PROGRESS SOFTWARE

Artix® ESB

Developing Artix Database Services

Version 5.5
December 2008

Developing Artix Database Services

Version 5.5

Published 03 Dec 2008

Legal Notices

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

About This Book

What is Covered in This Book

This book explains how to expose a database as a Web service using Artix ESB command line tools and Artix Designer.

Who Should Read This Book

This book is intended for Java developers using Artix ESB. It assumes that you have a good knowledge of the following:

- General programming concepts
- General database concepts
- Structured Query Language

How to Use This Book

This book is organized into the following chapters:

- *Introducing Artix Database Services* on page 17 provides an overview of the Artix database Web services.
- *Building a JAX-WS DB Service on the Command Line* on page 21 explains how to use the command line interface to build database services for the Artix ESB Java Runtime.
- *Building a JAX-RPC DB Service on the Command line* on page 35 explains how to use the command line interface to build database services for the Artix ESB C++ Runtime.
- *Creating DB Services with Artix Designer* on page 41 explains how to build database services for either runtime using the Artix Designer GUI tools.

The Artix ESB Documentation Library

For information on the organization of the Artix ESB library, the document conventions used, and where to find additional resources, see Using the Artix ESB Library [http://www.iona.com/support/docs/artix/5.1/library_intro/index.htm].

Introducing Artix Database Services

This chapter provides an overview of how to expose a database as a Web service using Artix ESB.

Exposing a Database as a Web Service

In a database Web service, the WSDL operations are implemented by SQL queries and stored procedures defined in the database. You can generate a database Web service in either JAX-RPC using the Artix ESB C++ Runtime or in JAX-WS using the Artix ESB Java Runtime. You can define the service using either the Artix command line tools or Artix Designer.

Supported DBMSs

You can use Artix to expose the following database management systems as Web services:

- MySQL 4.0 and higher
- Oracle8i 8.1.7 and higher
- Sybase 12.5 and higher
- Microsoft SQL Server 2000 or higher
- Apache Derby (Artix ESB Java runtime only)

Database configuration files

The starting point of any Artix database Web service is the DB service configuration file. This file is in XML format and takes the extension `.xml.db`.

You can create the DB config file by hand or you can generate it using Artix Designer. If you are using the Artix ESB Java Runtime, you can also generate the file using the **artix sql2dbconfig** command.

Note

The DB config files used with the Java and the C++ runtimes are based on different schemas and are not compatible.

Sample Applications

Artix ESB includes sample applications that demonstrate how to expose a database query and a stored procedure as Web services in both the Java and C++ runtime. Each sample folder includes a `readme` file that explains how to run the application.

Java runtime sample

The Artix ESB Java Runtime sample uses Apache Derby as its embedded database. You can find the sample here:

ArtixInstallDir\java\samples\dbservice\basic

C++ runtime sample

The Artix ESB C++ Runtime sample does not ship with a database, but should work with any of the DBMSs listed in Supported DBMSs on page 18. You can find the sample here:

ArtixInstallDir\cxx_java\samples\db_service

Building a JAX-WS DB Service on the Command Line

This chapter explains how to create a database service for the Artix ESB Java Runtime using the command line interface.

Creating a DB Config File

The database configuration file is the starting point of any Artix database service. It contains details of the database connection that you wish to expose as a Web service and it is where you can map SQL queries to WSDL operations.

JAX-WS DB config file format

You can find the schema for the DB config file in the *ArtixInstallDir*\java\lib\it-soa-dbservice-api-dbconfig-*version*.jar library.

Example 1 on page 22 shows the format of a complete DB config file for use with the Artix ESB Java Runtime.

Example 1. JAX-WS DB Config File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DbServiceDescription
    xmlns="http://schemas.iona.com/dbservice/config">
  <Connection>
    <DriverClass>org.apache.derby.jdbc.EmbeddedDriver</DriverClass>
    <ConnectionUrl>jdbc:derby:derbyDB</ConnectionUrl>
    <Properties value="user1" name="password"/>
    <Properties value="user1" name="user"/>
    <C    onnectionPool enabled="false"/>
  </Connection>
  <Operations name="myQuery">
    <Sql query="select * from emp where ename = ?" isStoredProcedure="false"/>
    <Parameters position="1" jdbcType="12" isNullable="true" direction="IN"/>
    <Results type="RESULT_SET" position="1">
      <Columns jdbcType="2" isNullable="true" columnName="EMPNO"/>
      <Columns jdbcType="12" isNullable="true" columnName="ENAME"/>
      <Columns jdbcType="12" isNullable="true" columnName="JOB"/>
      <Columns jdbcType="2" isNullable="true" columnName="MGR"/>
      <Columns jdbcType="91" isNullable="true" columnName="HIREDATE"/>
      <Columns jdbcType="2" isNullable="true" columnName="SAL"/>
      <Columns jdbcType="2" isNullable="true" columnName="COMM"/>
      <Columns jdbcType="2" isNullable="true" columnName="DEPTNO"/>
    </Results>
```

