WmDB User's Guide

Version 10.1

October 2017

Document ID: IS-WMDB-UG-101-20171017

Table of Contents

About this Guide

This guide describes how to use the WmDB package to access a database. The WmDB package contains services and Dynamic Server Pages (DSPs) that you can use to build services that connect to databases. The webMethods Integration Server can connect to databases through WmDB flow services; these WmDB services can then be invoked through Java and C/C++ clients. This guide is for users who want to connect to databases in this manner.

This guide describes how to:

- Configure access to database systems by supplying database connection information.
- Access databases with flow services or through Java or C/C++ clients.

Note: The WmDB package is not included in the main "Program" component of webMethods Integration Server. If you need this package, be sure to select WmDB on the list of installable components during installation.

Important: The WmDB package has been deprecated. When developing new applications, use the webMethods Adapter for JDBC instead of the WmDB package. The Adapter for JDBC provides more functionality, may improve performance, and supports adapter notifications. For more information about the Adapter for JDBC, see *webMethods Adapter for JDBC Installation and User's Guide*.

There are certain scenarios in which you might need to continue using the WmDB package. For more information, see "Determining Whether to Use the WmDB Package or the Adapter for JDBC " on page 33.

Database Drivers

To use the WmDB package, you must use a valid type 2 JDBC driver on an operating system supported by the webMethods Integration Server.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.

Convention	Description
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Configuring Access to Database Systems

Overview

Before a server can connect to a database, the server must first receive information about the database. You supply database connection information by configuring a database *alias*. You must configure database aliases if you want to create database flow services.

This chapter describes what a database alias consists of and how to identify, update, and delete a database alias in the Integration Server Administrator.

Note: The WmDB package is not included in the main "Program" component of webMethods Integration Server. If you need this package, be sure to select WmDB on the list of installable components during installation.

Supported Databases

The webMethods Integration Server can connect to the following databases using WmDB:

- Oracle 8i and 9i
- SQL Server 2000
- DB2 UDB 8.1
- MS SQL 7
- MS Access

WmDB connects to these databases using the JDBC driver provided by the vendor.

Important: The WmDB package does *not* support Sybase ASE 12.5.

Database Connection Information

A database alias consists of the following information:

- **The type of database and where it is located.** JDBC uses a URL naming scheme to locate and identify databases. Each JDBC driver responds to a slightly different syntax for these URLs. Consult the documentation for the driver you are using for details. If you are using the JDBC-ODBC bridge driver, the URL is `jdbc:odbc:`*`your-datasource`*.
- **The user name and password required to connect to the database.** If the database requires a user name and password to connect to it, you must specify the server with the user name and password it should supply.

- **The JDBC driver.** You identify the fully qualified Java class name of the JDBC driver to use for this connection (for example, `com.company.jdbc.Driver`). This class must be in the server's classpath.

The server uses the database connection information to connect to a database when:

- You use the Integration Server Administrator to display information about database tables when creating a database service (Generate from table). The server must connect to the database to get information about the structure of the database.
- The server executes a database service that was created using the Integration Server Administrator.

Note: For information about creating services that access a database, see "Accessing Databases with Services" on page 19.

About the Server Configuration File

The server configuration file (server.cnf), located in the *Integration Server_directory*\instances*instance_name* \config directory, contains the parameters you will add to control the server connection to the database.

Note: Typically, you will use the Settings > Extended screen from the Integration Server Administrator to update this file, but there might be times when you need to edit the file directly using a text editor. If you edit the file directly, you should first shut down the Integration Server before updating the file. After you make the changes, restart the server.

The following table describes the parameters that you must add to server.cnf to specify database connection settings:

This parameter...	Specifies...
watt.server.db.connectionCache	How the server manages connections to a database. ■ Specifying `server` tells the server to maintain a pool of connections for each database that is defined to the server through an alias. If a request cannot be satisfied because the pool has reached its maximum number of connections, the server blocks the request and tries again later. ■ Specifying `session` tells the server to treat requests from each service individually. That is, when it receives a request, the server immediately requests a connection from the database. If the database has no available connections, the request fails. This is the default.

This parameter...	Specifies...
	Although enabling database connection pooling creates a pool for each database defined to your server, you can control the characteristics of each pool *individually* by using the Edit Alias Information screen of the Integration Server Administrator. For more information, see " Enabling Database Connection Pooling" on page 16.
watt.server.jdbc.defaultDriver	The name of the Java class for the driver you want to use to connect to databases when no driver name is supplied for a database alias. For more information, see " Identifying Database Aliases" on page 13.
watt.server.jdbc.driverList	A comma-delimited list of JDBC drivers you want the server to load when it initializes. There is no default. For more information, see "Preloading JDBC Drivers at Server Initialization" on page 12.

