CAPLIN

DS4DB 4.4

DataSource for Databases Administration Guide

October 2009

CONFIDENTIAL

Contents

1 Preface

1.1 What this document contains

This document describes how to install, configure and manage Caplin's DataSource for Databases product, version 4.4.

Note: In this document the DataSource for Databases product is also referred to by the acronym "DS4DB"

About Caplin document formats

This document is supplied in three formats:

- Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document navigate to the *HTMLDoc_m_n* folder and open the file *index.html*.
- Microsoft HTML Help (*.CHM* file), which is an HTML format contained in a single file.
  To read a *.CHM* file just open it – no web browser is needed.

For the best reading experience

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

Restrictions on viewing .CHM files

You can only read *.CHM* files from Microsoft Windows.

Microsoft Windows security restrictions may prevent you from viewing the content of *.CHM* files that are located on network drives. To fix this either copy the file to a local hard drive on your PC (for example the Desktop), or ask your System Administrator to grant access to the file across the network. For more information see the Microsoft knowledge base article at
http://support.microsoft.com/kb/896054/.

1.2 Who should read this document

This document is intended for System Administrators and Operators who need to install, configure and manage DataSource for Databases.

1.3 Related documents

- **DataSource for Databases Overview 4.4**

  Provides an overview of the Caplin DataSource for Databases product.
  You are strongly recommended to read the overview document before reading this Administration Guide.

- **DataSource for Databases API Reference**

  Defines the Java^{TM} APIs relating to DataSource for Databases.
  Explains how to implement custom result set processing.

- **DataSource Overview**

  Describes the DataSource concept and how it fits into the Caplin platform.

- **DataSource for Java API Reference**

  (Also known as **DataSource for Java SDK Documentation**.)

  Includes reference documentation for the DataSource for Java configuration files (*DataSource.xml* and *Fields.xml*) – see Configuration overview 12.

1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

Type	***Uses***
aMethod	Function or method name
aParameter	Parameter or variable name
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	Program output and code examples
The `value=10` attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
`Something typed in`	User input – things you type at the computer keyboard
XYZ Product Overview	Document name
◆	Information bullet point
■	Action bullet point – an action you should perform

Note: Important Notes are enclosed within a box like this.
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

Tip: Useful information is enclosed within a box like this.
Use these points to find out where to get more help on a topic.

1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Please email your feedback to documentation@caplin.com.

1.6 Acknowledgments

Adobe® Reader is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Sun, *Solaris*, *Java* and *JDBC* are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

MySQL is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Sybase and *Sybase SQL Server* are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

2 What is DataSource for Databases?

DataSource for Databases is a Caplin DataSource that provides a mechanism for moving data between a database and the Caplin Platform. It can read data from a database and forward it to other DataSource-enabled applications as structured records with fields; conversely it can accept data from other DataSource-enabled applications and persist it on the database. A typical database could be an existing repository of data, for example a database of financial market data.

DS4DB can interact with the Database on a timed basis, or when connections are made with other DataSources, or on receipt of DataSource messages.

As with other types of DataSource component, DS4DB can communicate with any other suitably configured DataSources, including Caplin's Liberator, Transformer and PriceMaster products, and any user defined DataSources that have been implemented using the DataSource SDK 101.

See also the **DataSource for Databases Overview**.

Tip: For an explanation of the DataSource concept refer to the document **DataSource Overview**.

Features of DataSource for Databases

- DataSource for Databases is built on the standard DataSource SDK, so it can communicate with a network of other DataSource peers.
- Written in Java™ and will run on any Java 1.5 enabled operating system platform. See Technical assumptions and restrictions 7 for the list of supported platforms.
- Will work with any database management systems that support the JDBC 101™ API.
- Can access data in multiple databases, with queries spread across them.
- Works with existing databases.
- Implements parallel processing for optimal throughput and responsiveness:
  - Simultaneous handling of multiple events.
  - Events can issue and respond to multiple database reads across multiple databases simultaneously.
  - Configuration settings control the degree of parallel processing.
- Includes a comprehensive XML-based configuration facility.
  - Allows you to implement fully functioned applications with no need to write code.
  - Configuration changes can be dynamically applied to a running system.
- Includes a Java API that allows you to implement customized result set processing (requires Java programming expertise).

3 What's new in this release?

Several new features have been introduced in this release of DataSource for Databases.

Parallel processing

The previous versions of DS4DB were single threaded, so that only one event could be processed at a time and within an event the database queries could only be executed sequentially against a single database. These restrictions have been removed in this release to provide significant performance improvements in throughput and responsiveness.

- **Parallel event processing**

  Multiple events can now execute in parallel. The degree of parallelism is controlled thorough the new `MaxEventParallelism` attribute of the `<DataSourceForDatabases>` XML configuration tag. See Parallel processing 61, and About the <DataSourceForDatabases> tag 32.

- **Access to multiple databases**

  Multiple databases can be accessed simultaneously, both within an event and across multiple events. See Parallel processing 61, and Accessing multiple databases 35.

- **Parallel database reads**

  Parallel database connections allow several read queries within an event to be executed concurrently, within one or more databases. This feature is controlled through the new `maxConnections` attribute of the `DatabaseConfiguration` XML configuration tag. See Parallel processing 61, and Configuring multiple connections to a database 36.

Dynamic configuration updates

Changes to DS4DB configuration can now be deployed dynamically. There is no longer any need to stop and restart DS4DB for configuration changes to take effect. See Dynamic configuration update 32.

Acknowledgement of Request Events

The acknowledgement behavior of Request Events can now be configured using the new `ackWhenComplete` attribute of the `<RequestEvent>` XML configuration tag. See How to define a Request Event 40.

sendName attribute in <CollapsingResultSetProcessor> tag

The `<CollapsingResultSetProcessor>` 75 tag now has a `sendName` attribute, which behaves in the same way as the `sendName` attribute in `<StandardResultSetProcessor>`. See Collapsing result set processing 57.

QueryRetryPeriod attribute in <DataSourceForDatabases> tag

The `<DataSourceForDatabases>` tag now has a `QueryRetryPeriod` attribute, which configures the time to wait before resubmitting queries when the DBMS has returned a deadlock error. See Deadlock Handling 34.

IMPORTANT – Existing XML configuration must be updated

In this release of DataSource for Databases the XML configuration has a new mandatory `dbRef` attribute, introduced to support access to multiple databases. This attribute is used in the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags.

Because it is mandatory, the `dbRef` attribute is not backwards compatible with the XML configuration used in previous releases of DS4DB. XML configuration files from previous releases will therefore not work with this release; DS4DB will report XML schema validation failures at start up and will refuse to load the file. See Missing dbRef attribute 70 in Troubleshooting 68.

Edit your old configuration files to include the new `dbRef` attribute in the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags. For how to do this, see sections Configuring access to databases 19 and The <DbRead> and <DbWrite> tags 44.

4 Getting Started

This section explains how to install DataSource for Databases and set up a simple configuration. It then tells you how to run the DS4DB application.

4.1 Technical assumptions and restrictions

Platforms and Java

DataSource for Databases is has been tested on, and is supported for, the following operating systems:

- Linux®
- Sun® Solaris™
- Microsoft® Windows® XP
- Microsoft Windows 2000
- Microsoft Windows 2003 Server

All these platforms must run the Java Runtime Environment (JRE™) or Java Development Kit (JDK™) – version 1.5 or later.

Supported DBMS

DataSource for Databases should work with any Database Management System that implements the Java Database Connectivity (JDBC) interface. It has been tested to work with the following DBMS:

- Sybase SQL Server®
- Microsoft SQL Server
- MySQL®

JDBC

DS4DB connects to the DBMS using Java Database Connectivity (JDBC) version 2.0. You will need to install a suitable JDBC technology-based driver that is compatible with JDBC version 2.0, so that DS4DB can successfully communicate with the DBMS. The DS4DB install kit contains JDBC libraries for Sybase SQL Server, Microsoft SQL Server and MySQL, but it is recommended that you obtain the latest version of the required driver from your DBMS vendor. Alternatively, refer to the list of JDBC drivers on the Sun Java web site page "JDBC Access API" at http://developers.sun.com/product/jdbc/drivers.

XML

The XML configuration schema conforms to XML version 1.0 and the XML schema version defined at http://www.w3.org/2001/XMLSchema.

Restrictions

1. DataSource for Databases does not provide an implicit database transaction environment. See Transaction environment 42.
2. DataSource for Databases does not currently support the use of stored procedures that return more than one result set.
3. There can only be one SQL query or stored procedure call in a the `query` attribute of the `<DbRead>` 84 configuration tag (see Defining database queries 42). If there are more then the behaviour is undefined; for example the queries might fail or only partially execute.
4. The JDBC interface defines a standard list of data types (SQLTYPEs) that may be used in SQL queries sent to the DBMS. DS4DB uses a subset of these data types; in particular it does not use a date/time data type, so dates and times must be dealt with as text strings.
   For the list of SQL data types that DS4DBsupports see the reference documentation for the <SQLParam> configuration tag 95.

4.2 Installing DataSource for Databases

Prerequisites

Before installing DataSource for Databases on your machine first make sure that it has a suitable environment set up, as follows:

- Ensure that there is a suitable version of the Java Runtime Environment (JRE) or Java Development Kit (JDK) installed; see Technical assumptions and restrictions 7.

Note: On Microsoft Windows platforms there is a Microsoft version of the Java Virtual Machine (JVM) built into some versions of Windows Internet Explorer®. However, DS4DB will not run correctly under the Microsoft JVM; you must install the JRE or JDK from Sun Microsystems.
See Technical assumptions and restrictions 7.

- Ensure there is a suitable JDBC technology-based driver installed, so that DS4DB can communicate with your DBMS. See JDBC 7 in Technical assumptions and restrictions 7, and Installing the JDBC driver 9.
- Check that your machine can access the DBMS.

Installing on Linux or Sun Solaris

The install kit is contained in a zip file called *DataSourceForDatabases-<version_number>.zip*

1. Copy the zip file to a base directory where you want the installed software to be located, such as */apps/caplin*
2. Unzip the file:

   ```
   unzip DataSourceForDatabases-<version_number>.zip
   ```

   The software will be unzipped into a new directory */DataSourceForDatabases-<version_number>* under your base directory, for example
   /apps/caplin/DataSourceForDatabases-4.4.0-2

Note: In the rest of this guide the directory where the DS4DB software is located is referred to as *$INSTALL_DIR*
For example, *$INSTALL_DIR* could refer to
/apps/caplin//DataSourceForDatabases

3. Install the JDBC driver – see Installing the JDBC driver 9.

Tip: When you have finished installing DataSource for Databases read the release note *$INSTALL_DIR/RELEASENOTE.txt* before proceeding any further.

Now read the configuration overview 12.

Installing on a Windows platform

The install kit is contained in a zip file called *DataSourceForDatabases-<version_number>.zip*

1. Copy the zip file to a base directory where you want the installed software to be located, such as *C:\Program Files\apps\Caplin*
2. Unzip the file using a suitable zip utility

The software will be unzipped into a new directory *\DataSourceForDatabases-<version_number>* under your base directory, for example
C:\Program Files\apps\Caplin\DataSourceForDatabases-4.4.0-2

Note: In the rest of this guide the directory where the DS4DB software is located is referred to as *$INSTALL_DIR* For example, *$INSTALL_DIR* could refer to
C:\Program Files\apps\Caplin\DataSourceForDatabases-4.4.0-2

3. Install the JDBC driver – see Installing the JDBC driver 9.

Tip: When you have finished installing DataSource for Databases read the release note *$INSTALL_DIR/RELEASENOTE.txt* before proceeding any further.

Now read the configuration overview 12.

Installing the JDBC driver

You will need to install a suitable JDBC technology-based driver that is compatible with JDBC version 2.0, so that DataSource for Databases can successfully communicate with the DBMS.

The install kit contains JDBC driver libraries for Sybase SQL Server, Microsoft SQL Server and MySQL.

Tip: You are recommended to obtain the latest version of the required JDBC driver from your DBMS vendor.
Alternatively you can refer to the Sun Java web site page containing a list of JDBC drivers: http://developers.sun.com/product/jdbc/drivers.

The JDBC drivers supplied with the install kit are located in *$INSTALL_DIR/lib*. If you are not using one of the supplied drivers, it is recommended that you copy the jar file containing your intended driver to *$INSTALL_DIR/lib*

Now edit the start up script to define the path to the required JDBC driver in the `CLASSPATH` environment variable:

Defining the JDBC CLASSPATH on Linux or Sun Solaris:

Edit the start up script *$INSTALL_DIR/run.sh*

If you are using one of the JDBC drivers provided in the install kit then remove the comment from the `CLASSPATH` definition that specifies the jar for the required JDBC driver:

```
# JAR for MySQL JDBC driver:
#export JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar

# JAR for Sybase JDBC driver:
# export JDBCLIBS=lib/jconn2d.jar:lib/jconn3d.jar:lib/jTDS2d.jar:
lib/jTDS3d.jar

# JAR for SQL server JDBC driver:
# export JDBCLIBS=lib/msbase.jar:lib/mssqlserver.jar:lib/msutil.jar

export CLASSPATH=$CLASSPATH:ds4db.jar:lib/datasrc.jar:
lib/xercesImpl.jar:$JDBCLIBS

java com.caplin.datasrc.ds4db.DataSourceForDatabases
```

In the above example, to use the JDBC driver for the MySQL DBMS product, remove the leading `#` from `#export JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar`

If you are not using one of the supplied drivers add an export statement defining the jar file for your intended driver.

Example:

```
# JAR for MySQL JDBC driver:
#export JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar

# JAR for Sybase JDBC driver:
# export JDBCLIBS=lib/jconn2d.jar:lib/jconn3d.jar:lib/jTDS2d.jar:
lib/jTDS3d.jar

# JAR for SQL server JDBC driver:
# export JDBCLIBS=lib/msbase.jar:lib/mssqlserver.jar:lib/msutil.jar

# JAR for my JDBC driver
export JDBCLIBS=lib/my-JDBC-driver-jar-name.jar

export CLASSPATH=$CLASSPATH:ds4db.jar:lib/datasrc.jar:
lib/xercesImpl.jar:$JDBCLIBS

java com.caplin.datasrc.ds4db.DataSourceForDatabases
```

Defining the JDBC CLASSPATH on Windows

Edit the start up batch file *$INSTALL_DIR/run.bat*

If you are using one of the JDBC drivers provided in the install kit then remove the comment from the `CLASSPATH` definition that specifies the jar for the required JDBC driver:

```
REM JAR for MySQL JDBC driver:
REM set JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar

REM JAR for Sybase JDBC driver:
REM set JDBCLIBS=lib/jconn2d.jar;lib/jconn3d.jar;lib/jTDS2d.jar;lib/jTDS3d.jar

REM JAR for SQL server JDBC driver:
REM set JDBCLIBS=lib/msbase.jar;lib/mssqlserver.jar;lib/msutil.jar

set CLASSPATH=%CLASSPATH%;ds4db.jar;lib/datasrc.jar;
lib/xercesImpl.jar;%JDBCLIBS%

java com.caplin.datasrc.ds4db.DataSourceForDatabases
```

In the above example, to use the JDBC driver for the MySQL DBMS product, remove the leading "`REM`" comment marker from
`REM set JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar`

If you are not using one of the supplied drivers add a statement defining the jar file for your intended driver.

Example:

```
REM JAR for MySQL JDBC driver:
REM set JDBCLIBS=lib/mysql-connector-java-5.0.4-bin.jar

REM JAR for Sybase JDBC driver:
REM set JDBCLIBS=lib/jconn2d.jar;lib/jconn3d.jar;lib/jTDS2d.jar;lib/jTDS3d.jar

REM JAR for SQL server JDBC driver:
REM set JDBCLIBS=lib/msbase.jar;lib/mssqlserver.jar;lib/msutil.jar

REM JAR for my JDBC driver
set JDBCLIBS=lib/my-JDBC-driver-jar-name.jar

set CLASSPATH=%CLASSPATH%;ds4db.jar;lib/datasrc.jar;
lib/xercesImpl.jar;%JDBCLIBS%

java com.caplin.datasrc.ds4db.DataSourceForDatabases
```

4.3 Configuration overview

DataSource for Databases uses three XML format configuration files:

- *DataSource.xml*
- *Fields.xml*
- *DataSourceForDatabases.xml*

These files are all located in *$INSTALL_DIR/conf*

Fields.xml and *DataSource.xml* are basic configuration files whose format is common to all DataSource applications built on the DataSource for Java SDK. *DataSourceForDatabases.xml* is unique to DS4DB.

DataSource.xml defines the overall characteristics and connectivity of the DataSource application; for example

- the name of the DataSource application,
- the name of its log file,
- the other DataSource peers to which DS4DB must try to connect,
- the other DataSource peers from which it will accept incoming connection requests.

Fields.xml defines the mapping between field names and field numbers as used by the DataSource communication protocol.

DataSourceForDatabases.xml defines the functionality of the DS4DB application. By specifying the relevant configuration items in *DataSourceForDatabases.xml* you determine the run time behaviour of DS4DB. The configuration allows you to specify:

- The databases, and their associated database management systems, to which the DS4DB application connects.
- The events that are triggered by incoming DataSource messages or the internal event timer.
- The database queries issued when processing each type of event.
- The processing carried out on result sets, and hence the formatting of the messages broadcast in response to events.
- How many events can execute at the same time, and how many concurrent queries can be run on each database.

For a description of how DS4DB processes events, database queries and result sets, see Defining events 39, Defining database queries 42 and Defining result set processing 53.

4.4 Setting up a simple configuration

The following sections explain how to set up a simple DataSource configuration for one instance of a DataSource for Databases application that communicates with one other DataSource peer, for example a Liberator server. This example DS4DB application is configured with a single timed event, so that at regular intervals it reads a simple database table and broadcasts any changed data to the connected peer.

The database table is called **equity.** It has three columns called **stock**, **ask_price** and **bid_price**, as shown here.

Database table equity

Column Name:	**stock**	**ask_price**	**bid_price**
Data type:	varchar(20)	float	float
Row 1	ABC	52.032	51.160
Row 2	XYZ	11.650	10.904

The *DataSourceForDatabases.xml* configuration file supplied in the installation kit defines the configuration for this application, see Configuring a timed event 21.

Note: If you want to try this example you will need to define and populate the **equity** table on a suitable database.