```
  </Operations>
</DbServiceDescription>>
```

Generating a DB config file

When creating a database Web service for the Artix ESB Java Runtime, you do not need to build the DB config file from scratch. You can generate it in the correct format using the **artix sql2dbconfig** command.

sql2dbconfig syntax

The syntax of the **artix sql2dbconfig** command is as follows:

`artix sql2dbconfig` [`-d` *`output-dir`*] [`-new` [`-driver` *`driver-class`*] [`-connectionurl` *`connection-url`*] [`-property` *`property`* ...] [`-pool`] [`-maxactive` *`pool-maxactive`*] [`-maxidle` *`pool-maxidle`*] [`-transaction` *`transaction-level`*] [`-autocommit` { true | false }] [`-readonly` { true | false }]] [`-test`] [`-add` [`-name` *`name`*] [`-query` *`query`*] [`-isprocedure`] [`-isupdate`] [`-oktoexecute`] [`-parametertype` *`parameter-type`* ...] [`-parameterdirection` *`parameter-direction`* ...] [`-parameternullable` *`parameter-nullable`* ...] [`-parametervalue` *`parameter-value`* ...] [`-timeout` *`timeout`*]] [`-delete` [`-name` *`name`*]] [-v] [[-verbose] | [-quiet]] { *`outfile`* }

Required argument

The **artix sql2dbconfig** command takes one required argument: *`outfile`* specifies the path and the name of the generated database service configuration file. The default is `./dbconfig.xml.db`.

Optional arguments

The **artix sql2dbconfig** command takes the following optional arguments:

Option	Interpretation
`-d` *`output-directory`*	Specifies the directory into which the generated configuration file is placed.
`-new`	Specifies that a new configuration file is to be generated. Only connection information will be added to the newly created configuration file. You can add operations to the configuration file in subsequent commands by specifying the `-add` option. The target output file should not exist in

Option	Interpretation
	the filesystem. The following options can be used with the `new` option: • `-driver` • `-connectionurl` • `-property` • `-pool` • `-maxactive` • `-maxidle` • `-transaction` • `-autocommit` • `-readonly`
`-driver` *`driver-class`*	Specifies the JDBC driver class for the new connection. This option can only be used with `-new`.
`-connectionurl` *`connection-url`*	Specifies the connection url for the new connection. This option can only be used with `-new`.
`-property` *`property`*	Specifies a connection property for the new connection. This option can only be used with `-new`.
`-pool`	Specifies that connection pooling should be enabled. This option can only be used with `-new`.
`-maxactive` *`pool-maxactive`*	Specifies the maximum active connections in the connection pool. This option can only be used when both `-new` and `-pool` are specified.

Option	Interpretation
`-maxidle` *`pool-maxidle`*	Specifies the maximum idle connections in the pool. This option can only be used when both `-new` and `-pool` are specified.
`-transaction` *`transaction-level`*	Specifies the transaction isolation level for the new connection. This option can only be used with `-new`. The value should be an integer value as defined in JDBC specification or one of the following: `none`, `read_committed`, `read_uncommitted`, `repeatable_read`, and `serializable`.
`-autocommit` { true | false }	Specifies the auto commit value for the connection. This option can only be used with `-new`.
`-readonly` { true | false }	Specifies the read only value for the connection. This option can only be used with `-new`.
`-test`	Test a connection using the connection information provided in an configuration.
`-add`	Specified when adding a new operation to an existing database service configuration. This option cannot be used with `-new`. The following options can be used with `-add`: • `-name` • `-query` • `-isprocedure` • `-isupdate` • `-oktoexecute` • `-parametertype` • `-parameterdirection`

Option	Interpretation
	• `-parameternullable` • `-parametervalue` • `-timeout`
`-name` *name*	Specifies the name of the operation.
`-query` *query*	Specifies the query or procedure call of the operation.
`-isprocedure`	Specifies that the operation is a stored procedure.
`-isupdate`	Specifies that the operation will write to the database. If both `-isupdate` and `-isprocedure` are specified, `-oktoexecute` must be specified. If the operation requires IN/INOUT parameters, users must provide parameter values for executing the operation. The reason is that a stored procedure can return multiple results. The command needs to execute the stored procedure to obtain the result metadata. However, if only `-isupdate` is specified, the command does not need to execute the operation and the result is assumed to be an update count.
`-oktoexecute`	Specifies that it is OK to execute the operation to get resultset metadata.
`-parametertype` *parameter-type*	Specifies the JDBC type for a parameter by position. Positions start at 1. The value should be the integer or string value of the JDBC type as defined in the JDBC specification. For example `-parametertype 1=3` specifies that the first parameter is a JDBC decimal.