Preloading JDBC Drivers at Server Initialization

You can have the server preload the JDBC drivers when it initializes. Loading the JDBC drivers at initialization increases the performance of the first run of the first database service that requires a specific JDBC driver. If the server does not have a JDBC driver loaded when a database service requires it, the server must load the JDBC driver when the database server executes. Also, if you preload JDBC drivers, the server displays the list of drivers in a drop down list on the screens that require a JDBC driver name. When the drop down list is available, you can select a driver from it rather than type the name of a driver.

To have the server load JDBC drivers when it initializes, specify a comma-delimited list of JDBC drivers that you want preloaded in the `watt.server.jdbc.driverList` field in the server.cnf file in the *Integration Server_directory*\instances*instance_name* \config directory. If this field does not exist in the server.cnf file, you can add it. To change this setting use the Settings > Extended screen from the Integration Server Administrator as described in *webMethods Integration Server Administrator's Guide*.

Making the Java Classes for JDBC Drivers Accessible

The server requires access to the Java classes for each JDBC driver that it is to use. You need to place the Java classes in a location that the server can access. Typically, you place the Java classes in the server's classpath.

To place the classes in the server's classpath, place the .zip or .jar file containing the classes in the *Integration Server_directory*\lib\jars\custom directory. If the jars or custom subdirectories do not exist, create them. The server will automatically add the .zip or .jar libraries to its classpath.

Note: If you want to make the Java classes available only to a specific Integration Server instance, copy the extracted .jar file into this location: *Integration Server_directory*/instances/*instance_name* /lib/jars/custom.

If a patch you are unzipping or “unjarring” overwrites existing classes, be sure to back up the existing classes in case the patch does not work.

Note: If you are using the Oracle JRE, the JDBC-ODBC bridge driver is `sun.jdbc.odbc.JdbcOdbcDriver`. An ODBC data source corresponding to your database *must* exist.

If the server does not have access to the Java classes for the JDBC driver, users receive the following error when the server attempts to connect to the database:

```
Could not connect: No suitable driver
```

If the driver loads successfully but cannot connect to the database for some reason, the resulting error message usually includes some driver-specific codes or messages. The most common source of such errors is an invalid user name or password on the database. Consult your driver documentation for details.

Identifying Database Aliases

Identify a database alias to specify the connection information that the server must supply to connect to a database. Set up aliases if:

- You want developers to be able to define database flow services using the webMethods Integration Server Administrator.
- You want to maintain all database connection parameters in one place to allow for easier maintenance. For example, you can easily update the information if a password changes or a database moves.

To better manage requests for database connections, you can specify that the server use *database connection pools*. With database connection pools in effect, the Integration Server controls the number of database connections it makes to a database, preventing the database from exceeding its connection limit and rejecting requests. If a request exceeds a database connection pool’s limit, the server blocks it and tries the connection request again later.

See " Enabling Database Connection Pooling" on page 16 for more information about how database connection pools work and how to set them up.

Note: Before you configure a database alias, make sure the WmDB package is enabled. For information about enabling packages, see *webMethods Integration Server Administrator’s Guide*.

To identify a database alias

1. Open the Integration Server Administrator if it is not already open.
2. In the **Adapters** menu of the navigation area, click **Database**. The server displays a database management screen.
3. In the navigation area, click **Alias Management**.
4. Click **Add**. The server displays the New DB Alias screen.
5. Set the new alias parameters as follows:

For this parameter...	Specify...
Alias	The alias name that you want to use for the database. You can specify any combination of letters, numbers, and the underscore character for an alias name; there is no restriction on the number or of characters you can specify.
DB URL	The URL for the database. For example: `jdbc:odbc:Support.`
DB Username	The user name that the server must supply to log on to the specified database. If a user name and password are not required, leave this field blank.
DB Password	The password that the server must supply to log on to the selected database. If a user name and password are not required, leave this field blank.
DB Driver	The name of the Java class for the JDBC driver. If the server has any drivers currently loaded, the server displays a **Loaded Drivers** field that has a drop down list containing the drivers that are loaded. If you select a driver from the list, the server places the driver name in the **DB Driver** field. If you do not specify a driver, the server uses the default driver specified in the `watt.server.jdbc.defaultDriver` parameter in the server.cnf file. To see the value of the default driver, use the Settings>Extended screen from the Integration Server Administrator as described in *webMethods Integration Server Administrator's Guide*. **Important:** Be sure the server has access to the Java classes for the driver. For more information about making the