Setting up DataSource.xml

In this simple configuration the DataSource for Database application initiates the connection to the remote DataSource, but you could also set up the configuration so that either DataSource initiates the connection.

You will also need to set up the DataSource configuration of the remote DataSource application, so that it can talk to your DS4DB application.

Setting up the DataSource configuration for DS4DB

Set up the *DataSource.xml* configuration for DS4DB as follows.

```
<?xml version="1.0" encoding="utf-8"?>
<dataSource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            id="2"
            appName="DataSourceForDatabases">
   <logging default="true"
            defaultFileName="logs/DataSourceForDatabases.log"
            level="FINE" />
   <packetLog append="true"
              pattern="logs/packet-DataSourceForDatabases.log" />
   <peer>
      <remote name="LibDS4DBTest" type="CONTRIB" />
      <destination address="localhost" port="25000" />
   </peer>
</dataSource>
```

The tags and attributes have the following meanings.

- `<dataSource>` tag

  `id="2"` This is the ID number of the DataSource for Databases application. It must be unique in the network of DataSource peers. Make sure that you change the *DataSource.xml* configuration for each DataSource that connects to your application so that the `<remote>` tag identifies DS4DB by the same id – see Setting up the DataSource configuration for a remote Java DataSource 16 (or for non-Java DataSources specify the correct id in the *.conf* configuration file – see Setting up the DataSource configuration for a remote Liberator 17).

  `appName="DataSourceForDatabases"` This is the name of this DataSource; it defines how DataSource peers will identify it. You can change it to some other name that more accurately reflects your DS4DB application.

- `<logging>` tag

  `defaultFileName="logs/DataSourceForDatabases.log"` This is the name of the log file. You might want to change the name of the log file to more accurately reflect the name of the DS4DB application.

  `level="FINE"` This will produce a detailed log file. If you want less detail in the log file change this to `"INFO"`.

- `<packetLog>` tag

  This tag defines the packet log, which is where DataSource for Database application records binary messages sent between it and its peers. If you do not want DS4DB to record these messages then add the attribute `noPacketLog="true"`. It may be better to leave the packet log on initially, so you can look at the message activity when you run the application.

  `pattern="logs/packet-DataSourceForDatabases.log"` defines the directory path and name of the packet log. You might want to change the name of the log file to more accurately reflect the name of the DS4DB application.

- `<peer>` tag

  The `<peer>` tag defines the characteristics of the connection between the DS4DB application and the remote DataSource with which it will communicate.

- `<remote>` tag

  This tag defines how the remote DataSource appears to the DS4DB application when a connection is established.

  `name="LibDS4DBTest"` defines the name of the remote DataSource; in this case it is assumed to be a Liberator (see the `datasrc-name` attribute in Setting up the DataSource configuration for a remote Liberator 17).

  `type="CONTRIB"` indicates that the remote DataSource will accept Update messages from the DS4DB application.

- `<destination>` tag

  This tag defines how to establish a connection with the remote DataSource application

  The `address` attribute is the connection address of the remote DataSource application. The value `"localhost"` means the remote DataSource application is assumed to be on the same machine as the DS4DB application.

  The `port` attribute is the connection port number. Port 25000 is the standard port number for a Caplin Liberator. In this simple example the DS4DB initiates the connection to the Liberator. If your remote DataSource is a different application, change the port attribute accordingly.

Tip: For a full definition of all the XML tags in the DataSource XML configuration schema and what they are used for see the **DataSource for Java API Reference**.

Setting up the DataSource configuration for a remote Java DataSource

If the remote application is another Java based DataSource then you must produce another version of *DataSource.xml* located in the *$INSTALL_DIR/conf* directory of the remote application:

```
<?xml version="1.0" encoding="utf-8"?>
<dataSource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            appName="MyJavaDataSource"
            port="25000">
  <logging default="true"
           defaultFileName="logs/MyJavaDataSource.log"
           level="FINE" />
  <packetLog append="true"
             pattern="logs/packet-MyJavaDataSource.log" />
  <peer>
    <remote name="DataSourceForDatabases" id="2" />
  </peer>
</dataSource>
```

The essential points about this configuration are:

- `<dataSource>` tag

  `appName="MyJavaDataSource"` This is the name of this DataSource application.

  `port="25000"` This is the network port on which the DataSource application listens for connections from peers. The `port` attribute of the `<destination>` tag in the DS4DB *DataSource.xml* must match this value, so the DS4DB knows how to connect to the Liberator. The MyJavaDataSource application will listen on this port for incoming connections. In this case the port is the standard port for a Caplin Liberator, so as to match the value in the example DS4DB DataSource configuration. You may want to change the value to something more appropriate and alter the DS4DB configuration accordingly.

- `<peer>` tag

  The `<peer>` tag defines the characteristics of the connection between the DataSource peer and DS4DB.

- `<remote>` tag

  This tag defines how the DS4DB application appears to the MyJavaDataSource DataSource application when a connection is established.

  `name="DataSourceForDatabases"` defines the name of the remote DS4DB application. This is the name specified in the *DataSource.xml* for DS4B (`appName` attribute of the `<dataSource>` tag).

  `id="2"` is the id of the DS4DB application. This is the id specified in the *DataSource.xml* for DS4B (`id` attribute of the `<dataSource>` tag)

  Note that, unlike the *DataSource.xml* for DS4B, there is no `<destination>` tag, because the MyJavaDataSource application does not initiate the connection to the DS4DB peer. However, in a more sophisticated implementation other DataSources may need to initiate the connection to the DS4DB peer and you would therefore include a `<destination>` tag.

Setting up the DataSource configuration for a remote Liberator

If the remote application is not a Java based DataSource, for example it is a Caplin Liberator or Transformer, then the DataSource configuration file has a different format. The following example shows how to configure a Liberator to connect to the DS4DB application. The configuration file is called *rttpd.conf* and is located in the */etc* directory of the Liberator installation. Only the relevant portions of the file are shown here.

```
# Caplin Liberator configuration file
#
#
datasrc-name           LibDS4DBTest
...

## DATASRC ###################################################
#
#

datasrc-port                             25000

add-peer
        remote-id                         2
        remote-name                       DataSourceForDatabases
        remote-type                       2
        label                             DataSourceForDatabases
end-peer

add-data-service

        service-name            DS4DBDemo
        include-pattern         ^/

        add-source-group
                required        true
                add-priority
                        label DataSourceForDatabases
                end-priority
        end-source-group

end-data-service
...
```

The essential points about this configuration are:

- The Liberator's DataSource name is "`LibDS4DBTest`".
- The Liberator's `datasrc-port` is `25000`, which is the same as the `<port>` attribute of the `<destination>` tag in the DS4DB *DataSource.xml*, so the DS4DB application knows how to connect to the Liberator. The Liberator will listen on this port for incoming connections.
- The `remote-id` attribute with value `2` in the `add-peer` section is the same as the id attribute of the `<local>` tag in the *DataSource.xml* of the DS4DB application, so the Liberator knows how to connect to the DS4DB.
- The `remote-name` attribute "`DataSourceForDatabases`" matches the `appName` attribute of the `<dataSource>` tag in the *DataSource.xml* of the DS4DB application.
- The `remote-type` attribute of `2` ("contrib") indicates that the Liberator expects to receive Update messages ("contributions") from the DS4DB application.

- The `label` in the `add-source-group` of the `add-data-service` section must match the `label` in the add-peer section.

Setting up the message fields in Fields.xml

You must set up the mapping of DataSource message field numbers to field names in the *Fields.xml* configuration file. The DataSource protocol identifies message fields internally by number. DataSource enabled applications can be written to just refer to fields by their numbers, but DataSource for Databases refers to fields by name, so it needs a *Fields.xml* file.

The *Fields.xml* file in the DS4DB install kit includes the following entries which are used in the example DS4DB application:

```
<?xml version="1.0" encoding="utf-8"?>
<fieldManager xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
<!-- used by the 'DataSource for Databases' simple application -->

  <field name="stock" number="30001" />
  <field name="ask_price" number="30002" />
  <field name="bid_price" number="30003" />

</fieldManager>
```

The field names `stock`, `ask_price` and `bid_price`, are the names of the database fields in the **equity** table whose values are broadcast when the timed event is triggered; see Configuring a timed event 21.

Note: Each field number must be unique across all the DataSource applications in your DataSource network.

Tip: The reference information for the *Fields.xml* file is in the **DataSource for Java API Reference**.

If the DataSource peer that connects to DS4DB is another DataSource for Java application, then make sure that its *Fields.xml* file contains these same definitions. If the DataSource peer is not a DataSource for Java application (if it is a Caplin Liberator for example), then you should edit the *fields-datasourcefordatabases.conf* file instead and include it in the main *fields.conf* file. See More about basic configuration 30.

For the example DS4DB application, the field definitions in *fields-datasourcefordatabases.conf* would be:

```
#fields used by simple example in DS4DB
add-field       stock                   30001    0
add-field       ask_price               30002    0
add-field       bid_price               30003    0
```

Configuring access to databases

DataSource for Databases needs to know how to access your databases. This is defined in the `<DatabaseConfiguration>` tags of the DS4DB XML configuration file *DataSourceForDatabases.xml*. When DS4DB starts up it uses the information in these tags to automatically connect to defined databases.

Edit the example copy of the DS4DB XML configuration file
DataSourceForDatabases.xml located in *$INSTALL_DIR/conf*.
This configuration file contains skeletons for accessing a single database via the Sybase SQL Server, Microsoft SQL Server, or MySQL DBMS:

```
<?xml version="1.0" encoding="utf-8"?>
<DataSourceForDatabases>

<!-- Simple configuration that implements a timed event -->

<!-- Choose one of the following database connections, or implement your own
     Edit the connectionURL attribute to set up the connection to your database
-->

<!-- Sybase SQL Server connection
   <DatabaseConfiguration
     dbRef="DB1"
     driver="com.sybase.jdbc2.jdbc.SybDriver"
     connectionURL=
"jdbc:sybase:Tds:localhost:5000/dbname?user=uname&amp;charset=cp1250" />
-->

<!-- Microsoft SQL Server connection
  <DatabaseConfiguration
    dbRef="DB1"
    driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
    connectionURL=
"jdbc:microsoft:localhost://localhost:1433;User=uname;Password=password;
DatabaseName=dbname" />
-->

<!-- MYSQL connection
   <DatabaseConfiguration
     dbRef="DB1"
     driver="com.mysql.jdbc.Driver"
     connectionURL=
"jdbc:mysql://localhost/dbname?user=uname&amp;password=password" />
-->
...
```

If you are using one of these DBMS then:

- Remove the comment surrounding the relevant `<DatabaseConfiguration />` tag.
- Edit the `connectionURL` attribute to define the URL needed to connect to your DBMS. If the DBMS server is not on the same machine as the DS4DB application then you will need to replace `localhost` with the relevant domain name, for example `abc.com`.
  Replace `dbname` with the name of the database, for example `ds4dbtestdb`.
  The URL may also need to include a username, and possibly a password, so that the database server will allow the application to communicate with it. In the skeleton above these are defined as `uname` and `password` respectively.

The `dbRef` attribute of the `<DatabaseConfiguration>` tag gives the database a name that is referred to by subsequent `<DbRead>` and `<DbWrite>` tags defining the queries that can be sent to the database. Each database must have a `dbRef` value that is unique within the DS4DB configuration.

Example:

```
<DatabaseConfiguration dbRef="DB1" driver="com.mysql.jdbc.Driver"
  connectionURL="jdbc:mysql://abc.com/ds4dbtestdb?user=alice" />
```

This example specifies the JDBC driver for the MySQL database server with:

- a connection URL `abc.com`,
- for the database called `ds4dbtestdb`, referred to in the rest of the configuration as "`DB1`",
- the user "`alice`",
- no password.

If you are using some other DBMS then define a new

`<DatabaseConfiguration />` tag:

```
<?xml version="1.0" encoding="utf-8"?>
<DataSourceForDatabases>

<!-- My DBMS Server connection -->
   <DatabaseConfiguration
     dbRef="MY_DB_REF"
     driver="DRIVER_DATASOURCE_FOR_DATABASES"
     connectionURL="CONNECTIONURL_DATASOURCE_FOR_DATABASES" />
   ...

</DataSourceForDatabases>
```

- Replace `DRIVER_DATASOURCE_FOR_DATABASES` with the specification of the JDBC technology-based driver required to access your DBMS. This must be the Java package name of the driver in the jar that you specified when you installed the JDBC driver – see Installing the JDBC driver 9.
- Replace `CONNECTIONURL_DATASOURCE_FOR_DATABASES` with the actual URL needed to make a connection to your DBMS. The URL may need to include a username and possibly a password so that the database server will allow the application to communicate with it.

Tip: The database administrator within your organization who manages your DBMS should be able to supply this information.

See also Accessing multiple databases 35, Configuring multiple connections to a database 36, and JDBC 7 in Technical assumptions and restrictions 7.

Configuring a timed event

The *DataSourceForDatabases.xml* configuration file supplied in the installation kit defines the configuration of the example DS4DB application (see Setting up a simple configuration 13). The configuration defines a timed event that reads information from a simple relational database table called **equity**. For convenience the database table is shown again here:

Database table equity

Column Name:	**stock**	**ask_price**	**bid_price**
Data type:	varchar(20)	float	float
Row 1	ABC	52.032	51.160
Row 2	XYZ	11.650	10.904

Note: If you want to try this example you will need to define and populate the **equity** table on a suitable database.
Also make sure the configuration correctly specifies how to access the database; see Configuring access to databases 19.

The example DS4DB configuration is defined as follows.

<DbRead> tag

The SQL query that reads the data from the database query is defined using a `<DbRead>` tag:

```
<DbRead id="GET_PRICES"
        dbRef="DB1"
        prefix="/EQUITY/"
        query="select stock, ask_price, bid_price from equity" >
  <StandardResultSetProcessor nameColumn="stock" sendName="true" />
</DbRead>
```

- The `id` attribute gives the query a name that other configuration tags can refer to.
- The `dbRef` attribute defines the database ("`DB1`") to which the query is to be sent. This should be value of the `dbRef` attribute in the `<DatabaseConfiguration>` tag defining the required database.
- The `prefix` attribute is a text string that will be added to the front (left hand end) of the subject (symbol) name in the DataSource Update message generated from the selected data. In this case the prefix is "`/EQUITY/`".
- The SQL query is
  `"select stock, ask_price, bid_price from equity".`
  You may need to adjust the syntax of this query to suit the requirements of your particular DBMS.

<StandardResultSetProcessor> tag

The type of result set processing to be carried out on the retrieved data is defined by a `<StandardResultSetProcessor>` tag:

```
<DbRead id="GET_PRICES"
        prefix="/EQUITY/"
        dbRef="DB1"
        query="select stock, ask_price, bid_price from equity" >
  <StandardResultSetProcessor nameColumn="stock" sendName="true" />
</DbRead>
```

The `<StandardResultSetProcessor>` tag is a child of the `<DbRead>` tag. This type of result set processor produces one DataSource Update message for each row returned in the result set (see Standard result set processing 54).

The `nameColumn` attribute defines the column of the result set that DS4DB will use to obtain the subject name for each generated Update message. In this case the subject name is the contents of the column called **stock**.

<TimedEvent> tag

The timed event that will trigger the database read is defined by a `<TimedEvent>` tag:

```
<DbRead id="GET_PRICES"
        prefix="/EQUITY/"
        dbRef="DB1"
        query="select stock, ask_price, bid_price from equity" >
  <StandardResultSetProcessor nameColumn="stock" sendName="true" />
</DbRead>

<TimedEvent refreshPeriod="5000">
  <DbReadRef ref="GET_PRICES" />
</TimedEvent>
```

The `refreshPeriod` attribute is the interval between successive timed events; in this case the interval is 5000 milliseconds, so DS4DB will read the database every 5 seconds.

The child tag `<DbReadRef>` specifies that when the event is triggered, DS4DB must run the **GET_PRICES** query, as defined in the `<DbRead>` whose `id` is `"GET_PRICES"`.

<PeerConnectionEvent> tag

There is also a `<PeerConnectionEvent>` tag:

```
<PeerConnectionEvent>
  <DbReadRef ref="GET_PRICES"/>
</PeerConnectionEvent>
```

The Peer Connection Event event causes the DataSource for Databases application to publish the database data when a connection to the remote DataSource is established. The event fires each time a connection is established, including the situation when an existing connection has been broken and is subsequently re-established.

The generated message

When you run DS4DB using this configuration and the database data shown above, DS4DB will read the **equity** database table when it connects to the remote DataSource (for example, a Liberator). As a result it will send the following update message to the remote DataSource:

Message no.	Subject (Symbol)	Fields
1	/EQUITY/ABC	ask_price=52.032, bid_price=51.160
2	/EQUITY/XYZ	ask_price=11.650, bid_price=10.904

Thereafter DS4DB will run every 5 seconds, but it will only send further messages when an **ask_price** or **bid_price** changes. You can test this by manually modifying the values in the database (see Monitoring the simple timed event application 24).

Running the simple timed event application

- Start the other DataSource (for example a Caplin Liberator).
- Then start the DataSource for Databases application - see Starting DataSource for Databases 27.