`-parameterdirection` *parameter-direction*	Specifies the direction for a parameter by position. Positions start at 1. The value should be one of `IN`, `INOUT`, and `OUT`.

Option	Interpretation
	For example, `-parameterdirection 2=IN` specifies that the second parameter is an input parameter.
`-parameternullable` *`parameter-nullable`*	Specifies whether a parameter is nullable by position. Positions start at 1. For example, `-parameternullable 1=true` specifies that the first parameter can be null.
`-parametervalue` *`parameter-value`*	Specifies a parameter's value by position. Positions start at 1. This option is necessary for parameterized operation when `-oktoexecute` is used. For example, `-parametervalue 1=12.0` specifies the first parameter's value is `12.0`.
`-timeout` *`timeout`*	Specifies the number of seconds before the operation times out.
`-delete`	Specifies that an operation is to be deleted from the database service configuration file.
`-name` *`name`*	Specifies the name of the operation to delete.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the generation.
`-quiet`	Suppresses comments during the generation.

Generating a WSDL File

You can generate the WSDL file that forms the basis of your database Web service by running the **artix dbconfig2wsdl** command.

dbconfig2wsdl syntax

The syntax of the **artix dbconfig2wsdl** command is as follows:

`artix dbconfig2wsdl` [-a *`address`*] [-d *`output-dir`*] [-servicename *`service-name`*] [-jdbctypemappings *`jdbc-type-mapping-file`*] [-mp { element | type }] [-t *`target-namespace`*] [-o *`output-file`*] [-logical] [-v] [[-verbose] | [-quiet]] { *`dbconfigurl`* }

Required argument

The **artix dbconfig2wsdl** command takes one required argument: *`dbconfigurl`* specifies the URL of the configuration file.

Optional arguments

The **artix dbconfig2wsdl** command takes the following optional arguments:

Option	Interpretation
`-a` *`address`*	Specifies the value of generated `soap:address` element's `location` attribute.
`-d` *`output-dir`*	Specifies the folder into which the generated WSDL is placed.
`-servicename` *`name`*	Specifies the value of the generated `service` element's `name` attribute. The default is `DataService`.
`-jdbctypemappings` *`jdbc-type-mapping-file`*	Specifies the name of the file containing the mappings between JDBC types and XSD types. See Customizing JDBC Type Mappings on page 33 for details.
-mp { element | type }	Specifies if the generated message parts should be types or elements. The default is elements.
`-t` *`target-namespace`*	Specifies the target namespace for the generated contract.
`-o` *`output-file`*	Specifies the name of the generated WSDL document.

Option	Interpretation
`-logical`	Specifies the tool only generates the logical portion of the WSDL document.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Generating JAX-WS Database Service Code

You use the **artix wsdl2dbservice** command to generate the code for an Artix ESB Java Runtime database service.

wsdl2dbservice syntax

The syntax of the **artix wsdl2dbservice** command is as follows:

`artix wsdl2dbservice` [-jdbctypemappings *`jdbc-type-mapping-file`*] [-db *`data binding name`*...] [-wv *`wsdl version`*...] [-p [*`wsdl namespace`*=]*`Package Name`*...] [-sn [*`wsdl namespace`*=]*`Package Name`*] [-b *`binding-name`*...] [-d *`output-directory`*] [-compile] [-classdir *`compile-classes-directory`*] [-impl] [-server] [-client] [-all] [-ant] [-nexclude *`schema namespace`* [=*`java packagename`*]...] [-exsh { true | false }] [-dns { true | false }] [-dex { true | false }] [-validate] [-wsdlLocation *`wsdlLocation attribute`*] [-v] [[-verbose] | [-quiet]] { *`wsdlurl`* }

Required argument

The **artix wsdl2dbservice** command takes one required argument: *`wsdlurl`* specifies the WSDL file from which the database service is generated.

Optional arguments

The **artix wsdl2dbservice** command takes the following optional arguments:

Option	Interpretation
`-jdbctypemappings` *`jdbc-type-mapping-file`*	Specifies the location of the JDBC to XSD mapping file. See Customizing JDBC Type Mappings on page 33 for details.
`-db` *`data binding name`*	Specifies the data binding to use. The default is JAXB.
`-wv` *`wsdl version`*	Specifies the WSDL version to use. The default is WSDL 1.1.
`-p` [*`wsdl namespace`*=] *`Package Name`*	Specifies the Java package name to use for the generated code. Optionally, you can specify the WSDL namespace mapping to a particular Java package name.
`-sn` [*`wsdl namespace`*=] *`Package Name`*	Specifies the service name to use for the generated code. Optionally, you can specify the WSDL namespace.

Option	Interpretation
`-b` *`binding-name`*	Specifies an external JAXWS or JAXB binding files.
`-d` *`output-directory`*	Specifies the directory into which the generated code is placed.
