which embeds the BIRT engine, to pass in an external context map object to the specialized engine task that runs a report.

```
package org.eclipse.birt.report.engine.api;
public interface IEngineTask
{
   …

   public abstract void setAppContext( java.util.Map context );
}

Usage sample:

IRunAndRenderTask runTask =
   (IRunAndRenderTask) new MyRunAndRenderTask();
runTask.setAppContext( contextMap );
```

The BIRT report engine would in turn pass the same application context object to **all** the components interested in the engine task context. Existing interested components include callback methods implemented by the embedding application, e.g. the implementation class of engine API's `IHTMLImageHandler`. It is up to each individual context recipient to process the external context object as appropriate.

It is very likely that an application would pass in various context objects to multiple BIRT components, such as Java scripting objects, images, ODA data-source specific context objects, etc. Thus, in BIRT 2.0, the `IEngineTask.setContext` method with an Object argument is changed ~~from~~ to `IEngineTask.setAppContext` ~~an Object to~~with a Map argument. Enforcing the use of a Map to pass in application context object(s) seeks to minimize the need for all context-handling components to revise their implementation, whenever an application adjusts its context object content.

Since the context Map will be available to potentially multiple recipients, the Map's key value is recommended (not required) to use a fully-qualified name of the intended recipient of the corresponding value object; for example: "com.company.data.myodadriver".

In BIRT 2.0, the BIRT Data Engine component will be added to the list of components to receive the context of the report run task. Thus the BIRT report engine will pass the same context ~~object~~ Map to the Data Engine as well, using the Data Engine API method described below, to be the context of **all** data set queries in that report run task.

2.2 Data Engine API

The BIRT Data Engine public API, in BIRT 2.0, adds a new interface method to allow one to provide a context map object for preparing and executing a report's data set query. The BIRT Report Engine is responsible for calling this method; it is thus transparent to the application that passes in the context. The Data Engine would in turn pass the context map as an object to its underlying custom ODA run-time driver, using the ODA API methods described in the next section.

```
package org.eclipse.birt.data.engine.api;

abstract public class DataEngine
{
   …

   /**
    * Verifies the elements of a report query spec
    * and provides a hint and application context object(s) to the
query
    * to prepare and optimize an execution plan.
    * This has the same behavior as the
    * prepare( IQueryDefinition querySpec ) method,
    * with an additional argument for an application to pass in a
    * context map object to the underlying data provider,
    * e.g. an ODA run-time driver.
    * @param querySpec     Specifies the data access
    *                and data transforms services needed
    *                from DtE to produce a set of query results.
    * @param appContext    The application context map for
    *                preparation and execution
    *                of the querySpec.
    * @return        The <code>IPreparedQuery</code> object that
    *                contains a prepared query ready for execution;
    *                could be null.
    * @throws        BirtException if error occurs during the
    *                preparation of querySpec
    * @since         2.0
    */
  abstract public IPreparedQuery prepare( IQueryDefinition querySpec,
                                          Map appContext )
              throws BirtException;
}
```

☞ *Note: In BIRT 2.0, if a BIRT scripting data source provider, instead of a custom ODA driver, supports the query definition being prepared, the context object is not passed through. The BIRT report engine API currently provides a more direct channel to pass through a context object to a BIRT Scripting object:*

```
package org.eclipse.birt.report.engine.api;
IEngineTask.addScriptableJavaObject(String jsName, Object obj);
```

This approach seems to be sufficient for the known use cases. When there are use cases where a context object should pass through the BIRT Data Engine to a BIRT Scripting data source provider, the Data Engine can then provide further support in future releases.

2.3 ODA API

The DTP ODA run-time public API in ODA version 3.0, which is used by BIRT 2.0, adds new interface methods, `setAppContext`, to allow ~~one~~ an ODA consumer application to pass an application context object into an ODA driver instance, plus each of its data source connection and data set query instances.

~~The BIRT Data Engine is responsible for calling these methods to push the application context object to an underlying ODA driver. An application context object could be null.~~ Note that the ODA `setAppContext` methods take an Object argument for application context, whereas BIRT Engine API enforces a Map argument type due to the nature of its expected usage. The ODA interfaces are designed for use by any ODA consumer applications, which may or may not have the requirement to use a Map. When an application customizes an ODA driver to handle its context, it would be in the position to know the type of context object that it passes to an ODA driver, and would implement the ODA `setAppContext` methods to handle it accordingly.

The `setAppContext` methods may be called with a null argument, i.e. passing a null context object to an ODA ~~driver~~ instance, only if a non-null context was previously passed through to the same ODA ~~driver~~ instance.

In the BIRT engine implementation, the BIRT Data Engine serves as the ODA consumer, and is responsible for calling these ODA methods to push the application context object to an underlying ODA driver. If `IEngineTask.setAppContext` was not called, or was called with a null argument, the BIRT Data Engine will **not** call the ODA `setAppContext` method. Other ODA consumer applications may however choose to handle it differently, and may call `setAppContext` multiple times on the same ODA instance, to change or reset the application context.

A custom ODA run-time driver provided by an application, which embeds the BIRT engine, should implement these ODA interface setter methods to process the context object, as appropriate.

```
package org.eclipse.datatools.connectivity.oda;

public interface IDriver
{
   …

   /**
    * Sets the driver context passed through from an application.
    * Its handling is specific to individual driver implementation.
    * Note: This method should be called before
    * getConnection( String ).
    * An optional method.
    * If any part of the context is not recognized by the driver,
    * it should simply ignore, and not throw an exception.
```

```
     * @param context Application context object of this instance.
     * @throws OdaException       if data source error occurs
     * @since         3.0
     */
    public void setAppContext( Object context ) throws OdaException;
}

public interface IConnection
{
    …

    /**
     * Sets the connection context passed through from an application.
     * Its handling is specific to individual driver implementation.
     * Note: This method should be called before open().
     * An optional method.
     * If any part of the context is not recognized by the driver,
     * it should simply ignore, and not throw an exception.
     * @param context Application context object of this instance.
     * @throws OdaException       if data source error occurs
     * @since         3.0
     */
    public void setAppContext( Object context ) throws OdaException;
}

public interface IQuery
{
    …

    /**
     * Sets the query context passed through from an application.
     * Its handling is specific to individual driver implementation.
     * Note: This method should be called before prepare().
     * An optional method.
     * If any part of the context is not recognized by the driver,
     * it should simply ignore, and not throw an exception.
     * @param context Application context object of this instance.
     * @throws OdaException       if data source error occurs
     * @since         3.0
     */
    public void setAppContext( Object context ) throws OdaException;
}
```