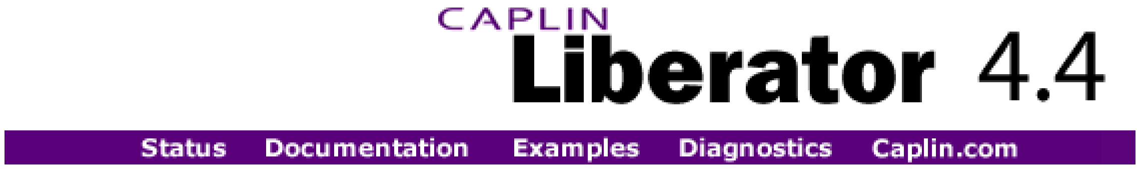
Monitoring the simple timed event application

If the remote DataSource is a Liberator then you can use the Object Browsing Tool to examine the output of the timed event DS4DB application:

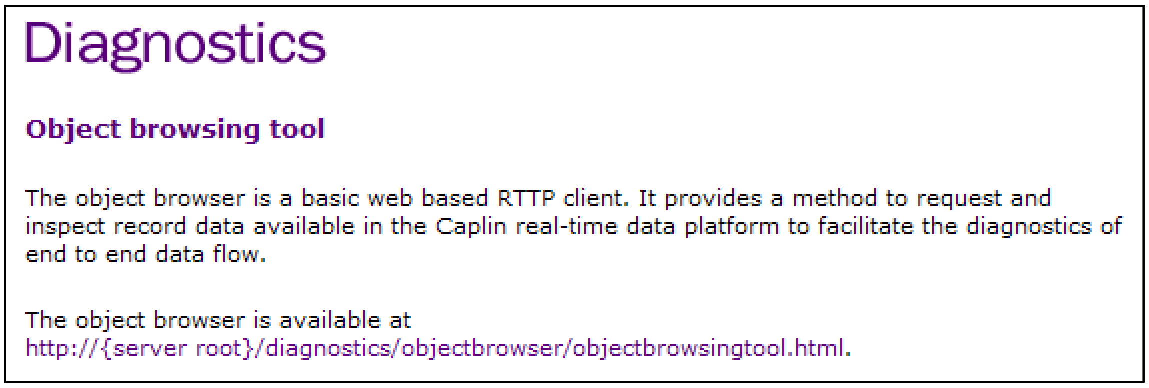
1. Open a web browser and enter the home page address of the Liberator's web site in the address bar of the browser:

   `http://name-of-liberator-host :8080`

2. When the home page is displayed, click on the link called Diagnostics:



3. When the diagnostics page is displayed click on the Object Browser link ("The object browser is available at http://{server root}/diagnostics/objectbrowser/objectbrowsingtool.html"):



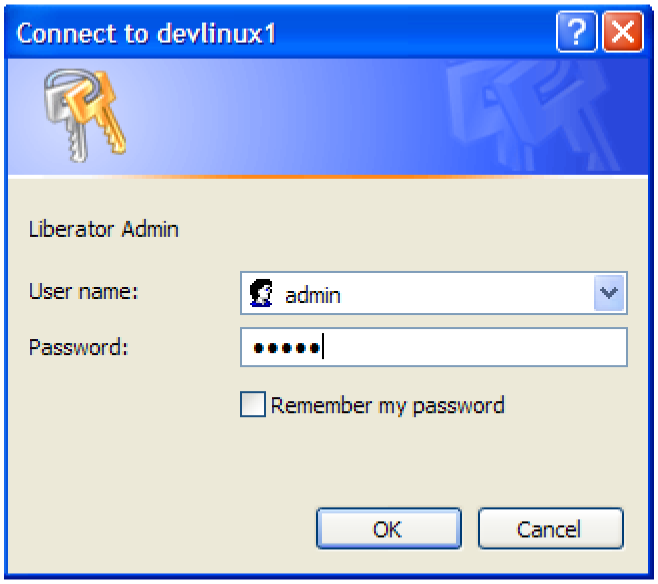
Diagnostics

Object browsing tool

The object browser is a basic web based RTTP client. It provides a method to request and inspect record data available in the Caplin real-time data platform to facilitate the diagnostics of end to end data flow.

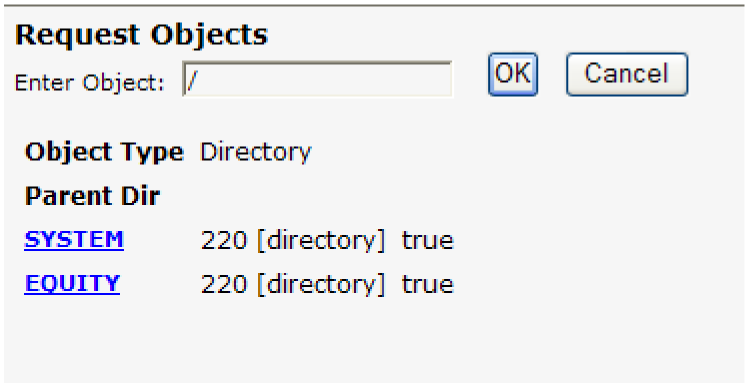
The object browser is available at http://{server root}/diagnostics/objectbrowser/objectbrowsingtool.html.

4. Enter a user name and password when you are prompted for them. The default user name and password configured in the standard Liberator installation kit are `admin` and `admin`, but they may be different for your particular Liberator installation:



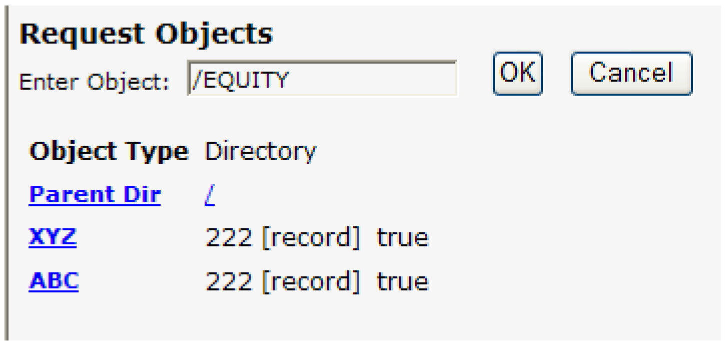
The object browser tool will open in a new web browser window.

5. Key / in the Enter Object box, click on the OK button and the tool will display the root symbols for the available objects (symbols). If the DataSource for Databases application is working correctly, and is connected to the Liberator, the tool will display the directory object **EQUITY** as a link:



Object Type	Directory	
Parent Dir		
SYSTEM	220 [directory]	true
EQUITY	220 [directory]	true

6. Click on the EQUITY link and the Object Browser will display the two symbols **XYZ** and **ABC**; these are the symbols that the DataSource for Databases application has published from the **equity** database table:

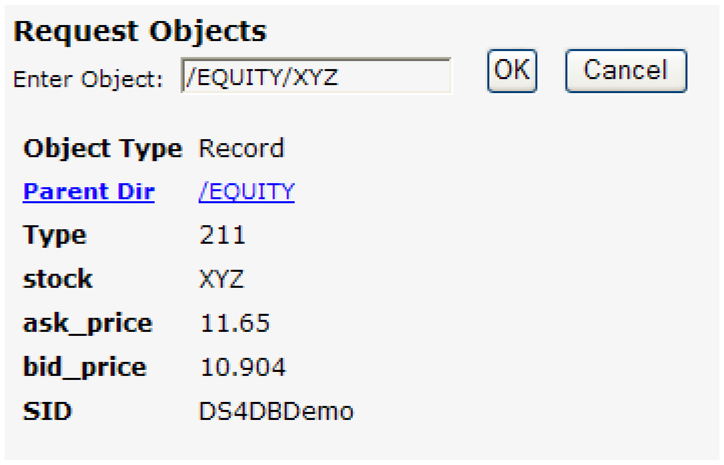


Request Objects

Enter Object: /EQUITY OK Cancel

Object Type	Directory	
Parent Dir	/	
XYZ	222 [record]	true
ABC	222 [record]	true

7. Click on the **XYZ** symbol to see the values of the symbol:



Request Objects

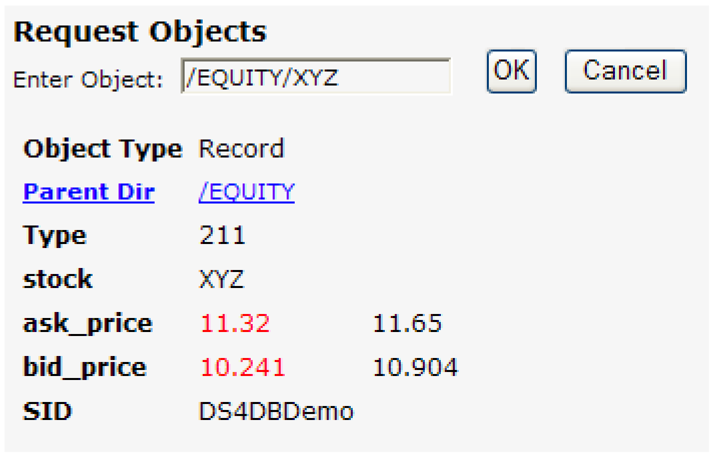
Enter Object: /EQUITY/XYZ OK Cancel

Object Type	Record
Parent Dir	/EQUITY
Type	211
stock	XYZ
ask_price	11.65
bid_price	10.904
SID	DS4DBDemo

8. Use the SQL utility supplied with your DBMS to modify the values of one or both symbols on the database, for example:

```
update equity set ask_price=11.32, bid_price=10.241 where stock="XYZ";
```

The changes will be published within five seconds and the updates will appear in the Object Browser:



Request Objects

Enter Object: /EQUITY/XYZ OK Cancel

Object Type	Record	
Parent Dir	/EQUITY	
Type	211	
stock	XYZ	
ask_price	11.32	11.65
bid_price	10.241	10.904
SID	DS4DBDemo	

Tip: You can watch DataSource for Databases in action by looking at the event log in *$INSTALL_DIR/logs/DataSourceForDatabases.log*.

In the `/logs` directory enter the command:
```
tail -f DataSourceForDatabases.log
```

4.5 Running DataSource for Databases

Starting DataSource for Databases

On Linux or Sun Solaris you can start DataSource for Databases as a foreground process or as a background process. On Windows you can only start DS4DB as a foreground process.

Starting DS4DB on Linux or Sun Solaris

To run DataSource for Databases as a foreground process:

1. Change the current directory to *$INSTALL_DIR*
2. Run the shell script *run.sh*:

```
./run.sh
```

To run DataSource for Databases as a background process:

1. Change the current directory to *$INSTALL_DIR*
2. Start the shell script *run.sh* as a background process:

```
./run.sh &
```

Here is typical output from the run command:

```
-bash-3.00$ DataSource for Databases starting...
Product         : DataSource for Databases
Version         : 4.3.1a
Build Date      : 21-Nov-2006
Build Time      : 17:45
Build Number    : 55097
Copyright       : Copyright 1995-2006 Caplin Systems Ltd
Using DataSourceForDatabases Schema directory 'file:./conf/'

Using DataSourceForDatabases Configuration File './conf/DataSourceForDatabases.xml'
```

Stopping DataSource for Databases

The procedure for stopping DataSource for Databases depends on whether the application is running on Linux/sun Solaris or on Windows.

Stopping DS4DB on Linux or Sun Solaris

If DataSource for Databases is running as a foreground process:

- Type `Ctrl/C` to interrupt the process.

If DataSource for Databases is running as a background process:

- Issue the following command to the find the PID of the DS4DB process:

  ```
  ps -ef | grep com.caplin.datasrc.ds4db.DataSourceForDatabases
  ```

Example (PID highlighted):

```
-bash-3.00$ ps -ef | grep com.caplin.datasrc.ds4db.DataSourceForDatabases
          27600  0 10:31 pts/14   00:00:01
java com.caplin.datasrc.ds4db.DataSourceForDatabases
    28793  9383  0 10:36 pts/14   00:00:00
grep com.caplin.datasrc.ds4db.DataSourceForDatabases
```

- Kill the process using the `kill` or `kill -9` command.
  For example:

  ```
  kill 27601
  ```

Stopping DS4DB on Windows

In the command window in which DataSource for Databases is running:

- Type `Ctrl/C` to interrupt the program

  The command window displays the prompt:

```
Terminate batch job (Y/N)?
```

- Type `y<return>` or `Y<return>`

  DS4DB will stop running.

Starting DS4DB on Windows

In a command window:

1. Change the current directory to *$INSTALL_DIR*
2. Run the batch file *run.bat*:

   ```
   run.bat
   ```

   DataSource for Databases will run as a foreground process within the command window.

Here is typical output from the run command:

```
C:>cd C:\Program Files\apps\Caplin\DataSourceForDatabases-4.3.0-2

C:\Program Files\apps\Caplin\DataSourceForDatabases-4.3.0-2>run.bat

C:\Program Files\apps\Caplin\DataSourceForDatabases-4.3.0-2>java
-jar DataSourceForDatabases.jar
DataSource for Databases starting...
Product        : DataSource for Databases
Version        : 4.3.1a
Build Date     : 21-Nov-2006
Build Time     : 17:45
Build Number   : 55097
Copyright      : Copyright 1995-2006 Caplin Systems Ltd
Using DataSourceForDatabases Schema directory 'file:./conf/'

Using DataSourceForDatabases Configuration File './conf/DataSourceForDatabases.xml'
```

5 More about basic configuration

Configuring the DataSource characteristics (DataSource.xml)

You must configure your DataSource for Databases application so that it can operate correctly as a Caplin DataSource within a network of other DataSource enabled applications and Caplin products. This configuration is defined in XML format in the XML file *DataSource.xml*.

The characteristics you define include:

- How other DataSource peers will identify and connect to your DS4DB application (`<dataSource>` tag).
- How your DS4DB application communicates with its DataSource peers (`<peer>` tag and its elements).
- How your DS4DB application logs internal status messages and error messages `<logging>` tag)
- How your DS4DB application records binary messages sent between it and its peers (if required). (`<packetLog>` tag

The configuration you define is validated at run time against an XML schema.

For a full definition of all the XML tags in the DataSource XML configuration schema, and what they are used for, see the **DataSource for Java API Reference**.

Configuring DataSource message field numbers (Fields.xml)

The DataSource protocol identifies fields in message by field numbers, but the DS4DB application refers to these fields by name. The *Fields.xml* configuration file defines the mapping between field names and field numbers. You must define a name—number mapping for every field in every message type that can be sent to or from DS4DB.

Example *Fields.xml* configuration file:

```
<fieldManager xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
   <field name="stock" number="1001" />
   <field name="ask_price" number="1002" />
   <field name="bid_price" number="1003" />
</fieldManager>
```

Tip: You don't need to define field name—number mappings for database fields whose names are not used in a DataSource message. For example, a database field may be called 'AFX1' but DS4DB may change the name of this field to 'FX1' in messages it generates. In this case, although the field name 'FX1' must be assigned a field number in *Fields.xml*, 'AFX1' does not need to be assigned a field number.

For a full definition of all the XML tags in the FieldManager XML configuration schema, and how they are used, see the **DataSource for Java API Reference**.

More on configuring DataSource message field numbers (fields-datasourcefordatabases.conf)

There is also a companion fields definition file called *fields-datasourcefordatabases.conf*. If your DS4DB application sends messages to DataSource peers that are not Java applications (such as Caplin Liberator, Caplin Transformer or Caplin PriceMaster), then you should edit *fields-datasourcefordatabases.conf* so that it contains the same field name—number mappings as *Fields.xml*. The *fields-datasourcefordatabases.conf* file can then be read by the non-Java peers, so these peers will assign the correct field names to the field numbers in messages received from your DS4DB application. You normally include *fields-datasourcefordatabases.conf* in the DataSource application's standard fields configuration file *fields.conf*, using the `include-file` construct.

Example *fields-datasourcefordatabases.conf* **configuration file:**

```
add-field       stock                   1001    0
add-field       ask_price               1002    0
add-field       bid_price               1003    0
```

The format of each entry in this file is:

`add-field FieldName FieldNumber FieldFlags`

`FieldName` is the name of the field.

`FieldNumber` is number of the field.

`FieldFlags` defines flags used by Caplin Liberator and is always set to `0` in other DataSource applications.

6 About the <DataSourceForDatabases> tag

The root tag of the *DataSourceForDatabases.xml* configuration file is `<DataSourceForDatabases>` 82
:

```
<DataSourceForDatabases>

   <!-- Child tags here -->

</DataSourceForDatabases>
```

All the attributes of the `<DataSourceForDatabases>` tag are optional, with default values. These attributes control certain aspects of the run-time behaviour of your DS4DB application. They are:

- `ConfigScanInterval` – controls the dynamic configuration update 32 facility.
- `EnableCache` – controls query caching behaviour 33.
- `MaxEventParallelism` – configures parallel event processing 33.
- `MaxQueryRetries` and `QueryRetryPeriod` – control deadlock handling 34.
- `PM4Output` – controls PriceMaster compatibility 34.

6.1 Dynamic configuration update

When you have made changes to the configuration of DS4DB in the *DataSourceForDatabases.xml* file, you do not need to stop and restart your installation of DS4DB for the changes to take effect; they can be deployed dynamically while DS4DB is executing.

This facility is controlled through the `ConfigScanInterval` attribute of the `<DataSourceForDatabases>` 82 configuration tag, which defines how often DS4DB should scan its configuration file for changes. The default value of `ConfigScanInterval` is 0, which means that dynamic configuration update is turned off.

To turn on dynamic configuration updating, set `ConfigScanInterval` to a positive value; this is the time interval in seconds at which DS4DB will check for configuration changes. DS4DB examines the last-modified time of the configuration file *$INSTALL_DIR/conf/DataSourceForDatabases.xml* at the defined intervals. If the last-modified time has altered, it reloads the configuration from the file as soon as it can (when there are no outstanding events to process).

DS4DB generates DataSource alert messages to record the status of dynamic configuration updates – see Alerts raised by DS4DB 99

Tip: You can force DS4DB to reload its configuration just by touching the configuration file so that the file modification date changes.

See also the impact of dynamic configuration update on query caching behavior 33.

6.2 Query caching behavior

By default DS4DB caches the results of read queries made on the database, so it will only send an Update message if the data returned by the read query has actually changed. The message contains just the fields that have changed.

You can turn off the cache by specifying the `<DataSourceForDatabases>` attribute `EnableCache="false"`; DS4DB will then *always* generate one or more Update messages for each successful query.

Note: It is recommended that you turn off the cache if your DS4DB application communicates with more than one DataSource peer.

The cache is only really designed for use with one DataSource peer. When there are multiple peers connected to DS4DB and caching is enabled, newly connecting peers may not receive all the data they require. This is because, assuming you have defined a Peer Connection Event that broadcasts data, when any peer connects other than the first one DS4DB responds by performing a database query. However, it only broadcasts the retrieved data that has changed.
Conversely, if the cache is turned off then a newly connecting peer will receive *all* the data retrieved by the database query associated with the Peer Connection Event.

Impact of query caching on dynamic configuration updates

When configuration is dynamically reloaded, DS4DB transfers any previously cached data to the new configuration. Any `<DbRead>` specification in the new configuration that has the same `id` as a `<DbRead>` specification in the old configuration is assumed to be the same, even if the query it executes and other details are different. The configuration will therefore inherit any cached data relating to the old specification.

Note: You should take this behavior into account when redesigning database read query specifications. Reusing `<DbRead> ids` could result in unwanted side effects when the new configuration is dynamically loaded.

6.3 Configuring parallel event processing

By default DS4DB only processes one event at a time. Newly arriving events are queued and are processed in the order of arrival.

To get better performance you can configure DS4DB to process multiple events in parallel. Set the `MaxEventParallelism` attribute of the `<DataSourceForDatabases>` 82 configuration tag to a value greater than one. The value you set represents the maximum number of events that DS4DB will simultaneously process.