For this parameter...	Specify...
	Java classes available, see "Making the Java Classes for JDBC Drivers Accessible" on page 12. If you are using the Oracle JRE on a Windows platform, the server has access to the Java classes for the JDBC-ODBC bridge driver named `sun.jdbc.odbc.JdbcOdbcDriver`.
Minimum Connections	The minimum number of connections to maintain in the database connection pool. The server creates these connections when it first receives a request to connect to the database. If more connections are needed, the server creates additional connections until the value for **Maximum Connections**, described below, is reached. The default is 0. **Note:** This field displays only if the server is configured for database connection pooling (that is, if the `watt.server.db.connectionCache` property in the server.cnf file is set to `server`). See " Enabling Database Connection Pooling" on page 16 for more information.
Maximum Connections	The maximum number of connections to maintain in the database connection pool. When the number of connection requests reaches this value, the server blocks the requests. The default is 1. **Note:** This field displays only if the server is configured for database connection pooling (that is, if the `watt.server.db.connectionCache` property in the server.cnf file is set to `server`). See " Enabling Database Connection Pooling" on page 16 for more information.
Expiration Time (ms)	The amount of time the server waits before discarding an inactive connection from the database connection pool. The server only discards connections if the pool contains more than the minimum number of connections. Afterwards, if more connections are needed than exist in the pool, the server creates new ones. The default expiration time is 60000 milliseconds.

For this parameter...	Specify...
	Note: This field displays only if the server is configured for database connection pooling (that is, if the `watt.server.db.connectionCache` property in the server.cnf file is set to `server`). See " Enabling Database Connection Pooling" on page 16 for more information.

6. Click **Submit**.

Enabling Database Connection Pooling

For greater scalability you can pool database connections. With this feature, the server creates a pool of connections for each database defined to the webMethods Integration Server. The server maintains these connections, creating and dropping them as needed.

The pool controls the number of connections a server can have to the database at any given time. For example, if your database allows 50 connections and you have five servers that connect to the database, you can limit each server's database connection pool to 10. This way, the number of connections to the database never exceeds 50 and no requests will be rejected due to lack of available slots. If the number of connection requests to a server exceeds 10, that server will block the request and try it again later. The request will not be rejected.

Without database connection pooling, each server will obtain as many connections to the database as requested until the database reaches its limit of 50. Subsequent requests will be rejected by the database until the number of database connections drops down below 50.

When you enable database connection pooling, it applies to *all* databases defined to the server. In other words, the server will maintain a connection pool for each database for which an alias is defined. You can, however, control the characteristics of each pool *individually*. For example, you might specify more connections for a large, busy database than for a small, relatively quiet one.

Use the following procedure to enable database connection pooling for your server.

To enable database connection pooling

1. Open the Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the navigation area, click **Extended**. The server displays a screen that lists configuration parameters specified in the server.cnf file.
3. Do one of the following:
   - If the **Extended Settings** lists `watt.server.db.connectionCache=server` then connection pooling is already enabled for your server.

- If the `watt.server.db.connectionCache` property is not set to `server` or does not appear in the **Extended Settings** list, click **Edit Extended Settings** and specify the following:

  ```
  watt.server.db.connectionCache=server
  ```

4. Click **Save Changes**.
5. If you want to change the alias parameters for the minimum and maximum number of connections in the connection pool and the connection expiration time, see "Updating Information for Database Aliases" on page 17.
6. Restart the server for the changes to take effect.

Updating Information for Database Aliases

If the connection information changes for a database, update the database alias information to identify the new connection information.

Note: This procedure assumes the database aliases have already been defined. If they have not, see " Identifying Database Aliases" on page 13 for instructions.

To update information for a database alias

1. Open the Integration Server Administrator if it is not already open.
2. In the **Adapters** menu of the navigation area, click **Database**. The server displays a database management screen.
3. In the navigation area, click **Alias Management**.
4. From the **Current Data Sources** field, select the database alias for the database you want to update.
5. Click **Edit**. The server displays the Edit Alias Information screen.
6. Set the alias parameters. For information about what to specify for each of the fields, refer to the procedure in " Identifying Database Aliases" on page 13.
7. Click **Submit**.
8. If you changed the **Minimum Connections** or **Maximum Connections** alias parameters, restart the server for the changes to take effect.