`-compile`	Specifies that the generated code is compiled.
`-classdir` *`compile-classes-directory`*	Specifies the directory into which the compiled class files are placed.
`-impl`	Generates a dummy implementation class.
`-server`	Generates a server mainline for the service.
`-client`	Generates the code needed to deploy a client.
`-all`	Generates all starting point code: types, service proxy, service interface, server mainline, client mainline, implementation object, and an Ant `build.xml` file.
`-ant`	Generates an Ant `build.xml`.
`-nexclude` *`schema-namespace`* [=*`java-packagename`*]	Ignore the specified WSDL schema namespace when generating code. This option may be specified multiple times. Also, optionally specifies the Java package name used by types described in the excluded namespace(s).
-exsh { true | false }	Enables or disables processing of extended soap header message binding.
-dns { true | false }	Enables or disables the loading of the default namespace package name mapping. Default is `true`.
-dex { true | false }	Enables or disables the loading of the default excludes namespace mapping. Default is `true`.
`-validate`	Enables validating the WSDL before generating the code.

Option	Interpretation
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool surpresses most messages.
`-verbose`	Specifies that the tool displays verbose messages.

Customizing JDBC Type Mappings

The mappings from JDBC types to the XML Schema types used in a Web service are controlled by a `JdbcTypeMapping.xml` file in the

ArtixInstallDir`/java/lib/it-soa-dbservice-rt-typemappers-`*version*`.jar`

library. Normally, there is no need to alter these mappings. However, where you are working with an unsupported database, you can customize them by creating a custom `JdbcTypeMapping.xml`.

Default mappings

The default JDBC to XML mappings are as follows:

Table 1. Default JDBC to XSD Type Mappings

JDBC Type	Artix Java Type	XML Schema Type
CHAR	String	string
VARCHAR	String	string
LONGVARCHAR	String	string
NUMERIC	java.math.BigDecimal	decimal
DECIMAL	java.math.BigDecimal	decimal
BIT	Boolean	boolean
BOOLEAN	Boolean	boolean
TINYINT	Integer	byte
SMALLINT	Integer	short
INTEGER	Integer	int
BIGINT	Long	long
REAL	Float	float
FLOAT	Double	double
DOUBLE	Double	double
BINARY	byte[]	hexBinary
VARBINARY	byte[]	hexBinary
LONGVARBINARY	byte[]	hexBinary
DATE	java.sql.Date	date
TIME	java.sql.Time	time
TIMESTAMP	java.sql.Timestamp	dateTime

JDBC Type	Artix Java Type	XML Schema Type
BLOB	byte[]	hexBinary
CLOB	String	string

Using alternative mappings

When you run the **artix dbconfig2wsdl** and **artix wsdl2dbservice** commands, the default JDBC to XSD mappings are used unless you use the `-jdbctypemappings`*`jdbc-type-mapping-file`* argument to point to a customized mapping file.

Building a JAX-RPC DB Service on the Command line

This chapter explains how to create a JAX-RPC database service for the Artix ESB C++ Runtime using the command line interface.

Creating a DB Config File

The database configuration file is the starting point of any Artix database service. It contains details of the database connection that you wish to expose as a Web service and it is where you can map SQL queries to WSDL operations.

JAX-RPC DB config file format

The examples below show DB config files for use with the Artix ESB C++ Runtime.

Example 2. Basic DB Config File for Artix ESB C++ Runtime

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<db-service>
  <operation name="findEmployeeByID" isStoredProcedure="false"
    timeout="3000" transactionIsolocation="Read.Committed"
    isReturnUpdateCount="false">
    <SQL_query>SELECT * FROM employee_account WHERE ID=?</SQL_query>
    <parameter jdbctype="INTEGER" name="employeeId" direction="IN"/>
    <result columnName="ID" jdbctype="INTEGER"/>
    <result columnName="Name" jdbctype="LONGVARCHAR"/>
    <result columnName="PhoneNumber" jdbctype="INTEGER"/>
    <result columnName="Birthday" jdbctype="DATE"/>
    <result columnName="Salary" jdbctype="REAL"/>
  </operation>
  <operation name="findEmployeeCount" isStoredProcedure="false" isReturnUpdateCount="false">

    <SQL_query>SELECT Count(*) FROM employee_account</SQL_query>
    <result columnName="Count(*)" jdbctype="BIGINT"/>
  </operation>
  <operation name="updateEmployeeDetail" isStoredProcedure="false" isReturnUpdateCount="true">

    <SQL_query>UPDATE employee_account SET PhoneNumber=? WHERE Name=?</SQL_query>
    <parameter jdbctype="INTEGER" name="phoneNumber" direction="IN"/>
    <parameter jdbctype="LONGVARCHAR" name="employeeName" direction="IN"/>
  </operation>
  <jdbc-connection>
    <driverclass>com.mysql.jdbc.Driver</driverclass>
    <connectionurl>jdbc:mysql://localhost:3306/DBSAMPLE</connectionurl>
    <username>root</username>
    <password/>
    <maxActive>10</maxActive>
    <maxIdle>5</maxIdle>
  </jdbc-connection>
</db-service>
```