For more information on improving DS4DB performance through parallel processing see Parallel processing 61.

6.4 Deadlock handling

Some DBMS may occasionally abort queries, in order to avoid deadlock. Should DS4DB detect that a query has been aborted in this way, it will automatically resubmit the query to the DBMS.

By default DS4DB will resubmit a deadlocked query up to 3 times. However, you can change the maximum number of retries using the `MaxQueryRetries` attribute of the `<DataSourceForDatabases>` tag. The queries are retried immediately by default, but you can also configure DS4DB to wait before resubmitting the query; this increases the chances of the resubmitted query succeeding. The period of time to wait between resubmitting deadlocked queries is configured using the `QueryRetryPeriod` attribute of the `<DataSourceForDatabases>` tag.

6.5 PriceMaster compatibility

If you are connecting your DS4DB application to a Caplin PriceMaster peer you should set the `<DataSourceForDatabases>` attribute `PM4Output` to `"true"`. This will make DS4DB behave as a PriceMaster 4 output handler when it becomes aware of newly created symbols. This behaviour is turned off by default (`PM4Output="false"`)

Note: For more information on when to use the `PM4Output` attribute please consult Caplin Systems.

7 More about configuring database access

7.1 Accessing multiple databases

DataSource for Databases can access more than one database at the same time, both within and across events.

Specify each database in a separate `<DatabaseConfiguration>` 80 tag immediately below the `<DataSourceForDatabases>` 82 tag.

For example:

```
<DataSourceForDatabases>

   <DatabaseConfiguration
      dbRef="DBIndexes"
      driver="com.sybase.jdbc2.jdbc.SybDriver"
      connectionURL="mydb.com/IndexesDatabase..."
      maxConnections="10" />

   <DatabaseConfiguration
      dbRef="DBPrices"
      driver="com.mysql.jdbc.Driver"
      connectionURL="mydb.com/PricesDatabase..."
      maxConnections="30" />
...

</DataSourceForDatabases>
```

The `dbRef` attribute uniquely identifies each database, so that the database query in each subsequent `<DbRead>` 84 and `<DbWrite` 87 > tag can be directed (through *its* `dbRef` attribute) to relevant database.

Note that DS4DB can access multiple databases managed by *different* DBMS.
In the above example the "DBIndexes" database is managed by Sybase SQL Server (`driver="com.sybase.jdbc2.jdbc.SybDriver"`),
whereas the "DBPrices" database is a MySQL database (`driver="com.mysql.jdbc.Driver"`).

Also see Configuring access to databases 19, Configuring multiple connections to a database 36, and Parallel processing 61

7.2 Configuring multiple connections to a database

DataSource for Databases allows multiple concurrent connections to a database. This allows multiple queries within the same event, and queries in different events, to access the database at the same time; each query is submitted and the result obtained across a separate connection.

To set up this capability define the `maxConnections` attribute of the `<DatabaseConfiguration>` tag with a value greater than the default of 1. The value you specify determines how many concurrent queries can be made against the database, across all events.

The following configuration example shows two databases, "DBIndexes" and "DBPrices".

DS4DB is allowed to have up to 10 concurrent connections to the "DBIndexes" database:

```
<DatabaseConfiguration
   dbRef=”DBIndexes”
   ...
   maxConnections="10" />
```

The "DBPrices" database is configured to allow up to 30 concurrent connections, because it has to handle a higher number of concurrent queries than "DBIndexes":

```
<DatabaseConfiguration
   dbRef=”DBPrices”
   ...
   maxConnections="30" />
```

An event can submit multiple concurrent queries to either or both of these databases.

Note: When defining `maxConnections` you should make sure its value is compatible with any connection limit setting imposed by the DBMS, allowing for the fact that clients other than DS4DB may also need to connect to the database at the same time as DS4DB.
If `maxConnections` is too high relative to the DBMS connection limit then, when the query rate is high, DS4DB may encounter extensive processing delays or database connection failures (depending on how the DBMS handles connection requests that exceed its own connection limit).

Also see Configuring access to databases 19, Accessing multiple databases 35, and Parallel processing 61

8 About Events

DataSource for Databases accesses the database in response to both internal Timed Events 37 and incoming DataSource events 37.

DS4DB can support multiple events. There is no limit on the number of events (of any type) that can be configured, other than system resources such as available memory.

To improve event processing performance you should configure DS4DB to process multiple events simultaneously. See Parallel processing 61 and Configuring parallel event processing 33.

8.1 Timed Event

A Timed Event is an event that fires at configured intervals. When the event fires, DataSource for Databases performs a configured set of actions on the database. For example you could define a timed event that executes queries against the database at one minute intervals and broadcasts the retrieved data to all connected peers.

8.2 DataSource Events

A DataSource event fires when DataSource for Databases receives a DataSource message that matches a (configured) set of criteria; typically a match against the object name (subject) and/or the values of one or more fields of the message. When the event fires, DS4DB performs a configured set of actions on the database.

There are three types of DataSource event:

- Peer Connection Event 37
- Request Event 38
- Update Event 38

Peer Connection Event

A Peer Connection Event occurs when a connection is made between DataSource for Databases and another DataSource peer. This happens regardless of how the connection was established; the peer could have asked DS4DB for a connection, or DS4DB could have initiated the connection with the peer.

An example of a Peer Connection Event might be for DS4DB to read from the database some meta-data about markets and market indices and forward this to its DataSource peers.

It is recommended that DS4DB applications are configured to handle Peer Connection Events; this ensures that when a peer connects to the DS4DB application it will receive the most up to date data (provided that the DS4DB cache is disabled – see Query caching behaviour 33).

Note: The data returned by any database queries associated with the Peer Connection Event is broadcast to *all* the connected peers.

Request Event

A Request Event occurs when DataSource for Databases receives a Request message from a DataSource peer and the message subject matches the subject specified in the event.

Update Event

An Update Event occurs when DataSource for Databases receives an Update message from a DataSource peer and properties associated with the message match the criteria specified for the event, for example subject, peer id number, field name, field value.

A DataSource peer can use Update messages to send DS4DB a large amount of data (as fields) for insertion in the database.

9 Defining events

You define the events that DataSource for Databases can handle by specifying the following configuration tags in the DS4DB configuration file *DataSourceForDatabases.xml*:

- `<TimedEvent>` 96
- `<PeerConnectionEvent>` 91
- `<RequestEvent>` 92
- `<UpdateEvent>` 97

and the child tags

- `<DbReadref>` 86
- `<DbWriteRef>` 88

The general format of an event definition is:

```
<EventName>
   <DbReadRef ref="Name-Of-A-DbRead" />
   <DbReadRef ref="Name-Of-Another-DbRead" />
   ...
   <DbWriteRef ref="Name-Of-A-DbWrite" />
</EventName>
```

The type of event is defined by the `<EventName>` tag: `<PeerConnectionEvent>`, `<UpdateEvent>`, and so on. The `<DbReadRef>` tags are references to `<DbRead>` 84 tags defining database read queries (see Defining database queries 42). Similarly the `<DbWriteRef>` tag is a reference to a database write or update query, defined in a `<DbWrite>` 87 tag. When the event is triggered these queries are run in the order that the `<DbReadRef>` and `<DbWrite>` tags appear in the configuration file.

There must be at least one `<DbReadRef>` or `<DbWriteRef>` tag in the event definition. There can only be one `<DbWriteRef>` tag and it must follow all the `<DbReadRef>` tags (if there are any).

For a complete description of each configuration tag see the DataSourceForDatabases.xml Configuration Reference 73.

The next sections explain how to use the individual event definition tags.

9.1 How to define a Timed Event

See Configuring a timed event 21 in Setting up a simple configuration 13.

9.2 How to define a Peer Connection Event

You define a Peer Connection Event using the `<PeerConnectionEvent>` 91 tag.

Example:

```
<PeerConnectionEvent>
   <DbReadRef ref="ALL_INDEXES">
      <Param name="ParamListCode" defaultValue="CAPLIN" />
   </DbReadRef>
</PeerConnectionEvent>
```

In this example, when a new peer connects, DS4DB runs the database read query called "ALL_INDEXES". The data returned by this query is broadcast to all the connected peers.

Note: A Peer Connection Event will fire when *any* peer connection is made, so normally there should only be one `<PeerConnectionEvent>` tag in the configuration file.

9.3 How to define a Request Event

You define a Request Event using the `<RequestEvent>` 92 tag. The event is triggered if the subject name in the message matches the subject name specified in the `triggerObject` attribute of the tag.

Example:

```
<RequestEvent triggerObject="/EVENTS/INDEX_OPEN/.*">
   <DbReadRef ref="INDEX" />
</RequestEvent>
```

The `triggerObject` attribute can be a regular expression, as shown here (`"/EVENTS/INDEX_OPEN/.*"`).
In this example the regular expression means the Request Event will be triggered by any Request message that has a subject name starting with `"/EVENTS/INDEX_OPEN/"`.

The event runs the database read query called "INDEX".

DS4DB always acknowledges receipt of the DataSource Request message that triggers the Request Event. The default behavior is to immediately send back an empty update message (DataSource message type DS_DATAUPDATE). This behavior can be changed by specifying the `ackWhenComplete` attribute with the value `true`, so that DS4DB delays sending the acknowledgment message until the event processing is complete. In this case if the event processing fails, the acknowledgment is a NoData message (DataSource message type DS_NODATA), rather than an empty update message.

Send acknowledgment message when event processing complete:

```
<RequestEvent triggerObject="/EVENTS/INDEX_OPEN/.*" ackWhenComplete="true">
   <DbReadRef ref="INDEX" />
</RequestEvent>
```

Note: When `ackWhenComplete` is set to `true`, make sure that any timeout set in the DataSource that sent the message triggering the Request Event is greater then the time taken for DS4DB to complete the event processing and return the request acknowledgement message, otherwise the sender may time out the reply.
If such timeouts occur and you cannot increase the message timeout on the requesting DataSource, you may need to redesign the event processing to execute more quickly (for example, by redesigning the database queries to run faster).

9.4 How to define an Update Event

You define an Update Event using the `<UpdateEvent>` 97 tag. The event is triggered if certain characteristics of the message match the attributes specified in the tag. The attributes are:

- `triggerPeer`
- `triggerObject`
- `triggerField`
- `triggerValue`

These attributes are defined in the reference documentation for <UpdateEvent> 97.

Example:

```
<UpdateEvent triggerObject="/CLOSE/FX/.*" triggerField="bid">
   <DbWriteRef ref="FX_CLOSE">
   ...
   </DbWriteRef>
</UpdateEvent>
```

The `triggerObject` attribute can be a regular expression, as shown here (`"/CLOSE/FX/.*"`).
In this example the regular expression means the Update Event will be triggered by any Update message that has a subject name starting with `"/CLOSE/FX/"` and a field called "`bid`".

The event runs the database write query called "`FX_CLOSE`".

10 Defining database queries

When an event is triggered in DataSource for Databases, the event processing usually involves accessing the associated database. You define the database queries, and associate them with events, by specifying the following configuration tags in the DS4DB configuration file *DataSourceForDatabases.xml*:

- `<DbRead>` 84
- `<DbReadRef>` 86
- `<DbWrite>` 87
- `<DbWriteRef>` 88

and the child tags

- `<SQLParam>` 95
- `<Param>` 90

The DataSourceForDatabases.xml Configuration Reference 73 contains a complete description of each tag.

The following sections explain how to use these tags to define database queries and pass parameter values to a query from an incoming message.

10.1 Query language

You write database queries using the syntax required by your DBMS 101. Most commonly they will be written in Structured Query Language (SQL). Your DBMS may also allow you to use stored procedures 102. Both SQL and stored procedures can write to a database as well as read from it.

10.2 Transaction environment

DataSource for Databases does not provide an implicit database transaction environment. If you want to package your SQL queries into database transactions you should do this in your database queries, using the syntax appropriate to the DBMS you are using, such as "`start transaction`", "`commit`", and so on.

10.3 Example query specification

Here is a simple example of how to define database queries in *DataSourceForDatabases.xml*:

```
<DbRead id="INDEX" dbRef="DB1" prefix="/INDEX/"
                   query="exec caplin_index_sl ? ?" >
   <SQLParam name="ParamIndexCode" sqltype="VARCHAR"/>
   <SQLParam name="ParamDateTime" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="IndexCode" />
</DbRead>

<DbWrite id="INDEX_CALC" dbRef="DB1" query="exec caplin_index_calc ? ?" >
   <SQLParam name="ParamIndexCode" sqltype="VARCHAR"/>
   <SQLParam name="ParamDateTime" sqltype="VARCHAR"/>
</DbRead>

<UpdateEvent triggerObject="/CLOSE/FX/" >
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Field" fieldName="FXIndex" />
      <Param name="ParamDateTime" source="Field" fieldName="FXTime" />
   </DbReadRef>
   <DbWriteRef ref="INDEX_CALC">
      <Param name="ParamIndexCode" source="Field" fieldName="FXIndex" />
      <Param name="ParamDateTime" source="Field" fieldName="FXTime" />
   </DbWriteRef>
</UpdateEvent>
```

The `<DbRead>` 84 tag contains a unique identifier (`id="INDEX"`) and specifies a query that reads the database referred to as "DB1" (`dbRef="DB1"`). The access details for DB1 are defined in the `<DatabaseConfiguration>` tag that has a `dbRef` attribute of `"DB1"` (see Configuring access to databases 19).

The database read is via a stored procedure call `exec caplin_index_sl`. The `<SQLParam>` 95 tags define two parameters that are passed to the stored procedure. DS4DB processes the results of the query using the standard result set processor 54, as defined by the `<StandardResultSetProcessor>` 94 tag.

The `<DbWrite>` 87 tag contains a unique identifier (`id="INDEX_CALC"`) and specifies a query that writes to the same database (`dbRef="DB1"`), via a call to the stored procedure `caplin_index_calc`. This stored procedure also takes two parameters, defined by `<SQLParam>` tags.

These two queries are run when DS4DB receives an Update message whose subject name starts with "`/CLOSE/FX`" – this name matches the `triggerObject` attribute of the `<UpdateEvent>` 97 tag. The Update Event 38 runs the queries in the order of the `<DbReadRef>` 86 and `<DbWriteRef` 88 `>` tags within the `<UpdateEvent>` tag. The `<DbReadRef>` and `<DbWriteRef>` tags identify, through their `name` attributes, the `<DbRead>` and `<DbWrite>` required to run the queries.

Query parameter values will usually come from DataSource messages; in this case they are from the `FXIndex` and `FXTime` fields of the message, as defined in the `<Param>` 90 tags of the `<DbReadref>` and `<DbWriteRef>` (see Mapping parameters to message fields 47.).

10.4 The <DbRead> and <DbWrite> tags

Use the `<DbRead>` 84 and `<DbWrite>` 87 tags to define the database queries that run when an event is triggered. Queries that read the database should be defined in `<DbRead>` tags, and queries that write to the database should be defined in `<DbWrite>` tags.

Each tag must have a unique identifier, defined by the id attribute, and must define through the `dbRef` attribute the database against which the query is to be run.

Write the query as a string in the query attribute; for example:

```
<DbWrite id="UPDATE_DATA" dbRef="DB1" query="exec stored_proc_b ?">
```

Here the query is against the database referred to as "DB1". The access details for DB1 are defined in a `<DatabaseConfiguration>` tag that has a `dbRef` attribute of `"DB1"`; see Configuring access to databases 19.

The query is a call to the stored procedure `exec stored_proc_b`. The '`?`' character is a placeholder for a parameter to be passed to the stored procedure; see How to pass parameters to database queries 46.

You specify the processing to be performed on the returned result set by inserting the appropriate result set processing tag as a child of the `<DbRead>`; see Defining result set processing 53.

10.5 More about query specifications

You must make sure that the detailed syntax of the text strings defining the SQL code and stored procedure calls conform to the rules imposed by your DBMS. DS4DB does not validate your SQL queries when it loads the configuration from the *DataSourceForDatabases.xml* configuration file; they are validated at run time by the DBMS when DS4DB submits them for execution.

Note: There can only be one SQL query or stored procedure call in a the `query` attribute of the `<DbRead>` 84 configuration tag. If there are more then the behaviour is undefined; for example the queries might fail or only partially execute.

The database accessing associated with an event consists of a number of DbRead actions (zero or more), followed by just one optional DbWrite action. There must be at least one action associated with an event, so the simplest action would be just a single DbRead or DbWrite.

When DataSource for Databases processes an event, it executes all the database reads for the event, as defined by the `<DbReadRef>` tags, and then executes the database write defined by the `<DbWriteRef>` tag, if there is one.

If the DbReads only access one database and simultaneous database connections are not enabled (that is, the `maxConnections` attribute of the `<DatabaseConfiguration>` tag is not specified or is set to 1), the DbReads are carried out in the order of the `<DbReadRef>` tags. Conversely, if the queries access multiple databases and/or `maxConnections` is greater than 1, DS4DB executes as many of the database reads as possible in parallel; in this situation the read queries can be executed in effectively any order. For more about this see Accessing multiple databases 35 and Configuring multiple connections to a database 36.

The result sets are retained and are only sent out as messages to the peers when all the database actions for the event are completed. In this respect, the set of queries executed in response to an event is effectively an *atomic* operation – either all messages are sent out and any changes to the database are made, or no messages are sent out at all and no changes are made to the database. Whether or not any messages are actually sent depends on whether caching is enabled and whether any data in the cache has changed – see Query caching behaviour 33.

Note: DataSource for Databases does not currently support the use of stored procedures that return more than one result set.

10.6 Referring to queries in events

Each `<DbRead>` and `<DbWrite>` is uniquely identified by its `id` attribute; for example:

```
<DbWrite id="INDEX_CALC" query="exec stored_proc_b ?">
...
</DbWrite>
```

Here the id is "`INDEX_CALC`".

When you define an event you use the `<DbReadRef>` and `<DbWriteRef>` tags to declare which queries the event is to run:

```
<TimedEvent refreshPeriod = "30000">
   <DbWriteRef ref="INDEX_CALC">
      <Param name="ParamIndexCode" source="Identifier" />
   </DbWriteRef>
</TimedEvent>
```

In this example the Timed Event specifies a `<DbWriteRef>` that refers to the `<DbWrite>` called `INDEX_CALC`, so the event will run the stored procedure `stored_proc_b`.

Once you have defined a query the `<DbRead>` and `<DbWrite>` tags allow you to use it in multiple events.