Deleting Database Aliases

When you no longer need access to a database, you can delete the database alias for the database.

To delete a database alias

1. Open the Integration Server Administrator if it is not already open.
2. In the **Adapters** menu of the navigation area, click **Database**. The server displays a database management screen.
3. From the **Current Data Sources** list, select the database alias you want to delete.
4. Click **Delete**. The server issues a prompt to verify that you want to delete the database alias.
5. Click **OK** to delete the database alias.

2 Accessing Databases with Services

Overview

All types of services can access databases, including flow services, Java service, and C/C ++ services. You can create:

- Flow services using the Integration Server Administrator. When you create a flow service that accesses a database, you make selections from Integration Server Administrator screens to describe the function that you want the service to perform. The webMethods Integration Server generates the flow service for you based on your selections.
- Java and C/C++ services using webMethods Integration Server or your own development environment. Integration Server provides built-in services that perform basic database operations, such as connecting to a database, selecting rows, inserting rows, and deleting rows. Your services can access these built-in services to perform database operations. Your services can also use other database APIs (for example, JDBC) to access databases.

This chapter describes how to decide which type of service to create, how to create database flow services, how to create database services with Java or C/C++, and how to create clients that access databases.

Deciding Which Type of Service to Create

Use the information in the following table to decide which type of service you should create.

Type of Service	Advantages	Disadvantages
Flow services created with the Integration Server Administrator	■ You can create the flow service using the Integration Server Administrator user interface. ■ You do not need to recompile code when you make changes to the flow service.	■ The flow service might not be able to handle nonstandard database features, such as Windows SQL types. ■ You can only make JDBC calls in a flow service. Use a Java or C/C++ service to connect to a database using other libraries.
Java or C/C++ Services	■ The service can be coded more flexibly in terms of processing data. You can manipulate data directly, rather than invoking	You must code and compile the service.

Type of Service	Advantages	Disadvantages
	Integration Server service libraries. For example, you can use configuration files to guide processing settings. ■ The service can include more robust error handling. For example, you can log errors to your own files or send e-mail notes to several people. ■ The service can make direct calls to database APIs (for example, JDBC, ADO, and ODBC) to handle nonstandard database features.	

Creating Database Flow Services

When you use the Integration Server Administrator to create flow services that access databases, you progress through a series of screens, selecting options and specifying information that indicates the function you want the service to perform. The server builds a flow service from the selections you specify.

The resulting flow service invokes the same built-in services that you can invoke from Java or C/C++. For more information about these services, see "Built-in Database Services" on page 29.

Using the Integration Server Administrator, you can create a flow service in one of the following ways:

- **Generate from tables.** You select the table that you want the service to access. Then, you select the database operation you want your service to perform (Select, Insert, Delete, or Update). The server displays the columns in the table that you can use to indicate the input that the service expects. When you use this option, the server automatically generates the required SQL statements. For more information, see "Generating a Database Flow Service from a Table" on page 25.
- **Generate from an SQL statement.** You specify the SQL statement that you want the service to perform. The SQL statement you specify can be more complex than the SQL statements the server generates when it generates a service from tables. In addition, you can specify database-specific SQL. Use this option if you want to use database-specific SQL that requires the fully qualified table names. For more information, see "Generating a Database Flow Service from an SQL Statement" on page 22.

Regardless of how you generate them, database flow services are all standard flow services. After a service is created, you can edit it using Designer in the same way you edit any other flow service. You can invoke the resulting database flow service from other services.

Note: All generated database flow services call the pub.db:execSQL service. This service executes an SQL statement. If you want to change the SQL statement the flow service executes, open that service in Designer and change the *$dbsql* parameter in the INVOKE execSQL step in the flow.

Important: Before you can use the Integration Server Administrator to build flow services, the server must have a configured database alias for the database that you want to access. The database alias must be configured when you create a service and when the server executes the service. For more information about how to configure a database alias, see " Identifying Database Aliases" on page 13.

Generating a Database Flow Service from an SQL Statement

To generate a service from an SQL statement, you must select the SQL statement that you want the service to perform. You can specify a static SQL statement or a dynamic SQL statement.

Important: The generated flow service is unable to properly call a stored procedure or return the result set or output parameters. If you want to call a stored procedure, create the service in Java or C/C++ and call the stored procedure using the pub.db:call service.

Specifying a Dynamic SQL Statement

You can create a dynamic SQL statement by including question marks or template tags in the SQL statement. When you specify a dynamic SQL statement, the service expects input values to replace the question marks or template tags that you specify.