Example 3. DB Config File including Stored Procedure

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<db-service>
  <operation name="addSalary"
    isStoredProcedure="true" isReturnUpdateCount="false">
    <SQL_query>{Call add_salary_inout(?,?,?) }</SQL_query>
    <parameter jdbctype="INTEGER" name="employeeId" direction="IN"/>
    <parameter jdbctype="VARCHAR" name="na" direction="OUT"/>
    <parameter jdbctype="FLOAT" name="sa" direction="INOUT"/>
  </operation>
  <jdbc-connection>
    <driverclass>com.mysql.jdbc.Driver</driverclass>
    <connectionurl>jdbc:mysql://localhost:3306/DBSAMPLE</connectionurl>
    <username>root</username>
    <password/>
    <maxActive>10</maxActive>
    <maxIdle>5</maxIdle>
  </jdbc-connection>
</db-service>
```

Creating a DB config file

Unlike the Java runtime, there is now way to generate a DB config file for the Artix ESB C++ Runtime. You can either create the file using a text editor, following the examples above, or use Artix Designer to generate and edit the file.

Generating WSDL and Code

You use the **dbconfigtowsdl** command to generate a WSDL document from an Artix ESB C++ Runtime DB config file. You can also use it to generate code for your database service, provided you do not plan to make further changes to the WSDL.

dbconfigtowsdl syntax

The syntax of the **dbconfigtowsdl** command is as follows:

`dbconfigtowsdl` [-a *`bindingAddress`*] [-fasttrack] [-plugin] [-p *`packageName`*] [-d *`dir`*] [-source *`dir`*] [-h] [-ant] [-v] [[-quiet] | [-verbose]] { *`dbconfigurl`* }

Required arguments

The **dbconfigtowsdl** command takes one required argument: *`dbconfigurl`* specifies the URL of the database configuration file.

Optional arguments

The **dbconfigtowsdl** command takes the following optional arguments:

Option	Interpretation
`-t` *`bindingAddress`*	Specifies the address to use in the port element of the generated WSDL. This flag is only valid when `-fasttrack` is also used. The default is `http://localhost:9000/DBConnection`
`-fasttrack`	Specifies that the tool will generate a default SOAP binding and HTTP endpoint for the database operations. In addition, the tool will generate the code for the intermediary required to expose the operations as a service.
`-plugin`	Specifies that the intermediary is generated as an Artix ESB C++ Runtime plug-in. This flag is only valid when `-fastttrack` is also used.
`-p` *`packageName`*	Specifies the Java package name to use for the genrated code.
`-d` *`dir`*	Specifies the output directory for the generated WSDL file. The default is the local directory. When `-fasttrack` is used, the default is `etc`.

Option	Interpretation
`-source` *dir*	Specifies the output directory for the generated code. This flag is only valid when `-fasttrack` is also used. The default is `java`.