How to use the `<Param>` tag is explained in Mapping parameters to message fields 47 and Taking a parameter value from a subject (symbol) name 49.

10.7 How to pass parameters to database queries

The database query specified in a `<DbRead>` or `<DbWrite>` can take parameters. The values of the parameters are normally obtained from the object (subject) name and/or fields within the incoming messages.

The following simple example shows how to write DataSource for Databases configuration that will pass a parameter value from an Update message to a stored procedure call.

```
<DbRead id="INDEX" prefix="/INDEX/" dbRef="DB1"
                   query="exec caplin_index_sl ? ?" >
   <SQLParam name="ParamIndexCode" sqltype="VARCHAR"/>
   <SQLParam name="ParamDateTime" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="IndexCode" />
</DbRead>

<UpdateEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX" />
</UpdateEvent>
```

The stored procedure call takes two parameters, indicated by the two '`?`' markers in the query text of the `<DbRead>`:

```
query="exec caplin_index_sl ? ?"
```

The parameters are given names and data types in two `<SQLParam>` tags, which are children of the `<DbRead>` tag. The `<SQLParam>` tags must be declared in the same order as the '`?`' markers in the query, so in this case the first '`?`' (the one on the left) is the parameter called `ParamIndexCode`, and the second '`?`' is the parameter called `ParamDateTime`.

The data type of each parameter is defined in the `sqltype` attribute. This is a SQL type as defined by the JDBC interface. You must assign a suitable SQL type to each parameter, and the SQL type must correspond to the data type of the equivalent database table column that the query will access. In the example each parameter is assigned the SQL type `"VARCHAR"`, because the equivalent columns in the database table(s) accessed by the stored procedure are of type VARCHAR.

The SQL types supported by DS4DB are defined in the reference documentation for <SQLParam> 95.

Note: DataSource for Databases supports a subset of the standard list of SQL types as defined by the JDBC interface.
In particular there is no date/time data type; dates and times must be dealt with as text strings.

The Update message triggers an Update Event:
`<UpdateEvent triggerObject="/EVENTS/INDEX_OPEN">`

Assume an Update message is received with subject (symbol) = "`/EVENTS/INDEX_OPEN`"
and the fields:

```
   Value = 23
   ParamIndexCode = UKX
   ParamDateTime = 2006-20-03T12:32:23
```

The message triggers the Update Event because the subject matches the `triggerObject` specification (`"EVENTS/INDEX_OPEN"`).

The resulting query is:
`exec caplin_index_sl "UKX" "2006-20-03T12:32:23"`

With this simple configuration, DS4DB will only find a parameter in the message if it contains a field whose name matches the name given in the `name` attribute of the `<SQLParam>` tag. To specify more flexible parameter matching use the `<param>` tag within the `<DbReadRef>` or `<DbWriteRef>` – see Mapping parameters to message fields 47 and Extracting part of a field value to pass as a parameter 50.

Mapping parameters to message fields

The simplest way to specify a field to be used as parameter is to just specify its name in the `name` attribute of the `<SQLParam>` tag:

```
<SQLParam name="ParamIndexCode" />
```

However, this means you can only extract the parameter value from messages containing fields of this name, and for the query to work, the name must also be a valid column name in the DBMS.

You can remove this dependency by inserting `<Param>` child tags in the `<DbReadRef>` and `<DbWriteRef>` tags.

Example:

```
<DbRead id="INDEX" dbRef="DB1" prefix="/INDEX/"
                   query="exec caplin_index_sl ? ?" >
   <SQLParam name="ParamIndexCode" sqltype="VARCHAR"/>
   <SQLParam name="ParamDateTime" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="IndexCode" />
</DbRead>

<UpdateEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Field" fieldName="MyIndex" />
      <Param name="ParamDateTime" source="Field" fieldName="DateTime" />
   </DbReadRef>
</UpdateEvent>
```

The tag `<Param name="ParamIndexCode" source="Field" fieldName="MyIndex" />` has the following effect, as specified by its attributes:

- `Param name="ParamIndexCode"`

  The parameter requiring a value is called `ParamIndexCode` .

- `source="Field"`.

  The parameter value is to be taken from a field in the message

- `fieldName="MyIndex"`

  The message field containing the parameter value is called "`MyIndex`".

Similarly, in `<Param name="ParamDateTime" source="Field" fieldName="DateTime" />` the value of the parameter `ParamDateTime` is obtained from the message field called `DateTime`.

At run time, DS4DB will pass the values in the message fields called `MyIndex` and `DateTime` to the query parameters `ParamIndexCode` and `ParamDateTime` respectively.

Tip: This technique allows you to the run same database query in response to messages received with different field names.

The default value of the source attribute is `"Field"`, so you can omit this attribute and achieve the same result as in the example above:

```
<UpdateEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" fieldName="MyIndex" />
      <Param name="ParamDateTime" fieldName="DateTime" />
   </DbReadRef>
```

Passing default parameter values

Use the `defaultValue` attribute of the `<Param>` tag to supply the database query with a default parameter value when a matching field cannot be found in the received message.

Example:

```
<UpdateEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Field" fieldName="MyIndex"
                   defaultValue="XYZ" />
   </DbReadRef>
</UpdateEvent>
```

Tip: Use the `defaultValue` attribute to pass parameter values to a query that is run in a Timed Event, since such an event has no incoming message from which parameter values can be obtained.

Example:

```
<TimedEvent refreshPeriod="10000">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode"
                   defaultValue="XYZ" />
   </DbReadRef>
</TimedEvent>
```

Taking a parameter value from a subject (symbol) name

If you specify `source="Identifier"` in the `<Param>` tag, DataSource for Databases takes the parameter value from the subject (symbol) name in the message.

Example 1:

```
<UpdateEvent triggerObject="/MYINDEX">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Identifier" />
   </DbReadRef>
</UpdateEvent>
```

Assume an Update message is received with subject (symbol) = "`/MYINDEX`"
and the fields:

```
   Value = 23
   ParamIndexCode = UKX
```

Then the parameter `ParamIndexCode` takes the value "`/MYINDEX`", *not* "`UKX`".

Example 2:

```
<UpdateEvent triggerObject="/MYINDEX/.*">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Identifier" />
   </DbReadRef>
</UpdateEvent>
```

In this example the `triggerObject` attribute of the `<UpdateEvent>` tag contains a regular expression `/MYINDEX/.*` that matches any Update message whose subject (symbol) name begins with "`/MYINDEX/`". So an Update message that has the subject name "`/MYINDEX/A1`" will trigger a database read query where the parameter `ParamIndexCode` takes the value "`/MYINDEX/A1`".

See also example 3 in Extracting part of a field value to pass as a parameter 50.

Extracting part of a field value to pass as a parameter

Your DataSource for Databases application may need to pass just part of a message field value as a query parameter. You can do this using the `pattern` attribute of the `<Param>` tag. The pattern value should be a regular expression that extracts the relevant part of the field value.

Example 1:

```
<UpdateEvent triggerObject="/INDEXCALC/.*">
   <DbWriteRef ref="INDEX_CLOSE">
      <Param name="ParamIndexCode" source="Field"
             fieldName="IndexCode" pattern="/INDEXVAL/(.*)" />
   </DbWriteRef>
</UpdateEvent>
```

Here the regular expression `/INDEXVAL/(.*)`is applied to the value of the field `IndexCode` in the received Update message. This particular regular expression means 'match the string beginning with `/INDEXVAL/` and extract just the part to the right of `/INDEXVAL/`.

The `<UpdateEvent>` tag also uses a regular expression to match the subject name, so any Update message whose subject starts with `/INDEXCALC/` will trigger the Update event.

Assume an Update message is received with subject `/INDEXCALC/ABC` and the field `IndexCode="/INDEXVAL/XYZ"`, then the Update Event is triggered and the parameter `ParamIndexCode` takes the value `XYZ`

Example 2:

```
<UpdateEvent triggerObject="/INDEXCALC/.*">
   <DbWriteRef ref="INDEX_CLOSE">
      <Param name="ParamIndexCode" source="Field"
             fieldName="IndexCode" pattern="/INDEXVAL/([0-9]*).*" />
   </DbWriteRef>
</UpdateEvent>
```

In this example the regular expression is `/INDEXVAL/([0-9]*).*` which means 'match any number of digits to the right of the string `/INDEXVAL` and stop when a non-digit character is found'.

If an Update message is received with subject `/INDEXCALC/ABC` and the field `IndexCode="/INDEXVAL/123_456"` then the Update Event is triggered and the parameter `ParamIndexCode` takes the value `123`

Example 3:

```
<UpdateEvent triggerObject="/MYINDEX/.*">
   <DbReadRef ref="INDEX">
      <Param name="ParamIndexCode" source="Identifier"
             pattern="/MYINDEX/(.*)" />
   </DbReadRef>

</UpdateEvent>
```

In this example the `triggerObject` attribute of the `<UpdateEvent>` tag contains the regular expression `/MYINDEX/.*` This regular expression matches any Update message whose subject (symbol) name begins with `/MYINDEX/`.

The source attribute of the `<Param>` tag is set to `"Identifier"`, so this time the database query parameter will be taken from the subject (symbol) name of the Update message. The pattern attribute is defined as a regular expression `/MYINDEX/(.*)` that matches the first part of the subject name (the string "`/MYINDEX/`"), and extracts the part to the right of `/MYINDEX/`.

So, if an Update message is received that has the subject name "`/MYINDEX/A1`", then the Update Event is triggered and the parameter `ParamIndexCode` takes the value "`A1`".

Passing parameters from Request messages

When the DataSource message is a Request, any parameter values that need to be passed to the database query must be transmitted *in the message* as an extension to the subject name (“symbol”). This is because Request messages do not contain fields, so it is not possible to transmit parameter values as field values.

The parameter syntax in the message is:

`subject-name&param-name-1=param-value-1&param-name-2=param-value-2`…

Example:

Assume the subject name in the Request message is:

`/EVENTS/INDEX_OPEN&ParamIndexCode=UKX&ParamDateTime=2006-20-03T12:32:23`

The message subject is `/EVENTS/INDEX_OPEN`

There are two parameters:

- `param-name-1` is `ParamIndexCode`, and `param-value-1` is `UKX`
- `param-name-2` is `ParamDateTime`, and `param-value-2` is `2006-20-03T12:32:23`

The following configuration will cause these parameters to be extracted:

```
<DbRead id="INDEX" dbRef="DB1" prefix="/INDEX/"
                   query="exec caplin_index_sl ? ?" >
   <SQLParam name="ParamIndexCode" sqltype="VARCHAR" />
   <SQLParam name="ParamDateTime" sqltype="VARCHAR" />
   <StandardResultSetProcessor nameColumn="IndexCode" sendName="true" />
</DbRead>

<RequestEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX" />
</RequestEvent>
```

When the Request message is received it triggers the Request Event, because a match is found between the subject and the `triggerObject` specification "`EVENTS/INDEX_OPEN`". Note that when DS4DB's event handling tries to match the subject it ignores the trailing parameter string in the message.

The `<DbRead>` then matches the parameter names on the end of the Request message to form the query:

```
exec Caplin_index_sl UKX 2006-20-03T12:32:23
```

10.8 Modifying the generated subject (symbol) name

When an event generates an outgoing message the subject name in the message (also called the "symbol") is derived from the name of a column in the result set. The column is specified in the result set processor definition associated with the event (see Defining result set processing 53).

You can use the optional `<DbRead>` attributes called `prefix` and `postfix` to modify the subject name in the messages resulting from the query. The `prefix` attribute adds a fixed text string to the front (left hand end) of the subject name, `postfix` adds a fixed text string to the end (right hand end) of the subject name.

Example:

```
<DbRead dbRef="DB1" id="GET_PRICES"
         prefix="/EQUITY/"
         query="select stock, ask_price, bid_price from equity">

   <StandardResultSetProcessor nameColumn="stock" />
</DbRead>
```

This `<DbRread>` specifies a prefix of "`/EQUITY/`". The standard result set processor associated with the `<DbRead>` specifies that the subject name in the generated message is the contents of the **stock** column in the result set (`nameColumn="stock"`). Assume the retrieved value of **stock** is "`ABC`". If the `<DbRread>` did not specify a `prefix` attribute, the subject name in the generated message would be "`ABC`", but in this case the subject name becomes "`/EQUITY/ABC`".

Note: If you specify a `prefix` and/or `postfix` string in a `<DbRead>`, the string will be applied to *all* messages resulting from the query, regardless of the event that triggered the query.

11 Result set processing

The data returned by a read query on the database is called a result set. A result set is a logical table divided into rows and columns, as in the standard depiction of a table within a relational database.

Example of a result set

Column:	1	2	3
Column Name:	**index_code**	**ask_price**	**bid_price**
Row 1	/ABC	52.032	51.160
Row 2	/XYZ	11.650	10.904

The columns in the above table represent the data fields of the result set; all the data items within a column are for the same field. In the example above, column 2 contains all the returned values for the field called '**ask_price**'. Each row contains the values of a related set of fields; for example row 1 is `index_code=/ABC, ask_price=52.032, bid_price=51.160`.

DataSource for Databases has a number of result set processors that turn a result set into one or more DataSource Update messages, which it then broadcasts to connected peers.

You define in the *DataSourceForDatabases.xml* configuration the result set processing to be applied to a query. There are three standard types:

- Standard result set processing 54
- Compound result set processing 55
- Collapsing result set processing 57

You can also implement customized result set processors 60.

Defining result set processing

You use configuration tags in the DS4DB configuration file *DataSourceForDatabases.xml* to specify the result set processing performed on the data retrieved by a read query. Insert a result set processing tag as a child of the `<DbRead>` 84.

Example:

```
<DbRead id="INDEXES" dbRef="DB1" prefix="/INDEX/" query="exec caplin_index_sls ?" >
   <SQLParam name="ParamListCode" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="index_code" sendName="true"/>
</DbRead>
```

In this example the result set processing is defined by the `<StandardResultSetProcessor>` tag

The result set processing tags are:

- `<StandardResultSetProcessor />` 94
- `<CompoundNameResultSetProcessor />` 77
- `<CollapsingResultSetProcessor>` 75
- `<CustomResultSetProcessor />` 79

These have child tags:

- `<Field />` 89
- `<CompoundField />` 76

The DataSourceForDatabases.xml Configuration Reference 73 contains a complete description of each tag.

The next sections explain how to use these tags.

11.1 Standard result set processing

The standard method for processing a result set is to convert each row returned from the database into a single DataSource message. Each field of the row becomes a value in the message (apart from the field identified by the `nameColumn` configuration attribute – see How to define Standard Result Set Processing 55). The XML configuration specifies which column of the result set contains the subject (symbol) name that is to be used for the subject of the Update message.

For example, consider the following result set:

Column:	1	2	3
Column Name:	**index_code**	**ask_price**	**bid_price**
Row 1	/ABC	52.032	51.160
Row 2	/XYZ	11.650	10.904

When applied to this result set, standard result set processing would produce the following two DataSource Update messages, assuming the configuration specifies that the subject name is obtained from the column called **index_code**:

Message no.	Subject (symbol)	Fields
1	`/ABC`	`ask_price=52.032, bid_price=51.160`
2	`/XYZ`	`ask_price=11.650, bid_price=10.904`

How to define Standard Result Set Processing

You define Standard result set processing using the <StandardResultSetProcessor> 94 tag.

Example:

```
<DbRead id="INDEXES" dbRef="DB1" prefix="/INDEX" query="exec caplin_index_sls ?" >
   <SQLParam name="ParamListCode" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="index_code" sendName="true"/>
</DbRead>
```

The `nameColumn` attribute specifies the result set column used to obtain the subject (symbol) name for each generated update message; in this case the column is called "`index_code`".
The `prefix` attribute of the <DbRead> 84 tag specifies the subject name prefix "`/INDEX`".
Assume a row of the result set has index_code="/`XYZ`". DS4DB concatenates the index_code value with the `prefix` attribute value to produce the subject name "`/INDEX/XYZ`".

If you don't specify a `sendName` attribute (or specify it as "`false`"), the fields belonging to the column identified by the `nameColumn` attribute do not appear in the generated messages. However, in this example `sendName` is set to "`true`", so DS4DB *does* put these fields in the message. For example, the index_code value "/`XYZ`" appears as a field "`index_code=/XYZ`".

When the result set shown above is processed according to the example configuration shown here, DS4DB generates the following messages:

Message no.	Subject (symbol)	Fields
1	`/INDEX/ABC`	`index_code=/ABC, ask_price=52.032, bid_price=51.160`
2	`/INDEX/XYZ`	`index_code=/XYZ, ask_price=11.650, bid_price=10.904`

11.2 Compound result set processing

Compound result set processing is like standard result set processing, with the addition that the subject name in the generated message is derived from more than one column.

For example, consider the following result set:

Column:	1	2	3	4	5
Column Name:	**index_code**	**currency_ code**	**dealer_ code**	**ask_price**	**bid_price**
Row 1	/ABC	USD	D021	52.032	51.160
Row 2	/XYZ	EUR	D340	11.650	10.904

Assume the configuration specifies that the subject name is to be derived from the columns called **index_code**, **currency_code** and **dealer_code.** When applied to this result set, compound result set processing would produce the following two DataSource Update messages:

Message no.	Subject (symbol)	Fields
1	`/ABC/USD/D021`	`ask_price=52.032, bid_price=51.160`
2	`/XYZ/EUR/D340`	`ask_price=11.650, bid_price=10.904`

How to define Compound Result Set Processing

You define Compound result set processing using the tag `<CompoundNameResultSetProcessor>` 77.

Example:

```
<DbRead id="SPOT" dbRef="DB1" prefix="/Spot" query="exec caplin_fx_spot_sls ?>
   <SQLParam name="ParamSeriesCode" sqltype="VARCHAR" />
   <CompoundNameResultSetProcessor
      nameColumns="index_code,dealer_code,currency_code"
      nameFormat="{0}/{2}/{1}"
      sendName="true" />
</DbRread>
```

The `nameColumns` attribute specifies the result set columns used to obtain the subject name for each generated Update message; in this case the columns are "`index_code`", "`dealer_code`" and "`currency_code`".

The `nameFormat` attribute specifies how the subject name is to be constructed from the `nameColumns` list, where `{0}` means the first name in the list, `{1}` the second name, `{2}` the third name, and so on.