Using Question Marks (?) in SQL Statements

Use a question mark in place of a single parameter that the service expects as input. When you test the service in Integration Server Administrator, it recognizes the question mark and prompts you for the required input value.

Example 1

To select all rows from the Names table that match the values specified for the Last_Name column, specify the following SQL statement:

```
select * from Names where Last_Name=?
```

The service expects an input value to use to match rows in the `Last_Name` column.

Example 2

To add a new row to the Addresses table and populate it with specified values, specify the following SQL statement:

```
insert into Addresses (name,street,city,state,zip)
values (?,?,?,?,?)
```

The service expects input values to use to populate the new row.

Using Template Tags in SQL Statements

The template tags you can use in an SQL statement are the same tags you use in an output template. Besides allowing you to specify the values of individual input parameters, template tags also allow you to dynamically construct entire portions of the SQL statement at run time. (For a complete list of template tags, see the *Dynamic Server Pages and Output Templates Developer's Guide*.)

When you use template tags in an SQL statement, you cannot test the statement with the Integration Server Administrator; (the Integration Server Administrator does not recognize template tags and will not prompt you for input when you execute the service). To test a service that uses template tags, you must open that service in Designer and add to its input parameters any variables that are referenced in a tag. You can then test the service with Designer and it will prompt you for each input variable you defined. For information about declaring input parameters for a service, see the *webMethods Service Development Help*.

Example 1

The following shows an SQL statement that selects all rows satisfying the criteria that will be specified in a variable named *condition* at run time:

```
select * from Names where %value condition%
```

At run time, the server will substitute the value of *condition* for the `%value%` tag. The following shows examples of what you might use as the value of *condition* :

```
name = 'Steve'
name like 'Jim%'
```

Example 2

To delete all rows from the Music table that meet a specified condition, specify the following SQL statement:

```
delete from Music where %value outdated%
```

The service expects input for an input variable named *outdated* that it uses to replace the entire token. Following is an example of what a user can specify for %value outdated%:

```
ReleaseDate < '1950'
Format = '8-track'
```

Steps for Generating a Flow Service from an SQL Statement

Use the following procedure to generate a flow service from an SQL statement.

To generate a flow service from an SQL statement

1. Open the Integration Server Administrator if it is not already open.
2. On the **Database** menu, select **Service Generation**.
3. In the **Source alias** list, select the database you want the service to access.
4. In the **Package** list, select the package in which you want the service to reside.
5. In the **Folder** field, type the name of the folder in which you want to save the service.
6. In the **Service** field, type the name you want to assign to the service.
7. In the **Execute ACL** list, select the ACL group to which you want to limit access to the service. Only the users who are members of the groups in the selected ACL will be able to invoke the service.
8. If you want to overwrite any existing services with the same name as the new service, next to **Overwrite Existing**, select the **Generated objects overwrite existing objects** check box.
9. Click **Generate from SQL**. The server displays the Edit SQL Statement screen.
10. In the **Enter SQL statement** section of the screen, enter the SQL statement you want the service to execute. You can specify a statement that is dynamic (that is, contains question marks or template tags) or static.
11. If you included template tags in the SQL statement, select the **Process webMethods template tags in this SQL statement** check box.
12. Click **Evaluate**. The server displays the Input Binding Generation screen.

    If you specified question marks (?) in the SQL statement for input values, the server displays a **Bind parameters** section of the screen. Under **Bind parameters**, one **Parameter #** field exists for each question mark you specified in the SQL statement. The first question mark corresponds to **Parameter 0**, the second to **Parameter 1**, and so forth. Use this section of the screen to indicate additional information about the input parameters that a user will supply in place of the question marks.
13. For each **Parameter #** field under **Bind parameters**, specify the following information:

For this field...	Specify or select...
name	The name that you want the service to use for the input parameter.
type	The data type of the input parameter.

14. Click **Generate**.

Generating a Database Flow Service from a Table

When you specify that you want to generate the service from database tables, the server displays information about the tables in the selected database. If your database contains many tables, you can restrict the list of tables that the server displays.