`-h`	Displays the tool's usage statement.
`-v`	Displays the version number for the tool.
`-verbose`	Displays comments during the code generation process.
`-quiet`	Suppresses comments during the code generation process.

Generating JAX-RPC Database Service Code

You use the **wsdltodbservice** command to generate JAX-RPC compliant Java code from a WSDL file and database configuration file.

wsdltodbservice syntax

The syntax of the **wsdltodbservice** command is as follows:

`wsdltodbservice` [-d *`dir`*] [-source *`dir`*] [-plugin] [-h] [-v] [[-quiet] | [-verbose]] { *`dbconfig`* } { *`wsdlurl`* }

Required argument

The **wsdltodbservice** command takes one required argument: *`wsdlurl`* specifies the WSDL file from which the database service is generated.

Optional arguments

The **wsdltodbservice** command takes the following optional arguments:

Option	Interpretation
`-d` *`dir`*	Specifies the output directory for the generated DB service.
`-source` *`dir`*	Specifies the output directory for the generated source code. The default is `java`.
`-plugin`	Specifies that the DB service is to be generated as a plug-in for deployment into an Artix ESB C++ Runtime container.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.
`-quiet`	Specifies that the tool is to run in quiet mode.
`-verbose`	Specifies that the tool is to run in verbose mode.

Creating DB Services with Artix Designer

This chapter explains how to expose a database as a Web service using Artix Designer.

Creating a Database Services Project

Choosing a runtime

When building anything in Artix Designer The first step is to create a project. Artix Designer supports projects for building database Web services in both the Artix ESB Java Runtime, using the JAX-WS specification, and the Artix ESB C++ Runtime, using JAX-RPC.

Opening the Artix Database perspective

The Artix Database perspective contains all the views and editors that you need to create and edit a database Web services project.

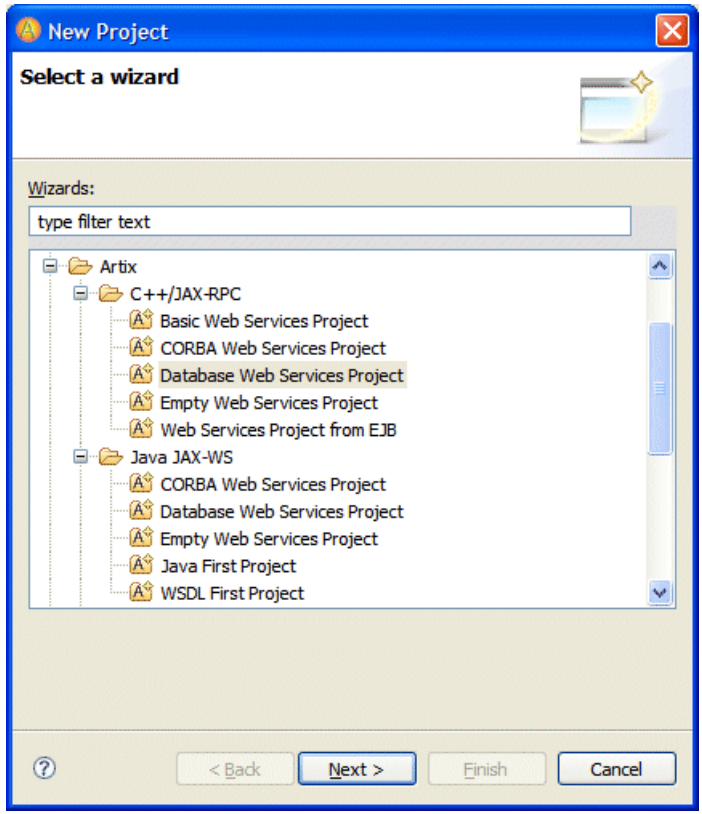
To open the Artix Database perspective, Select **Window** → **Open Perspective** → **Artix Database**.

Creating a project

To create a database Web services project in Artix Designer:

1. Select **File** → **New** → **Project from the menu bar**.
2. In the New Project wizard, expand the **Artix** folder.
3. Depending on the runtime that you want to use, expand either the **C++/JAX-RPC** folder or the **Java JAX-WS** folder.

Figure 1. The New Project Wizard



4. Select **Database Web Services Project** and click **Next**.
5. In the General Details panel, enter a name for the project and click **Next**.
6. In the Bookmark Details panel, select a bookmark from the drop-down list or click **New** to create a bookmark. See Creating a bookmark on page 44.