The `prefix` attribute of the `<DbRead>` 84 tag specifies the subject name prefix "`/Spot`"

Assume a row of the result set has
index_code="/`XYZ`" (item {0}),
dealer_code="`D340`" (item {1}) and
currency_code="`EUR`" (item {2}).

DS4DB concatenates these values in the order {0}, {2}, {1}, together with the `prefix` attribute value and the '`/`' separators specified in `nameFormat`, to produce the subject name:
"`/Spot/XYZ/EUR/D340`".

Because the (optional) `sendName` attribute is set to "`true`", DS4DB also puts the values of the `nameColumns` into the message as fields:

`"index_code=XYZ"`

`"dealer_code=D340"`

`"currency_code=EUR"`

If `sendName` were set to "`false`", the generated message would not contain these fields.

11.3 Collapsing result set processing

In collapsing result set processing the rows of the result set are collapsed into a smaller number of rows. Each row of the collapsed set becomes a DataSource Update message. The rows are collapsed on the basis of unique values for a designated column. Each field of a collapsed row becomes a value in the message (apart from the field identified by the `nameColumn` configuration attribute – see How to define Collapsing result set processing 58).

For example, consider the following result set:

Column:	1	2	3	4
Column Name:	**index_code**	**weighting**	**market_code**	**ticker**
Row 1	/UKX	2.0	MMM	SSS
Row 2	/LON	1.0	TTT	SSS
Row 3	/UKX	3.6	PPP	XXX

Assume the configuration specifies that the result set should be collapsed on the column called **index_code**. When applied to this result set, collapsing result set processing would produce the following two DataSource Update messages (applying the simplest possible configuration):

Message no.	**Subject (symbol)**	**Fields**
1	/UKX	NumberOfRows=2, weighting1=2.0, weighting2=3.6, market_code1=MMM, market_code2=PPP, ticker1=SSS, ticker2=XXX
2	/LON	NumberOfRows=1, weighting1=1.0, market_code1=TTT, ticker1=SSS

Message 1 is formed by collapsing rows 1 and 3, because both these rows have a **index_code** value of 'UKX'. Message 2 is formed from just row 2, which is the only row with an **index_code** value of 'LON'.

No data is lost by the process of collapsing rows; each message contains all the fields from the rows from which it was generated. The rows are distinguished by appending a number to the name of each field (weighting1, weighting2, market_code1, market_code2, and so on), and there is a generated field (NumberOfRows) that specifies the number of collapsed rows forming the message. You can configure the name of the "NumberOfRows" field.

Tip: Similar results could be achieved using a database stored procedure or SQL query. Consult the documentation that comes with your DBMS product for information on how to do this.

How to define Collapsing result set processing

You define Collapsing result set processing using the <CollapsingResultSetProcessor> 75 tag and its children, <Field> 89 and <CompoundField 76 >.

Example:

```
DbRead id="ALL_ITEMS" dbRef="DB1" prefix="/ITEMS"  query="exec all_items ?">
   <SQLParam name="ParamListCode" sqltype="VARCHAR" />
   <CollapsingResultSetProcessor
      nameColumn="index_code"
      sendName="true"
      numberOfRowsColumn="NumRows">
      <Field name="weighting" outputName="Weightg" />
      <CompoundField nameColumns="market_code,ticker"
                     dataFormat="/TICKER/{0}/{1}"
                     outputName="Element" />
   </CollapsingResultSetProcessor>
</DbRead>
```

Assume the database query returns the result set defined above. Then when DS4DB applies the collapsing result set processing it generates the following two messages:

Message no.	Subject (Symbol)	Fields
1	`/ITEMS/UKX`	`index_code=/UKX` `NumRows=2,` `Weightg1=2.0, Weightg2=3.6,` `Element1=/TICKER/MMM/SSS, Element2=/TICKER/PPP/XXX`
2	`/ITEMS/ LON`	`index_code=/LON` `NumRows=1,` `Weightg1=1.0,` `Element1=/TICKER/TTT/SSS`

The tags and their attributes play the following roles in generating these messages:

- **The `<DbRead>` tag**

  The `prefix` attribute of the `<DbRead>` causes the string "`/ITEMS`" to be added to left hand end of each subject (symbol) name.

- **The `<CollapsingResultSetProcessor>` tag**

  The `nameColumn` attribute specifies that the result set is to be collapsed on the column called "`index_code`".

  Applying this attribute causes rows 1 and 3 to be collapsed into one message, because they both have the value "`UKX`" in the index_code column. Row 2 has a different index_code ("`LON`") so it forms a separate message.

Note: You cannot collapse result set rows on the basis of more than one column.

The column specified by the `nameColumn` attribute is also the basis for the subject names in the generated messages. So, when the `<DbRead> prefix` attribute has been added on, the subject names in the generated message are
`/ITEMS/UKX` and `/ITEMS/LON`.

If you don't specify a `sendName` attribute (or specify it as "`false`"), the fields belonging to the column identified by the `nameColumn` attribute do not appear in the generated messages. However, in this example `sendName` is set to `"true"`, so DS4DB *does* put the subject names into the messages as fields. For example, subject "/UKX" appears as a field "`index_code=/UKX`".

The `numberOfRowsColumn` attribute specifies the name of a field in each generated message that contains the count of the number of collapsed result set rows in the message.
In this case the field is called "`NumRows`".

- **The `<Field>` tag**

  `<Field>` defines a simple field in the generated message. The `name` attribute specifies the result set field called "`weighting`" and the `outputName` attribute specifies that this field is to be called " `Weightg`" in the generated message.

- **The `<CompoundField>` tag**

  `<CompoundField>` allows you to combine two or more columns of the result set into a single field in the generated message.

  The `nameColumns` attribute:
  `nameColumns="market_code,ticker"`
  specifies that the result set fields called `market_code` and `ticker` are to be concatenated in the generated message.

  The `dataFormat` attribute:
  `"/TICKER/{0}/{1}"`
  specifies the format of the concatenated fields. `{0}` means take the first field in the `nameColumns` list, `{1}` means take the second field. The prefix "`/TICKER/`" is added to each concatenated field.

  The `outputName` attribute:
  `outputName="Element"`
  defines the name of the compound field in the generated message.

  Applying these attributes to Row 2 of the result set produces the generated field:
  `Element1=/TICKER/TTT/SSS`

  Applying these attributes to the collapsed Rows 1 and 3 of the result set produces the generated fields:
  ```
  Element1=/TICKER/MMM/SSS
  Element2=/TICKER/PPP/XXX
  ```

In each generated message, fields from different result set rows are distinguished by a number appended to the name of the field: `Weightg1`, `Weightg2`, `Element1`, `Element2`, and so on. These numbers are not related to the row numbers in the original result set; they are unique only within each message. For example, in message 1 the `Weightg1` and `Element1` fields are derived from row 1 of the result set and the `Weightg2` and `Element2` fields are derived from a different row (row3).

11.4 Custom result set processing

You can implement your own customized result set processors, using the Java programming language. To do this you need to write Java code (by implementing the Java Interface **ResultSetProcessor**). For more information on how to do this see the **DataSource for Databases API Reference**.

You refer to a custom result set processor in the DS4DB configuration using the `<CustomResultSetProcessor>` 79 tag, in which you specify the Java class name of the custom result set processor.

Example:

```
<CustomResultSetProcessor
   className="com.mydomain.ds4db.MySpecialResultSetProcessor" />
```

12 Parallel processing

DataSource for Databases has three features that enable you to improve performance through parallel processing.

- When several events occur at the same time, DS4DB can process them in parallel.
- Within an event the database read queries can be executed in parallel.
- DS4DB can access several databases simultaneously.

These features are controlled through configuration. When you first install DS4DB the configuration uses its default settings, which turn off the parallel processing features. This means that when DS4DB runs (and assuming it only accesses one database), it processes events one at a time in the order they arrive. Within an event the database reads are executed one at a time in the order of the `<DbReadRef>` 86 tags within the event definition tag.

Turning on the parallel processing features will allow DS4DB to process events in a more timely fashion and will allow it to handle a greater throughput of events. This is, of course, provided that the databases are able to cope with the increased load on them, and the machine running DS4DB has enough spare CPU capacity to run the additional DS4DB threads.

12.1 Parallel event processing

Enable parallel processing of events by setting the `MaxEventParallelism` attribute of the `<DataSourceForDatabases>` 82 configuration tag to a value greater than 1. The value defines the maximum number of events that can be processed simultaneously.

12.2 Parallel database reads

Parallel execution of database reads is enabled through configuring multiple connections to the database. This is done through the `maxConnections` attribute of the `<DatabaseConfiguration>` 80 tag, which defines the size of a pool of connections that DS4DB manages across all the concurrently running events accessing the database. Each database read or write operation within an executing event is given its own database connection from the pool, and the query is submitted across that connection (and the results processed) independently of the other queries. By this mechanism all read queries that are allocated to connections are executed concurrently.

If the database's connection pool is temporarily exhausted, any remaining read queries in an event cannot be allocated connections and so cannot execute until connections are available once again.

Regardless of how many database reads can take place in parallel, any database write specified at the end of the event (`<DBWriteRef>` 88 tag) only executes when all the event's database reads have completed and there is a database connection available for it.

When DS4DB has opened a connection to a database it will leave it open for use by successive queries. However, every 30 seconds DS4DB checks for idle connections in the connection pool for the database, and closes connections that have been idle for 60 seconds or more. This means that a database connection will be closed if it is idle for between 60 and 90 seconds.

12.3 Accessing multiple databases

DS4DB can also access more than one database.
To enable this, specify a `<DatabaseConfiguration>` 80 tag for each database to be accessed.

There is a separate connection pool for each database, as specified by the `maxConnections` attribute. You can configure a different `maxConnections` value for each database, so each connection pool can be sized to suit the capabilities of its associated database.

Note that if an event has two database reads, each on a different database, the reads will be submitted and processed in parallel, even when the `maxConnections` value for each database is just 1.

Because connections are pooled between events, the number of queries that run in parallel within an event can change from one invocation of the event to the next, depending on how many other queries are also executing in other events at the time.

12.4 Runtime implications of parallel processing

When you configure DS4DB to execute events and database reads in parallel, there are some runtime implications to be considered.

- The order in which the database read queries are executed within an event cannot be guaranteed.

  For example, assume an event has four read queries. On one invocation of the event there may be only two database connections available from the pool. Accordingly only the first two queries may execute in parallel, completing in the order say 2–1. Then the second two events execute in parallel, completing in the order 4–3. So the events complete in the order 2–1–4–3.

  On the next invocation of the event there may be four database connections available, so all four queries execute together and complete in the completely different order 4–1–3–2.

  An implication of this is that the results of one read query must not depend on the results of another.

- The order in which events execute and complete cannot be guaranteed.

  For example, if event A occurs just before event B, the processing of event B could finish before or after that for event A, depending on factors such as, the amount of work to be done in each event, the number of database connections available and the order in which they are allocated to database reads, the time taken for the DBMS to execute the queries, and the scheduling of DS4DB's processing threads.

  An implication of this is that you should be aware of possible unwanted interactions between events, such as potential lost database updates.

12.5 Example parallel processing configuration

The following XML configuration illustrates how you can define parallel processing using the features discussed above.

```
<DataSourceForDatabases MaxEventParallelism="10">

   <!-- Declare two databases, "DB1", and "DB2" -->
   <DatabaseConfiguration dbRef="DB1" maxConnections="1"
                          connectionURL="mydb.com/DB1" ... />
   <DatabaseConfiguration dbRef="DB2" maxConnections="8"
                          connectionURL="mydb.com/DB2" ... />

   <!-- Two database reads on DB1, two reads on DB2 -->
   <DbRead id="MyRead1" dbRef="DB1" query="...">
      <StandardResultSetProcessor nameColumn=col1" />
   </DbRead>
   <DbRead id="MyRead2" dbRef="DB1" query="...">
      <StandardResultSetProcessor nameColumn=col2" />
   </DbRead>
   <DbRead id="MyRead3" dbRef="DB2" query="...">
      <StandardResultSetProcessor nameColumn=col3" />
  </DbRead>
   <DbRead id="MyRead4" dbRef="DB2" query="...">
      <StandardResultSetProcessor nameColumn=col3" />
   </DbRead>

   <!-- A write on each database -->
   <DbWrite id="MyWrite1" dbRef="DB1" query="...">
      <SQLParam name="price" sqltype="FLOAT" />
   </DbWrite>
   <DbWrite id="MyWrite2" dbRef="DB2" query="...">
      <SQLParam name="price" sqltype="FLOAT" />
   </DbWrite>

   <!-- Two Timed Events and a Request Event -->
   <!- Event E1 -->
   <TimedEvent ...>
      <DbReadRef ref="MyRead1">
      </DbReadRef>
      <DbReadRef ref="MyRead2">
      </DbReadRef>
      <DbReadRef ref="MyRead3">
      </DbReadRef>
      <DbReadRef ref="MyRead4">
      </DbReadRef>
   </TimedEvent>

   <!- Event E2 -->
   <TimedEvent ...>
      <DbReadRef ref="MyRead3">
      </DbReadRef>
      <DbReadRef ref="MyRead4">
      </DbReadRef>
      <DbWriteRef ref="MyWrite2">
   </TimedEvent>

   <!- Event E3 -->
   <RequestEvent triggerObject="/EVENTS/INDEX_OPEN">
      <DbReadRef ref="MyRead4">
      </DbReadRef>
      <DbReadRef ref=”MyRead1”>
      </DbReadRef>
      <DbReadRef ref=”MyRead3”>
      </DbReadRef>
      <DbWriteRef ref=”MyWrite1”>
   </RequestEvent>

<DataSourceForDatabases />
```

Parallel events

Events E1, E2 and E3 can execute in parallel, because `MaxEventParallelism` is set to 10.

Database connections

Database DB2 has enough database connections configured (8) for all queries on the database to be executed in parallel most of the time. Database DB1 has only one connection configured, so queries on this database can only proceed one at a time.

Event E1

Event E1 has four database reads, two on database DB1 and two on database DB2.

- MyRead1 and MyRead2 are executed in sequence, because they are both reads on DB1 and only one connection is allowed to DB1
(`<DatabaseConfiguration dbRef=”DB1” maxConnections="1" ...>`).
- MyRead3 and MyRead4 on DB2 proceed concurrently with each other, concurrently with MyRead1 and/or Myread2, and concurrently with any other reads on DB2 in concurrently executing events E2 and E3.

Event E2

Event E2 has two database reads on DB2 and a write on DB2.

- MyRead3 and MyRead4 execute concurrently with each other, and concurrently with any other reads on DB2 in concurrently executing events E1 and E3.
- MyWrite2 executes only after MyRead3 and MyRead4 are complete.

Event E3

Event E3 has three database reads. MyRead4 and MyRead3 are on DB2, and MyRead1 is on DB1. It also has a database write on DB1.

- MyRead4 and MyRead3 execute concurrently with each other, concurrently with MyRead1 (different database), and concurrently with any other reads on DB2 in concurrently executing events E1 and E2.
- MyRead1 could be blocked by the MyRead1 or MyRead2 executing in E1, since there is only a single connection available on DB1.
- MyWrite1 executes only after MyRead1, MyRead3, and MyRead4 are complete, but it too could be blocked until the single connection on DB1 is free.

Reducing `MaxEventParallelism`

If `MaxEventParallelism` were only 2 then only two of the three events could execute in parallel. this limit would override the parallelism at database read level within the individual events.

13 Error Handling

DataSource for Databases logs all run time errors in log files in the same way as other DataSource for Java applications. See Monitoring and Logging 67.

13.1 Configuration file errors

When DS4DB starts up it validates the configuration file *DataSourceForDatabases.xml* against the XML schema in *DataSourceForDatabases.xsd*. If it encounters any errors in the configuration, the DS4DB application logs them and abandons the start up.

13.2 Communication errors

When DS4DB loses a peer connection it automatically tries to reconnect, or if outbound peer connections are configured it will fail over to another connection to the peer.

13.3 Database error handling

Loss of DBMS connection

DS4DB detects the loss of connection to the DBMS only when a query fails to execute because the connection is down. If this happens, DS4DB attempts to reconnect once every second until the connection is re-established. Meanwhile, before attempting to resend the query, DS4DB checks on a different execution thread for the connection to come back up. This second thread waits between checks, with a time delay that doubles on each attempt; thus it waits 1 second, then 2 seconds, 4 seconds, and so on, up to a maximum of 32 seconds. Thereafter it continues to check every 32 seconds. This means that the query may not be resubmitted for up to 32 seconds after the connection is re-established.

Deadlock handling

If a SQL query or stored procedure call results in a deadlock condition being returned by the DBMS, DS4DB submits the query again; it does this up to a configured maximum number of times.
The default retry count is 3, but you can change this using the `MaxQueryRetries` attribute of the `<DataSourceForDatabases>` 82 configuration tag.

The queries are retried immediately by default, but you can also configure DS4DB to wait before resubmitting the query; this increases the chances of the resubmitted query succeeding. The period of time to wait between resubmitting deadlocked queries is configured using the `QueryRetryPeriod` attribute of the `<DataSourceForDatabases>` tag.

No available database connections

If DS4DB runs out of available database connections from its pool (as defined by the `maxConnections` attribute of the `<DataBaseConfiguration>` tag), no error is generated. DS4DB merely blocks the execution of additional queries until connections are available again. This may cause events to take longer to complete than is desirable.

If the connection limit imposed by the DBMS is exceeded, then, depending on how the DBMS handles connection requests, further connection requests may be blocked, or the DBMS may reject such requests and return an error.

To avoid the DBMS rejecting connection requests, make sure the value of `maxConnections` is compatible with any connection limit setting imposed by the DBMS, allowing for the fact that clients other than DS4DB may also need to connect to the database at the same time as DS4DB.

Other database errors

If any other type of database error occurs, or the query retry count is exceeded, the event that triggered the query fails and DS4DB does not publish any data for it. "Any other type of database error" includes badly formed queries in `<DbRead>` and `<DbWrite>` configuration tags. Although DS4DB validates the configuration schema at start up, it does not validate the query specifications; the DBMS validates them at run time.

A failed database write will not cause loss of data integrity on the database, because any `<DbWrite>` is always the last database action and is the only write action for the event.

13.4 Query Parameter errors

If a database query takes parameters, then they must all be available when the query is submitted to the DBMS. Where a parameter is supplied from the incoming message that triggered the event, the relevant field must be present in the message. If DS4DB cannot locate the parameter for a query, the event that triggered the query fails and DS4DB does not publish any data for it, or in the case of a write query it does not write any data to the database.