Restricting the List of Database Tables

You can restrict the list of database tables that the server displays by specifying one or more of the following:

- **Catalog.** You specify the name of the catalog whose tables you want to use.
  - If you are not working with a distributed database, do not restrict by catalog.
  - If you are working with a distributed database, you can specify the name of the database with which you want to work. If you are using DB2, use this field to specify the name of a DB2 location.
- **Schema pattern.** You specify the schemas whose tables you want to work with (if you are using DB2, use this field to specify an AuthID).
  - If you want tables for all schemas in the selected database, do not restrict by schema pattern.
  - If you want to restrict your search to tables by selected schemas, specify the schema name. You can specify a pattern-matching string if your JDBC supports it. Most drivers support the pattern-matching characters described for Table name pattern below; however, check your driver's documentation for information about its pattern-matching capabilities.
- **Table name pattern.** You specify the names of the tables with which you want to work. Specify a table name or a pattern-matching string that specifies the names of the tables. Most drivers support the following pattern-matching characters; however, check your driver's documentation for information about its pattern-matching capabilities.

Use	To match	For example
%	Any string of characters	HR% matches: HR30all, HR15mgmt, HR01payroll, and so forth.
_	A single character	HR3_mgmt matches: HR30mgmt, HR31mgmt, HR3Amgmt, and so forth.

- **Table type.** The type of the tables for which you want information. You can choose from the following common JDBC table types: Table, View, System Table, Global Temporary, Local Temporary, Alias, and Synonym.

Steps for Generating the Flow Service from a Table

Use the following procedure to generate a flow service from database tables.

To generate a flow service from a table

1. Open the Integration Server Administrator if it is not already open.
2. On the **Database** menu, select **Service Generation**.
3. In the **Source alias** list, select the database you want the service to access.
4. In the **Package** list, select the package in which you want the service to reside.
5. In the **Folder** field, type the name of the folder in which you want to save the service.
6. In the **Service** field, type the name you want to assign to the service.
7. In the **Execute ACL** list, select the ACL group to which you want to limit access to the service. Only the users who are members of the groups in the selected ACL will be able to invoke the service.
8. If you want to overwrite any existing services with the same name as the new service, next to **Overwrite Existing**, select the **Generated objects overwrite existing objects** check box.
9. Click **Generate from table**. The server displays the Specify Connection Parameters screen.
10. Determine which tables you want to search by performing one of the following:
    - To search all database tables, click **Connect**.
    - To restrict your search, fill in one or more fields in the **Restrictions** parameters as follows, and then click **Connect**. For more information about how to restrict the search, see "Restricting the List of Database Tables" on page 25.

For this parameter...	Specify
Catalog	Catalog names whose tables you want to use. If you are not working with a distributed database, do not restrict by catalog.
Schema Pattern	Schemas whose tables you want to work with. If you want tables for all schemas in the selected database, do not restrict by schema pattern.
Table Name Pattern	Table name or a pattern-matching string that identifies the names of the table with which you want to work. If the default wildcard character “%” is specified, then every table name will match.

For this parameter...	Specify
Table Types	Types of tables for which you want information.

After you click **Connect**, the server displays the Tables in DataSourceName screen.

11. In the **Select a table** section of the screen, click the table name that you want to use to generate the service.
12. In the **Service type** section of the screen, select the database operation that you want the service to perform (**Select**, **Insert**, **Delete**, or **Update**). The server displays the Columns in TableName screen.
13. In the **Columns** section of the screen, select the columns that you want to use as input parameters. When the service is executed, it expects input values for each of the columns you select. The service uses the input values as follows:

For this operation...	The service...
Select	Selects rows that have columns that match the input values.
Insert	Populates the columns of the inserted rows using the input values. If you want to populate all columns in the row, select all column names.
Delete	Selects rows that have the columns that match the input values and deletes the selected rows.
Update	Selects rows and updates them. To identify the criteria to use to select rows, you check column names in the **Criteria** column. The service expects input values for each column name you check. The service selects rows that have columns that match the input values. If you want to select all rows, do not check any **Criteria** columns. To identify the columns to update in the selected rows, use the **Set** column. The service expects input values for each column you check. The service sets the value of the checked columns (for all selected rows) to the specified input values.

14. Click **Generate SQL**. The server displays the Input Binding Generation screen.

    If you selected any columns for input parameters, the server displays a **Bind parameters** section of the screen. There is one **Parameter #** field for each column you selected.

15. Use the **Bind parameters** section of the screen to identify the name and data type for each input parameter. Specify the following information:

For this parameter...	Specify...
name	The name that you want the service to use for the input parameter. By default, the server uses the database column name. If you want to use a different name, type the new name in this field.
type	The data type of the input value. By default, the server uses the data type that is associated with the database column. If you want to use a different data type, select it from the list.