7. Click **Finish**.

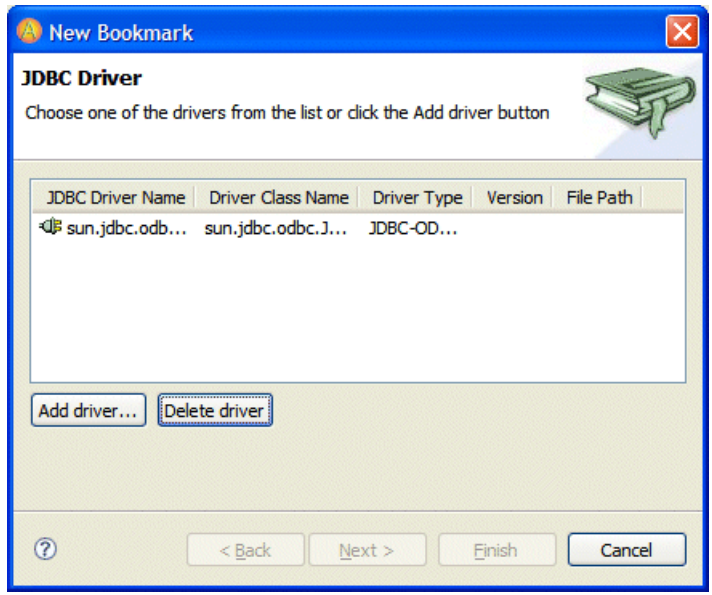
Creating a bookmark

In Artix Designer, a bookmark is a saved database connection. You can create a bookmark from within the New Project wizard or from the Database Bookmarks view in the Artix Database perspective:

To create a bookmark from the Database Bookmarks view:

1. Ensure that the Artix Database perspective is open. See Opening the Artix Database perspective on page 42.
2. Right-click in the Database Bookmarks view and select **New Bookmark**.
3. In the New Bookmark wizard, click the **Add driver** button.

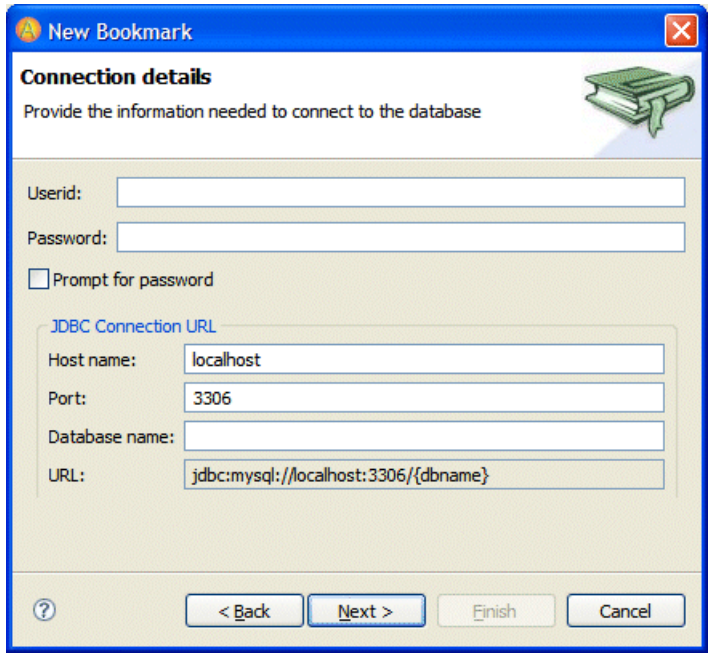
Figure 2. The New Bookmark Wizard



4. In the New JDBC Driver wizard click **Add External Jar** to select a JAR file containing the JDBC driver from your file system.

5. Click **Browse** to select the driver from the JAR file. The following JDBC drivers are supported:
   - MySQL - `com.mysql.jdbc.Driver`
   - Oracle - `oracle.jdbc.driver.OracleDriver`
   - Sybase - `com.sybase.jdbc3.jdbc.SybDriver`
6. Click **Finish** to return to the New Bookmark wizard.
7. Select the newly-added JDBC driver and click **Next**.
8. In the Connection Details panel, enter the following details:
   - The username and password that you use to connect to your DBMS
   - The name of the machine hosting the database
   - The port number that the database is running on
   - The database name

Figure 3. The Connection Details Panel



9. Click **Next**.

10. Provide a unique name for the database bookmark and click **Finish**.

Editing the DB Config File

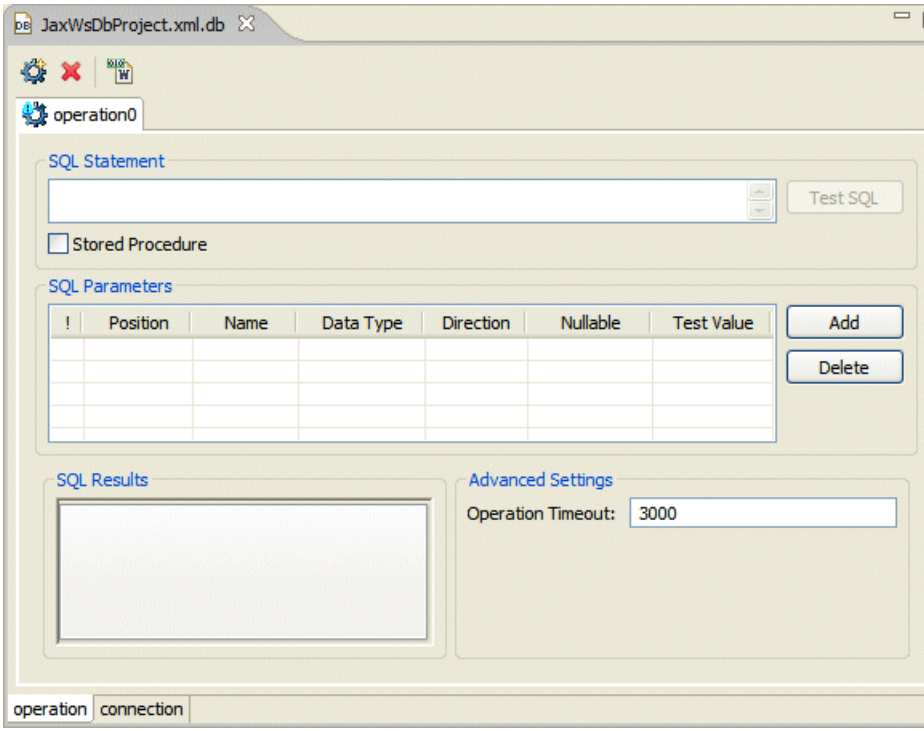
The database configuration file is at the heart of all Artix database Web services. It contains details of the database connection that you wish to expose as a Web service and it is where you can map SQL queries to WSDL operations.

Using the DB Config File Editor

Although the format of the DB config file differs depending on whether you are creating a service for the Artix ESB Java Runtime or the Artix ESB C++ Runtime, you need not worry about this when using Artix Designer. The DB Config File Editor creates the correct file format depending on which project type you have created.

As soon as you create a database Web services project, the DB Config File Editor opens with the **Operation** tab displayed.

Figure 4. The DB Config File Editor



Adding SQL statements and parameters

Enter your SQL query in the **SQL Statement** field. SQL statements must conform to SQL-92 and JDBC 1.x.

If the query contains a parameter, as in Example 4 on page 48 click the **Add** button to specify it in the **SQL Parameters** field.

Example 4. An SQL Query Containing a Parameter