14 Monitoring and Logging

JMX Monitoring

DataSource for Databases contains the basic JMX 101 interface, so you can monitor a DS4DB application using the Caplin JMX Console. In this release of the product, only the basic monitoring facilities are present; for example you can monitor the JMX Bean objects and the state of connections.

Logging status messages and errors

DS4DB logs internal status messages and error messages to log files in the same way as other DataSource for Java applications. You define the logging characteristics, including the log file name, in the `<logging>` tag of the standard DataSource XML format configuration file, *DataSource.xml*.

DS4DB's packet log file keeps a record of all received and transmitted message data and to which destinations (peers) the data was sent. You define the packet logging characteristics in the `<packetLog>` tag of the standard DataSource XML format configuration file *DataSource.xml*. Packet logging is turned on by default. Since the packet log files can rapidly become very large, you can turn off logging by setting the `nopacketLog` attribute to `"true"`.

For more information about configuring DS4DB's logging characteristics see the **DataSource for Java API Reference**.

15 Troubleshooting

If your DataSource for Databases application does not appear to work, first look in the log file in *$INSTALL_DIR/logs/*. The name of the log file is defined in the `<logging>` tag of the *DataSource.xml* configuration file. In the configuration file supplied with the install kit the log file name is *DataSourceForDatabases.log*.

The following sections show some typical error situations, the log file entries that identify them, and explain how to correct the errors.

15.1 Troubleshooting DataSourceForDatabases.xml configuration errors

When DS4DB starts up it validates the configuration file *DataSourceForDatabases.xml* against the XML schema in *DataSourceForDatabases.xsd*. If it encounters any errors in the configuration, the DS4DB application logs them and abandons the start up.

Here is an example of a *DataSourceForDatabases.xml* configuration error report:

```
DataSource for Databases starting...
Product        : DataSource for Databases
Version        : 4.3.0a
Build Date     : 6-Oct-2006
Build Time     : 15:41
Build Number   : 53032
Copyright      : Copyright 1995-2006 Caplin Systems Ltd
Using DataSourceForDatabases Schema directory 'file:./conf/'

Using DataSourceForDatabases Configuration File './conf/DataSourceForDatabases.xml'

FATAL
SystemId: file:C:/sandbox/DataSourceForDatabases/./conf/DataSourceForDatabases.xml
PublicId: null
Line Num: 10
Col Num : 30
Message : Open quote is expected for attribute "refreshPeriod" associated with
an element type  "TimedEvent".

WARNING: Could not load XML DataSourceForDatabases config file called
/conf/DataSourceForDatabases.xml
Open quote is expected for attribute "refreshPeriod" associated with
an  element type  "TimedEvent".
org.xml.sax.SAXParseException: Open quote is expected for attribute
"refreshPeriod" associated with an  element type  "TimedEvent".
        at org.apache.xerces.util.ErrorHandlerWrapper.createSAXParseException
(Unknown Source)
        at org.apache.xerces.util.ErrorHandlerWrapper.fatalError
(Unknown Source)
        at org.apache.xerces.impl.XMLErrorReporter.reportError
(Unknown Source)
        at org.apache.xerces.impl.XMLErrorReporter.reportError(Unknown Source)
        at org.apache.xerces.impl.XMLScanner.reportFatalError(Unknown Source)
        at org.apache.xerces.impl.XMLScanner.scanAttributeValue(Unknown Source)
        at org.apache.xerces.impl.XMLNSDocumentScannerImpl.scanAttribute
(Unknown  Source)
        at org.apache.xerces.impl.XMLNSDocumentScannerImpl.scanStartElement
(Unknown Source)
        at org.apache.xerces.impl.XMLDocumentFragmentScannerImpl
$FragmentContentDispatcher.dispatch
(Unknown Source)
        at org.apache.xerces.impl.XMLDocumentFragmentScannerImpl.scanDocument
(Unknown Source)
        at org.apache.xerces.parsers.XML11Configuration.parse(Unknown Source)
        at org.apache.xerces.parsers.XML11Configuration.parse(Unknown Source)
        at org.apache.xerces.parsers.XMLParser.parse(Unknown Source)
        at org.apache.xerces.parsers.AbstractSAXParser.parse(Unknown Source)
        at org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXParser.parse(Unknown Source)
        at org.apache.xerces.jaxp.SAXParserImpl.parse(Unknown Source)
        at javax.xml.parsers.SAXParser.parse(Unknown Source)
        at com.caplin.config.xml.XMLConfigurationContext.parse(Unknown Source)
        at com.caplin.datasrc.ds4db.DataSourceForDatabases.
configureDataSourceForDatabases(Unknown Source)
        at com.caplin.datasrc.ds4db.DataSourceForDatabases.main(Unknown Source)
Exiting...
```

The highlighted lines above identify that the error is in the DS4DB configuration file *DataSourceForDatabases.xml* and the cause of the error is in the `refreshPeriod` attribute of a `<TimedEvent>` tag.

The cause of the error is missing quotes on the attribute value:

```
 ...
<TimedEvent refreshPeriod=5000>
   <DbReadRef ref="HELLO"/>
 </TimedEvent>
 ...
```

`<TimedEvent refreshPeriod=5000>`

should be

`<TimedEvent refreshPeriod="5000">`

15.2 Missing dbRef attribute

Because the `dbRef` attribute of the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags is mandatory, earlier versions of the configuration XML that do not include this attribute will not work with this release. If you inadvertently supply DS4DB with an old configuration file that does not contain `dbRef` attributes, it will report the following error:

```
ERROR
SystemId: file:...conf/DataSourceForDatabases.xml
PublicId: null
Line Num: nn
Col Num : mm
Message : cvc-complex-type.4: Attribute 'dbRef' must appear on element
          'DatabaseConfiguration'.
```

15.3 Cannot connect to peer

The following log file entry shows the situation where a DS4DB application cannot connect to a DataSource peer.

```
Nov 23 16:56:24.575 +0000 - INFO: Connecting to peer 0 on localhost:25,001...
Nov 23 16:56:24.576 +0000 - INFO: Failed connecting to peer 0
on localhost:25,001.
Exception : java.net.ConnectException: Connection refused
```

To fix this error you should first check that the peer is actually running. If it is, then examine the DS4DB's *DataSource.xml* configuration file to see whether the connection parameters are correctly defined. For example, check that you have defined the correct port number for the peer in the `<destination>` tag:

```
<peer>
  <remote name="LibDS4DBTest" type="CONTRIB" />
  <destination address="localhost" port="25001"/>
</peer>
```

In this case port number 25001 is incorrect, as the peer is a Caplin Liberator whose configuration file *rttpd.conf* defines the Liberator port number as 25000:

```
## DATASRC ###################################################################
#
#

datasrc-port                                       25000
```

15.4 Cannot connect to database

If your DS4DB application cannot connect to the specified database, the log file will contain an error report that looks like this:

```
Nov 23 15:06:15.170 +0000 - INFO: Connecting to database
using URL <jdbc:mysql://john.xyz.com/ds4dbx?user=john>
Nov 23 15:06:15.177 +0000 - WARNING: Connection Error:
Unknown database 'ds4dbx'
com.mysql.jdbc.exceptions.MySQLSyntaxErrorException: Unknown database 'ds4dbx'
        at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:936)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2870)
        at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:812)
        at com.mysql.jdbc.MysqlIO.secureAuth411(MysqlIO.java:3269)
        at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:1182)
        at com.mysql.jdbc.Connection.createNewIO(Connection.java:2670)
        at com.mysql.jdbc.Connection.<init>(Connection.java:1531)
        at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:266)
        at java.sql.DriverManager.getConnection(DriverManager.java:525)
        at java.sql.DriverManager.getConnection(DriverManager.java:193)
        at com.caplin.datasrc.ds4db.A.B.B(Unknown Source)
        at com.caplin.datasrc.ds4db.A.B.A(Unknown Source)
        at com.caplin.datasrc.ds4db.A.B$1.run(Unknown Source)
        at java.util.TimerThread.mainLoop(Timer.java:512)
        at java.util.TimerThread.run(Timer.java:462)
```

The highlighted lines above identify that the connection to the database called `ds4dbx` has failed because the DBMS server has no record of a database with this name.

In this case the error is in the DS4DB configuration file *DataSourceForDatabases.xml*, where the database name has been spelled incorrectly in the `connectionURL` attribute of the `<DatabaseConfiguration>` 80 tag.

```
<DatabaseConfiguration dbRef="MyDB" driver="com.mysql.jdbc.Driver"
   connectionURL="jdbc:mysql://john.xyz.com/ds4dbx?user=john" />
```

The database name `ds4dbx` should be `ds4db`.

15.5 Failed database query

Although you specify your database queries in the *DataSourceForDatabases.xml* configuration file, DS4DB does not validate them when it loads the configuration; they are only validated at run time by the DBMS when DS4DB submits them for execution.

The following log file entry shows the situation where a query submitted to the MySQL DBMS has failed:

```
Nov 23 16:08:28.224 +0000 - INFO: SQLException caught
(SQLState=42S22, ErrorCode=1054):
com.mysql.jdbc.exceptions.MySQLSyntaxErrorException:
Unknown column 'stokk' in 'field list'
Nov 23 16:08:28.224 +0000 - INFO: Skipping active connection test
because no testTableName has been specified
Nov 23 16:08:28.224 +0000 - INFO: Ran active connection test (result: connection ok)
because of exception com.mysql.jdbc.exceptions.MySQLSyntaxErrorException:
Unknown column 'stokk' in 'field list'.
Nov 23 16:08:28.224 +0000 - WARNING: Task failed, SQLState: 42S22
```

The query that caused this error is:

```
<DbRead id="GET_PRICES"
        prefix="/EQUITY/"
        query="select stokk, ask_price, bid_price from equity" >
  <StandardResultSetProcessor nameColumn="stock" sendName="true" />
</DbRead>
```

In this case there is a spelling error in the query specification; the column `stokk` should be `stock`:

```
<DbRead id="GET_PRICES"
        prefix="/EQUITY/"
        query="select stock, ask_price, bid_price from equity" >
  <StandardResultSetProcessor nameColumn="stock" sendName="true" />
</DbRead>
```

16 DataSourceForDatabases.xml Configuration Reference

This section contains the reference information for the DS4DB configuration. The configuration is defined in the XML format file *$INSTALL_DIR/conf/DataSourceForDatabases.xml*.

At start up, DS4DB validates the configuration in *DataSourceForDatabases.xml* against the XML schema in *$INSTALL_DIR/conf/DataSourceForDatabases.xsd* This schema enforces order, so elements (tags) within other elements must appear in the correct order as declared in the schema.

The hierarchy and order of tags is shown for convenience below (but note that the schema contains the master definition of this information). The | marker means 'or'; for example
`<PeerConnectionEvent | RequestEvent>`
means that the `<PeerConnectionEvent>` tag *or* the `<RequestEvent>` tag is valid at that particular point in the configuration file.

Note: Tags without children must be self-closing (end with `/>`), or the run-time validation will fail.

Hierarchy and order of tags:

```
<DataSourceForDatabases>

   <DatabaseConfiguration />

   <DbRead>
      <SQLParam />
      <StandardResultSetProcessor />
      |
      <CompoundNameResultSetProcessor />
      |
      <CollapsingResultSetProcessor>
         <Field />
         <CompoundField />
      </CollapsingResultSetProcessor>
      |
      <CustomResultSetProcessor />
   </DbRead>

   <DbWrite>
      <SQLParam />
   </DbWrite>

   <PeerConnectionEvent | RequestEvent | UpdateEvent | Timed Event>
      <DbReadRef>
         <Param />
      </DbReadRef>
      <DbWriteRef>
         <Param />
      </DbWriteRef>
   </PeerConnectionEvent | RequestEvent | UpdateEvent | TimedEvent>

</DataSourceForDatabases>
```

The next sections define the XML tags in alphabetical order.

Each section contains the following information:

- The tag name.
- A brief definition of what the tag does.
- A tabular list of the tag's attributes:
  - The attribute name.
  - Its data type.
  - The default value the attribute takes (if any), when it is not explicitly included in the tag.
  - Whether or not the attribute must always be explicitly included in the tag ("Req? Y/N").
  - A description of the attribute.
- A list of the immediate children of the tag (if any).
- More explanation of how to use the tag, if required.
- A code example showing the tag in use.
- Links to relevant "how to" sections.

16.1 <CollapsingResultSetProcessor>

Tag: `<CollapsingResultSetProcessor>`

Defines how to apply collapsing result set processing to the result set returned by a database read (`<DbRead>`).

Attributes:

Name	Type	Default value	Req?	Description
`nameColumn`	string	(none)	Y	Specifies the name of a column in the result set on which the results are to be collapsed. For each row in the collapsed result set, the value in this column is also appended to the string defined by the `prefix` attribute of the enclosing `<DbRead>` tag, followed by the `<DbRead>` `postfix` string (if any). The resulting string forms the subject (symbol) name of the generated Update message.
`sendName`	boolean	`false`	N	If set to `true`, for each row in the collapsed result set, the value in the column specified by `nameColumn` is included as a field in the generated Update message.
`numberOfRows Column`	string	`"NumberOfRows"`	N	The name of the field in each generated Update message that contains the count of the number of collapsed result set rows in the message.

The immediate children of `<CollapsingResultSetProcessor>` are zero or more `<Field>` 89 tags followed by zero or more `<CompoundField>` 76 tags.

Example:

```
<DbRead id="ALL_ITEMS" prefix="/ITEMS"  query="exec all_items ?">
   <SQLParam name="ParamListCode" sqltype="VARCHAR" />
   <CollapsingResultSetProcessor
      nameColumn="index_code"
      sendName="true"
      numberOfRowsColumn="NumRows">
      <Field name="weighting" outputName="Weightg" />
      <CompoundField nameColumns="market_code,ticker"
                     dataFormat="/TICKER/{0}/{1}"
                     outputName="Element" />
   </CollapsingResultSetProcessor>
</DbRead>
```

For more information see How to define Collapsing Result Set Processing 58.

16.2 <CompoundField />

Tag: `<CompoundField />`

Defines a compound field in the messages generated by collapsing result set processing. This tag allows you to combine two or more columns of the result set into a single field in the generated message.

Attributes:

Name	Type	Default value	Req?	Description
`nameColumns`	string	(none)	Y	A comma separated list of the names of columns in the result set that are to be combined.
`dataFormat`	string	(none)	Y	The specification of how the data in the `nameColumns` list is to be combined. The format of the specification is `prefix{n1}separator{n2}separator{n3}...`, where `n1`, `n2`, `n3` are the numerical positions of the names in the list, starting from 0. Thus the first name in the `nameColumns` list is referred to as `{0}`, the second name as `{1}`, the third name as `{2}`, and so on. The prefix and separator strings are optional. See the example below.
`outputName`	string	(none)	Y	The name of the combined field within the generated message. An ordinal number is appended to each occurrence of the field name in each message, for example `Element1`, `Element2`, etc.

This tag can only be used as a child tag of `<CollapsingResultSetProcessor>` 75. It has no child tags.

Example:

```
DbRead id="ALL_ITEMS" prefix="/ITEMS/"  query="exec all_items ?">
   <SQLParam name="ParamListCode" sqltype="VARCHAR" />
   <CollapsingResultSetProcessor
      nameColumn="index_code"
      numberOfRowsColumn="NumRows">
      <Field name="weighting" outputName="Weightg" />
      <CompoundField nameColumns="market_code,ticker"
                     dataFormat="/TICKER/{0}/{1}"
                     outputName="Element" />
   </CollapsingResultSetProcessor>
</DbRead>
```

How the `dataFormat` attribute is used:
In the above example the generated value starts with the string “/TICKER/”, followed by the value of the `market_code` column (the zeroth element in the `nameColumns` list, hence {0}), followed by a “/”, followed by the value of the `ticker` column (the first element in the `nameColumns` list, hence {1}).
A typical generated field in the Update message would therefore be:

```
Element1=/TICKER/MMM/TTT
```

where `MMM` is a value in the `market_code` column and `TTT` is a value in the `ticker` column.

For more information see How to define Collapsing Result Set Processing 58.

16.3 <CompoundNameResultSetProcessor />

Tag: `<CompoundNameResultSetProcessor />`

Defines how to apply compound name processing to the result set returned by a database read (`<Dbread>` 84).

Attributes:

Name	Type	Default value	Req?	Description
`nameColumns`	string	(none)	Y	The result set columns that are used to define part of the subject (symbol) name for each generated Update message. This attribute contains a comma separated list of column names. For each row in the result set, the values in these columns are appended to the string defined by the `prefix` attribute of the enclosing `<DbRead>` tag, followed by the `<DbRead>` `postfix` string (if any). The resulting string forms the subject (symbol) name of the generated Update message.
`nameFormat`	string	(none)	Y	The specification of how the values defined by `nameColumns` are to be formatted into a string. The format of the specification is `{n1}separator{n2}separator{n3}...`, where `n1`, `n2`, `n3` are the numerical positions of the names in the list, starting from 0. Thus the first name in the `nameColumns` list is referred to as `{0}`, the second name as `{1}`, the third name as `{2}`, and so on. The separator strings are optional. See the example below.
`sendName`	boolean	`false`	N	If set to `true` then the values of the columns specified in `nameColumns` are included as fields in each generated Update message. The name of each field is as specified in the `nameColumns` list. If set to false then the generated messages do not contain these fields.

This tag has no child tags.

Example:

```
<DbRead id="SPOT" prefix="/Spot" query="exec caplin_spot_sls ?>
   <SQLParam name="ParamSeriesCode" sqltype="VARCHAR" />
   <CompoundNameResultSetProcessor
      nameColumns="Code,SpotValCode,BaseValCode"
      nameFormat="{0}/{2}/{1}"
      sendName="true" />
</DbRread>
```

In the above example, if the result set columns specified by the `nameColumns` list contain “`S1`”, “`EUR`” and “`USD`” respectively, then the returned subject name is “`/Spot/S1/USD/EUR`”

Because the `sendName` attribute is set to `"true"`, DS4DB also puts the values of the `nameColumns` into the message as fields:

```
"Code=S1"
"SpotValCode=EUR"
"BaseValCode=USD"
```

For more information see How to define Compound Result Set Processing 56.

16.4 <CustomResultSetProcessor />

Tag: `<CustomResultSetProcessor />`

Defines a custom result set processor.

Attributes:

Name	Type	Default value	Req?	Description
`className`	string	(none)	Y	The fully qualified name of the Java class that implements the custom result set processor.