16. Click **Generate**.

Output from the Flow Service

When you generate a flow service from a database table, the database operation determines the service output. The following table describes the service output produced by each database operation:

Database Operation	Output
Select	Number of rows that the service retrieved and the columns from those rows that the service requested
Insert	Update count that the database returned in response to the SQL command
Delete	Update count that the database returned in response to the SQL command
Update	Update count that the database returned in response to the SQL command

Creating Database Services with Java or C/C++

You can code services that access databases in Java or C/C++. Code your own services if you need a service that performs more complex database operations than a flow service can provide.

To assist you in coding database services, you can use the built-in database services to perform basic database operations and test SQL statements before you add them to your services.

If you want your service to access a database that has nonstandard features (for example, data types that are not supported by SQL), use other database connection APIs. For example, you can make direct calls to JDBC or use other connection libraries, such as ADO.

Built-in Database Services

webMethods Integration Server provides several built-in database services that perform basic database operations. These services use JDBC to connect to the database to perform the specific database operation. You can invoke the built-in database services from:

- Java and C/C++ services
- Any flow service that you create in Designer by using the INVOKE flow step
- Clients

The *webMethods Integration Server Built-In Services Reference* contains a list of built-in database services and shows input and output information for each. Your service or client must invoke the built-in database service that opens a connection to a database before it can invoke any of the other built-in database services.

Testing SQL Statements

You can use the Integration Server Administrator to test an SQL statement that you want to execute using the built-in database service pub.db:execSQL or that you want to invoke directly using a JDBC call.

Before you can use the Integration Server Administrator to test an SQL statement, the server must have a configured database alias for the database that you want to access. For more information about how to configure a database alias, see " Identifying Database Aliases" on page 13.

To test an SQL statement

1. Open the Integration Server Administrator if it is not already open.
2. On the **Database** menu, select **Alias Management**.
3. Under **Current Data Sources**, select the database alias for the database you want the service to access.
4. Click **Connect**. The server displays the Connection Parameters screen.
5. Indicate which tables you want to search by performing one of the following:
   - To search all database tables, click **Connect**. The server displays the Tables screen.

- To restrict your search, fill in one or more fields in the **Restrictions** parameters as follows, and then click **Connect**. The server displays the Tables screen. For more information about how to restrict the search, see "Restricting the List of Database Tables" on page 25.

6. In the **Test SQL Queries** section of the Tables screen, type the SQL statement you want to test.
7. Click **Execute query**.

   The server sends the SQL statement you specified to the database for execution and displays a screen that lists the results from your SQL query.

Error Handling

Some database services return a *$dbMessage* output value which contains a text message that describes the results of the service. If the service results in an error, the service also returns standard error output values.

Creating Clients that Access Databases

You can access the databases from browser-based clients and clients coded in Java or C/ C++.

Invoking a Database Service from a Browser-based Client

Several of the database services can accept input from a browser-based client. (The descriptions of the database services in the *webMethods Integration Server Built-In Services Reference* indicate whether you can invoke a built-in database service from a browser-based client.) When a browser-based client submits input variables containing row information, these services automatically convert the row information into a nested IData object. The service determines which variables to convert based on the names of the variables. All variables except those whose names begin with _ or *$db* are converted into nested IData objects.

For example, suppose a browser user submits the following name-value pairs:

Variable Name	Contents
$dbTable	`Table Name`
Name	`Joe B`
Company	`Widgets, Inc.`

The pub.db:insert service converts the inputs as follows:

Variable Name	Contents	
$dbTable	Table Name	
$data	**Variable Name**	**Contents**
	Name	`Joe B`
	Company	`Widgets, Inc.`

Invoking a Built-in Service from a Java, C/C++, or VB Client

You can create client applications in Java or C/C++. The client applications can use the built-in database services to perform database operations. For information about these built-in services, see the *webMethods Integration Server Built-In Services Reference*.

If you want your client to access a database that has nonstandard features (for example, data types that are not supported by SQL), you use other database APIs. For example, you can make direct calls to JDBC or use other connection libraries, such as ADO.