```
SELECT * from employee_account WHERE name like ?
```

Specifying stored procedures

If the query that you have entered in the **SQL Statement** field is a stored procedure select the **Stored Procedure** check box. You should enclose the SQL statement that calls the stored procedure in braces, as shown in

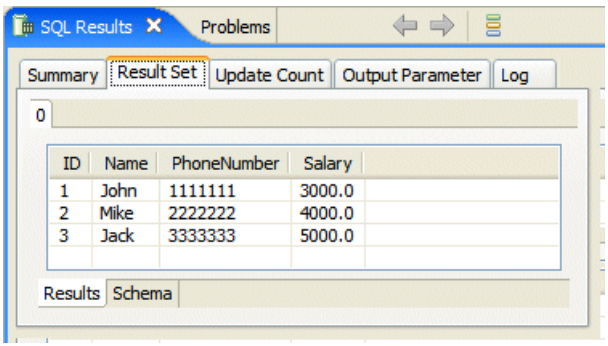
Example 5. Calling a Stored Procedure

```
{call get_all}
```

Testing your query

Once you have completed your query and added all the necessary parameters, click the **Test SQL** button. The output is returned to the **Result Set** tabbed page of the **SQL Results** view.

Figure 5. The SQL Results view



Renaming operations

By default any new operations that are added to the DB Config Editor are called "operation0". To rename an operation to something more meaninful, right-click the operation in the Outline view and select **Rename**.

Adding and deleting operations

To add operations to the DB config file, click the icon at the top of the DB Config Editor.

To delete an operation click the icon.

Generating WSDL and Code

Once you have completed the DB config file, you are ready to generate WSDL and JAX-WS or JAX-WS compliant Java code for your database Web service.

To generate code, click the  icon at the top of the DB Config Editor.

JAX-RPC generation options

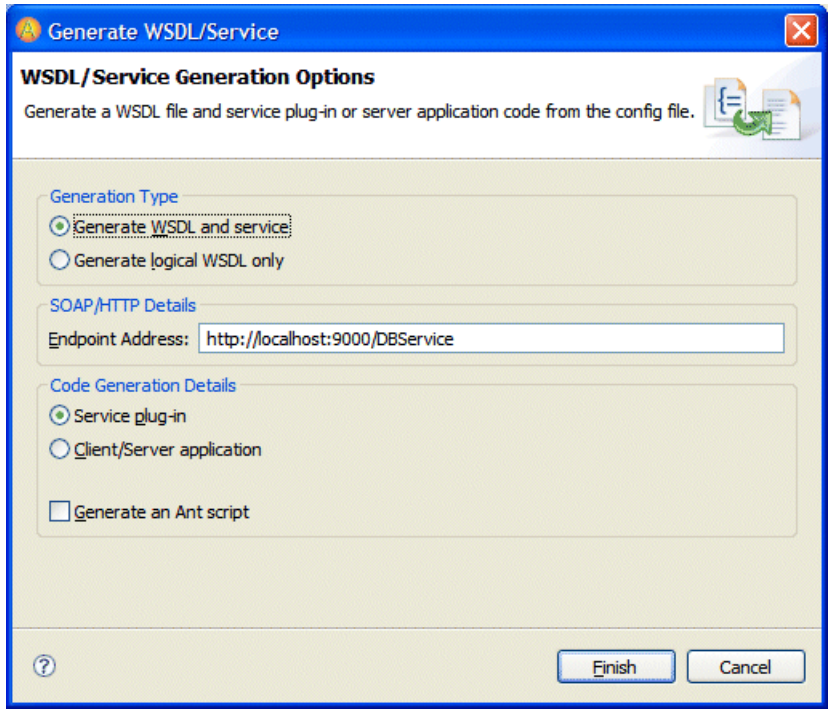
When generating WSDL and JAX-RPC compliant Java code for the Artix ESB C++ Runtime, the WSDL/Service Generation Options dialog box includes the following options:

Generation Type
: Select **Generate WSDL and service** to generate a full WSDL file and the code needed to run the service. Select **Generate logical WSDL only** to generate the logical part of a WSDL contract only. This is the equivalent of running the **dbconfigtowsdl** command without the `-fasttrack` argument.

SOAP/HTTP Details
: You can edit the port and service name in the **Endpoint Address** field, as required.

Code Generation Details
: Select **Service plug-in** to generate a service that you can deploy inside an Artix Container. Select **Client/server application** to generate a client application and a server mainline. You can also choose to generate an Ant `build.xml` file to manage subsequent builds.

Figure 6. WSDL/Service Generation Options for a JAX-RPC Service



JAX-WS generation options

When generating WSDL and JAX-WS compliant Java code for the Artix ESB Java Runtime, the WSDL/Service Generation Options dialog box includes the following options:

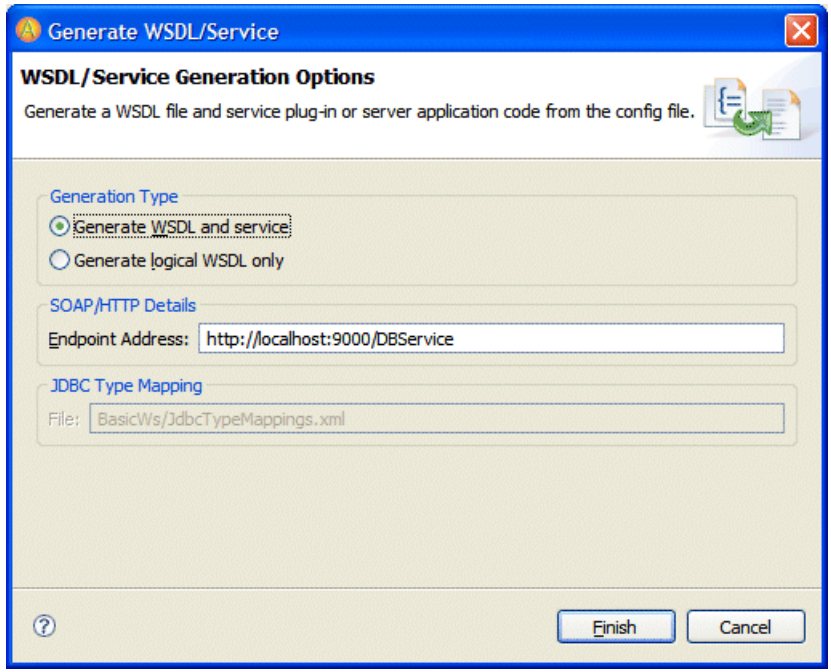
Generation Type
: Select **Generate WSDL and service** to generate a full WSDL file and the code needed to run the service. Select **Generate logical WSDL only** to generate the logical part of a WSDL contract only. This is the equivalent of running the **dbconfig2wsdl** command with the `-logical` argument.

SOAP/HTTP Details
: You can edit the port and service name in the **Endpoint Address** field, as required

Note

When you generate JAX-WS code for a database service, client and server mainline applications are generated by default.

Figure 7. WSDL/Service Generation Options for a JAX-WS Service



Index