Sample Code - IData

The following shows a sample Java client that accesses a database using an IData object to pass and receive data from the database service.

```
import com.wm.data.*;
import com.wm.app.b2b.client.Context;

public class DBClient
{
public static void main (String [] args) throws Exception
{
IData in = null;
IData out = null;
IData criteria = null;
IData set = null;

//
// connect to your integration server using an appropriate user
// name and password (doesn't have to be Administrator)
//
Context ctx = new Context();
ctx.connect ("localhost:5555", "Administrator", "manage");
//
// (1) request a DB connection by DB alias (if the DB
// changes location or something, we won't have to change
// this client code)
//
in = IDataFactory.create();
IDataCursor inCursor = in.getCursor();
inCursor.insertAfter ("$dbAlias", "Employees");
inCursor.destroy();
out = ctx.invoke ("wm.util.db", "connect", in);
//
// (2) update the Identification table to set Fonz's ID
// to 6500.  note that we couldn't do this from a Web
// browser because we couldn't build up the complex
// nested data structures
in = IDataFactory.create();
inCursor = in.getCursor();
inCursor.insertAfter ("$dbAlias", "Employees");
inCursor.insertAfter ("$dbTable", "Identification");
criteria = IDataFactory.create();
IDataCursor criteriaCursor = criteria.getCursor();
criteriaCursor.insertAfter ("name", "fonzie");
criteriaCursor.destroy();
inCursor.insertAfter ("$criteria", criteria);
set = IDataFactory.create();
IDataCursor setCursor = set.getCursor();
setCursor.insertAfter ("ID", "6500");
setCursor.destroy();
inCursor.insertAfter ("$set", set);
inCursor.destroy();

try {
out = ctx.invoke ("wm.util.db", "update", in);

//
// (3) look at the return values (updateCount is the
// most important in this case)
//
IDataCursor outCursor = out.getCursor();

if (outCursor.first("updateCount"))
{
int uc = Integer.parseInt ((String)outCursor.getValue());
System.err.println ("Update count: "+uc);
}
else
System.err.println ("Error: no update count returned");
outCursor.destroy();
} catch (Exception e) {
// maybe something went wrong with the DB access; we
// can get more information here
if (outCursor.first("$error")) {
   System.err.println ("Error: "+outCursor.getValue());
}
if (outCursor.first("$errorType")) {
   System.err.println ("Error type: "+outCursor.getValue());
}
outCursor.destroy();
}
}

}
```

A Determining Whether to Use the WmDB Package or the Adapter for JDBC

Overview

webMethods products provide two methods for accessing databases:

- WmDB package to use for prototyping, design, or temporary database access.
- webMethods Adapter for JDBC to use for all enterprise, mission-critical applications.

This appendix describes use cases to help you determine when to use the WmDB package and when to use the Adapter for JDBC.

Important: The WmDB package has been deprecated. When developing new applications, use the Adapter for JDBC instead of the WmDB package. The Adapter for JDBC provides more functionality, improved performance, and supports adapter notifications. For more information about the Adapter for JDBC, see the *webMethods Adapter for JDBC Installation and User's Guide*.

When to Use the WmDB Package

There are cases when it is more appropriate to use the WmDB package rather than the Adapter for JDBC. Use the WmDB package when:

- You need immediate results from the execution of an SQL query. The WmDB package allows you to run quick SQL queries directly from the Integration Server Administrator or DSP pages.
- You have a need for ad hoc introspection of database objects, such as tables, views, aliases, synonyms, or stored procedures.
- You want to customize management of the database connection pool.
- You have a limited number of tasks that need to access a database.
- You are required to integrate with a proprietary or a non-mainstream database, and you want to perform a quick, easy compatibility test before exploring an implementation that uses the Adapter for JDBC. The Adapter for JDBC supports all JDBC-compliant database implementations.
- You are developing a throw-away test scenario.

When to Use the webMethods Adapter for JDBC

You will want to use the Adapter for JDBC in most cases, including the following:

- Your system is likely to be expanded and modernized. This is typical for enterprise, mission-critical applications.

- You require better database performance. The performance of the Adapter for JDBC is superior to that of the WmDB package, particularly for batch operations. For example, to insert data into a database using the WmDB package, you use the pub.db:insert service; you supply the items to insert using a document list as input, and the pub.db:insert service processes each item in the document list sequentially without using JDBC standard facilities.
- You require or plan to use notification features.
- You need automatic and efficient database connection pool management.
- You require XA transactionality.
- You need to use JDBC standard data types, including BLOB and CLOB.
- You need to be able to configure services rather than hard code them.
- You need to use role-based security. The Adapter for JDBC allows the separation of the database administrator and development environments as needed. The WmDB package requires Administrator privileges.
- You want a configurable and structured user interface. Adapter for JDBC uses template-based configuration that is more structured and easier to use. The Adapter for JDBC uses metadata that helps protect the user's investment even if technology changes.
- You want to manage your database connection using webMethods Optimize for Infrastructure.

Index