To implement a custom result set processor you need to write Java code (by implementing the Java Interface **ResultSetProcessor**). For more information on how to do this see the **DataSource for Databases API Reference**.

Example:

```
<CustomResultSetProcessor
   className=”com.mydomain.ds4db.MySpecialResultSetProcessor” />
```

16.5 <DatabaseConfiguration />

Tag: `<DatabaseConfiguration />`

Defines the connection to a Database Management System (DBMS) to which the DS4DB application is to send database queries. The connection should specify, via the `connectionURL` attribute, a particular database that is managed by the DBMS.

Attributes:

Name	Type	Default value	Req?	Description
`dbRef`	string	(none)	Y	The identifier of this `<DatabaseConfiguration>`. This must be unique amongst all the occurrences of `<DatabaseConfiguration>` within the configuration.
`driver`	string	(none)	Y	The fully qualified name of the JDBC technology-based driver that handles the connection to the DBMS.
`connectionURL`	string	(none)	Y	The URL on which the connection to the DBMS is to be made.
`maxConnections`	int or the string `"unbounded"`	1	N	The maximum number of connections that can be made to the database at any one time. This determines the maximum number of database queries (as defined by `<DbRead>and <DbWrite>` tags) that DS4DB can execute at a time. If there are more queries to be submitted to this database than there are available connections, the excess queries are queued until connections are available for them. The value of this attribute should not be greater than any connection limit configured in or imposed by the DBMS. See Configuring multiple connections to a database 36. The default value of 1 causes DS4DB to behave as in previous versions of the software; that is, only one database query is submitted to this database at a time. The value `"unbounded"` means that DS4DB puts no limit on the number of database connections that it will allow, and all outstanding read queries for the database will be submitted at the same time. However there are likely to be limits on the number of connections imposed by the DBMS and other software and hardware resources – see the note below.

This tag has no child tags.

The configuration must contain an instance of the `<DatabaseConfiguration>` tag for each individual database that DS4DB can access, and there must be at least one such tag. They must be the first tags inside the `<DataSourceForDatabases>` 82 tag.

Note: Because the `dbRef` attribute is mandatory, earlier versions of the configuration XML that do not include this attribute will not work with this release. Edit your old configuration files to include the new `dbRef` attribute in the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags, otherwise DS4DB will report XML schema validation failures at start up and will refuse to load the file.
See Missing dbRef attribute 70 in Troubleshooting 68.

Note: Setting the `maxConnections` attribute to `"unbounded"` could cause query failures through the DBMS reaching its own connection limit, or because other software or hardware resource limits are reached, such as the maximum number of TCP/IP sockets that can be allocated.

Example:

```
<DataSourceForDatabases>
   <DatabaseConfiguration
      dbRef="DB1"
      driver="com.sybase.jdbc2.jdbc.SybDriver"
      connectionURL="jdbc:sybase:Tds:xyz.com:5000/bob?user=sa&amp;charset=cp1250"
      maxConnections="5" />
...
</DataSourceForDatabases>
```

For more information see Configuring access to databases 19.

16.6 <DataSourceForDatabases>

Tag: `<DataSourceForDatabases>`

The root tag for configuring a DS4DB application.

Attributes:

Name	Type	Default value	Req?	Description
`ConfigScanInterval`	int	`0`	N	Determines how often in *seconds* DS4DB should scan its configuration file for changes. If DS4DB detects that the configuration file has changed (by detecting a change to the to the last-modified time of the file), it will reload the file. DS4DB will validate the contents of the modified configuration file against the XML schema. If the XML in the file is incorrect according to the schema, DS4DB will ignore the new configuration and will continue to use the existing one. A value less than 1 disables automatic configuration updating. The default value of 0 therefore means that DS4DB does not scan its configuration file for changes.
`EnableCache`	boolean	`true`	N	If set to `true`, DS4DB caches the results of read queries and will only send an Update message if the relevant data in the cache has actually changed. The message contains just the fields that have changed. If `false`, DS4DB does not cache the results of read queries, so it will always generate Update messages for successful queries. See also Query caching behaviour 33.
`MaxEventParallelism`	int or the string `"unbounded"`	`1`	N	Defines the maximum number of DS4DB events that can execute at the same time. The default value of 1 causes DS4DB's event execution to behave as in previous versions of the software; that is, only one event is executed at a time and events are queued (and subsequently executed) in order of arrival time. When `MaxEventParallelism` is greater than 1, DS4DB will execute events in parallel, up to the maximum defined by this attribute. The value `"unbounded"` means that DS4DB puts no limit on the number of events that may be executed at the same time. However there are likely to be limits imposed by other software and hardware resources – see the note below.
`MaxQueryRetries`	int	`3`	N	The maximum number of attempts to resubmit a query when the DBMS has returned a deadlock error. If the query still fails after this number of retries then DS4DB abandons the event that triggered the query.

Name	Type	Default value	Req?	Description
`QueryRetryPeriod`	int	`0`	N	The period of time in seconds to delay the resubmission of a query if it is to be retried because the DBMS has returned a deadlock error.
`PM4Output`	boolean	`false`	N	If set to `true` this attribute causes DS4DB to behave as a Caplin PriceMaster 4 output handler when it becomes aware of newly created symbols. You only need to set this attribute if you are connecting DS4DB to a PriceMaster peer. *For more information on when to use this attribute please consult Caplin Systems.*

The configuration must contain one and only one instance of this tag, enclosing all the other tags.

The immediate children of `<DataSourceForDatabases>` are:

One or more of

`<DatabaseConfiguration />` 80

followed by at least one of

`<DbRead>` 84

`<DbWrite>` 87

and then at least one of

`<PeerConnectionEvent>` 91

`<RequestEvent>` 92

`<UpdateEvent>` 97

`<TimedEvent>` 96

Note: It is recommended that you turn off the cache (`EnableCache = "false"`) if your DS4DB application communicates with more than one DataSource peer.
See Query caching behavior 33.

Note: Setting the `MaxEventParallelism` attribute to "`unbounded`" could adversely affect system performance as a whole, because DS4DB may consume too many operating system and/or hardware resources when DS4DB events arrive at a high rate.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<DataSourceForDatabases ConfigScanInterval="180" MaxEventParallelism="5"
 EnableCache="true">
   <DatabaseConfiguration />
   <DbRead>...</DbRead>
...<RequestEvent>...</RequestEvent>
</DataSourceForDatabases>
```

For more information see About the <DataSourceForDatabases> tag 32

16.7 <DbRead>

Tag: `<DbRead>`

Defines a read request to be made to the DBMS.

Attributes:

Name	Type	Default value	Req?	Description
`id`	string	(none)	Y	A unique identifier of the read request.
`dbRef`	string	(none)	Y	A reference to the database that the `<DbRead>` should run its query against. The value should be the `dbRef` string in the `<DatabaseConfiguration>` tag that defines the required database.
`prefix`	string	`"/"`	N	A text string to be added to the front (left hand end) of the subject (symbol) name in the DataSource message that is generated as a result of the read operation.
`postfix`	string	empty string	N	A text string to be added to the end (right hand end) of the subject (symbol) name in the DataSource message that is generated as a result of the read operation.
`query`	string	(none)	Y	A text string defining the read query to be sent to the DBMS. Parameters within the query are represented by the '`?`' character; see How to pass parameters to database queries 46.

The immediate children of `<DbRead>` can be

`<SQLParam />` 95 (zero or more)

followed by just one of

`<StandardResultSetProcessor />` 94

`<CompoundNameResultSetProcessor />` 77

`<CollapsingResultSetProcessor />` 75

`<CustomResultSetProcessor />` 79

Note: Because the `dbRef` attribute is mandatory, earlier versions of the configuration XML that do not include this attribute will not work with this release. Edit your old configuration files to include the new `dbRef` attribute in the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags, otherwise DS4DB will report XML schema validation failures at start up and will refuse to load the file.
See Missing dbRef attribute 70 in Troubleshooting 68.

Note: There can only be one SQL query or stored procedure call in the `query` attribute. If there are more then the behaviour is undefined (for example the queries might fail or only be partially executed).

Example:

```
<DbRead id="ALL_INDEXES" dbRef="DB1" prefix="/INDEX/"
                         query="exec stored_proc_a ?" >
   <SQLParam name="ParamListCode" sqltype="VARCHAR" />
   <StandardResultSetProcessor nameColumn="IndexCode" sendName="true" />
</DbRead>
```

16.8 <DbReadRef>

Tag: `<DbReadRef>`

The `<Dbreadref>` tag is used within an event definition tag (`<PeerConnectionEvent>` 91, `<RequestEvent>` 92, `<TimedEvent>` 96, `<UpdateEvent>` 97) to specify a `<DbRead` 84`>` that is to be performed by the event.

Attributes:

Name	Type	Default value	Req?	Description
`ref`	string	(none)	Y	A reference to a database read specification. This is the `id` attribute of a `<DbRead>` tag.

The immediate children of the `<DbReadRef>` are zero or more `<Param />` 90 tags.

If the query specified by the associated `<DbRead>` takes parameters, the actual parameter values to be passed can be defined by corresponding `<Param />` tags inserted as children of the `<DbReadRef>`.

Example:

```
<DbReadRef ref="ALL_INDEXES">
   <Param name="ParamListCode" defaultValue="CAPLIN" />
</DbReadRef>
```

16.9 <DbWrite>

Tag: `<DbWrite>`

Defines a write request to be made to the DBMS.

Attributes:

Name	Type	Default value	Req?	Description
`id`	string	(none)	Y	A unique identifier of the write request.
`dbRef`	string	string	Y	A reference to the database that the `<DbWrite>` should run its query against. The value should be the `dbRef` string in the `<DatabaseConfiguration>` tag that defines the required database.
`query`	string	(none)	Y	A text string defining the write query to be sent to the DBMS. Parameters within the query are represented by the '`?`' character; see How to pass parameters to database queries 46.

The immediate children of `<DbWrite>` can be zero or more instances of the `<SQLParam />` 95 tag.

The SQL statement or stored procedure defined in the query attribute must issue an explicit database commit if the DBMS requires it; DS4DB does not itself issue any commit statements to the DBMS.

Note: Because the `dbRef` attribute is mandatory, earlier versions of the configuration XML that do not include this attribute will not work with this release. Edit your old configuration files to include the new `dbRef` attribute in the `<DatabaseConfiguration>`, `<DbRead>`, and `<DbWrite>` tags, otherwise DS4DB will report XML schema validation failures at start up and will refuse to load the file.
See Missing dbRef attribute 70 in Troubleshooting 68.

Note: There can only be one SQL query or stored procedure call in the `query` attribute. If there are more then the behaviour is undefined (for example the queries might fail or only be partially executed).

Example:

```
<DbWrite id="STOCK_CLOSE" dbRef="DB1" query="exec stored_proc_b ?.?,?,?">
   <SQLParam name="MarketCode" sqltype="VARCHAR"/>
   <SQLParam name="EquityCode"  sqltype="VARCHAR"/>
   <SQLParam name="ClosingPrice" sqltype="NUMERIC"/>
   <SQLParam name="ClosingTime"  sqltype="VARCHAR"/>
</DbWrite>
```

16.10 <DbWriteRef>

Tag: `<DbWriteRef>`

The `<DbWriteRef>` tag is used within an event definition tag (`<PeerConnectionEvent>` 91, `<RequestEvent>` 92, `<TimedEvent>` 96, `<UpdateEvent>` 97) to specify a `<DbWrite>` that is to be performed by the event.

Attributes:

Name	Type	Default value	Req?	Description
`ref`	string	(none)	Y	A reference to a database write specification. This is the id attribute of a `<DbWrite>` tag.

The immediate children of the `<DbWriteRef>` are zero or more `<Param />` 90 tags.

If the query specified by the associated `<DbWrite>` 87 takes parameters, the actual parameter values to be passed can be defined by corresponding `<Param />` 90 tags inserted as children of the `<DbWriteRef>`.

Example:

```
<DbWriteRef ref="STOCK_CLOSE">
   <Param name="MarketCode" source="Identifier" />
   <Param name="StockCode"  source="Field" fieldName="StkCode" />
   <Param name="ClosePrice" />
   <Param name="CloseTime" />
</DbWriteRef>
```

16.11 <Field />

Tag: `<Field />`

Defines a simple field in the messages generated by collapsing result set processing.

Attributes:

Name	Type	Default value	Req?	Description
`name`	string	(none)	Y	The name of a column in the result set whose value is to be inserted as a field in each generated Update message.
`outputName`	string	empty string	N	An alternative name for the field in the generated Update messages. An ordinal number is appended to each occurrence of the field name in each message, for example `Element1`, `Element2`, etc. If the `outputName` attribute is not specified, the `name` attribute is used as the field name.

This tag can only be used as a child tag of `<CollapsingResultSetProcessor>` 75.

It has no child tags.

Example:

```
DbRead id="ALL_ITEMS" prefix="/ITEMS/"  query="exec all_items ?">
   <SQLParam name="ParamListCode" sqltype="VARCHAR" />
   <CollapsingResultSetProcessor
      nameColumn="index_code"
      numberOfRowsColumn="NumRows">
      <Field name="weighting" outputName="Weightg" />
      <CompoundField nameColumns="market_code,ticker"
                     dataFormat="/TICKER/{0}/{1}"
                     outputName="Element" />
   </CollapsingResultSetProcessor>
</DbRead>
```

For more information see How to define Collapsing Result Set Processing 58.

16.12 <Param />

Tag: `<Param />`

Defines how to obtain the value of a parameter required for a database read (`<DbRead>` 84) or database write (`<DbWrite>` 87)

Attributes:

Name	Type	Default value	Req?	Description
`name`	string	(none)	Y	The name of a parameter within the database query specified by a `<DbRead>` or `<DbWrite>`. The name must match the name attribute of a `<SQLParam />` tag enclosed by the `<DbRead>` or `<DbWrite>`.
`source`	string	`"Field"`	N	The location of the parameter value; this takes one of the values: `"Identifier"` or `"Field"`. If `source="Identifier"` the parameter value is taken from the subject (symbol) name in the triggering message. If `source="Field"` (or the `source` attribute is not present) and the `fieldName` attribute is specified, the parameter value is taken from the field in the triggering message that has the same name as the value of the `fieldName` attribute. If `source="Field"` (or the `source` attribute is not present) and the `fieldName` attribute is not specified, the parameter value is taken from the field in the triggering message that has the same name as the value of the `name` attribute.
`fieldName`	string	(none)	N	The name of the message field that contains the value of the parameter. This attribute is only relevant if the `source` attribute is set to `"Field"`.
`defaultValue`	string	(none)	N	The default parameter value to be used if the parameter cannot be found in the incoming message.
`pattern`	string	(none)	N	A regular expression that allows the parameter value to be calculated based on the value obtained from the message. For an explanation of how to do this see Extracting part of a field value to pass as a parameter 50.

This tag is a child of `<DbReadRef>` 86 or `<DbWriteRef>` 88. It has no child tags.

For an explanation of how to use this tag see How to pass parameters to database queries 46.

Example:

```
<DbReadRef ref="ALL_CONSTITUENTS">
   <Param name="ParamListCode" defaultValue="CAPLIN" />
</DbReadRef>
```

16.13 <PeerConnectionEvent>

Tag: `<PeerConnectionEvent>`

Defines a Peer Connection Event. A Peer Connection Event occurs when a connection is made between DataSource for Databases and another DataSource peer.

Attributes: No user definable attributes.

The immediate children of the `<PeerConnectionEvent>` are zero or more `<DbReadRef>` 86 tags followed by an optional `<DbWriteRef>` 88 tag. There must be at least one `<DbReadRef>` tag, or a `<DbWriteRef>` tag.

Note: A Peer Connection Event will fire when *any* DataSource peer connects to DS4DB, so normally there should only be one `<PeerConnectionEvent>` tag in the configuration file.

Example:

```
<PeerConnectionEvent>
   <DbReadRef ref="ALL_INDEXES">
      <Param name="ParamListCode" defaultValue="CAPLIN" />
   </DbReadRef>
</PeerConnectionEvent>
```

For more information see How to define a Peer Connection Event 40.

16.14 <RequestEvent>

Tag: `<RequestEvent>`

Defines a Request Event. A Request event occurs when DS4DB receives a Request message from a DataSource peer (DataSource message type DS_SUBJREQ).

Attributes:

Name	Type	Default value	Req?	Description
`triggerObject`	string	(none)	Y	To trigger this event the incoming DataSource Request message must have this name in the object (subject field). The name is usually a symbol name.
`peerRequest`	string	(none)	N	If this attribute is defined then, whenever a DataSource peer connects to DS4DB, DS4DB will send the newly connected peer a Request message whose subject (symbol) is the value of the attribute. *For more information on when to use this attribute (for example to create a more flexible market data platform) please consult Caplin Systems.*
`ackWhenComplete`	boolean	`false`	N	When this attribute is false: On receipt of the DataSource Request that triggers the Request Event, DS4DB immediately sends back an empty update message as an acknowledgment (DataSource message type DS_DATAUPDATE). This is the same behavior as that exhibited by older versions of DS4DB that do not implement the `ackWhenComplete` attribute. When this attribute is true: DS4DB sends the acknowledgment message only when the event processing is complete. If the event processing completed without error, the acknowledgment is an empty update, and any updates resulting from the event processing follow. If the event processing fails, the acknowledgment is a NoData message (DataSource message type DS_NODATA). Note that when the attribute is false, if the event processing fails no message is sent back to the originator of the DataSource Request to indicate this. However some types of failure will generate an alert message (MAX_QUERY_RETRIES, DATABASE_CONNECTION_UNAVAILABLE – see Alerts raised by DS4DB 99).

The immediate children of the `<RequestEvent>` are zero or more `<DbReadRef` 86`>` tags followed by an optional `<DbWriteRef>` 88 tag. There must be at least one `<DbReadRef>` tag, or a `<DbWriteRef>` tag.

The `triggerObject` attribute can be a regular expression. For example
`<RequestEvent triggerObject="/INDEX/.*">`
means that the Request Event will be triggered by any Request message that has an object name starting with "/INDEXCALC/".

Note: Any parameter values that need to be passed to the database query associated with the Request Event must be transmitted in the message as an extension to the subject name ('symbol'). This approach is taken because Request messages do not contain fields, so it is not possible to transmit parameter values as field values.
For information on how to pass parameters in the subject name see Passing parameters from Request messages 51.

Example:

```
<RequestEvent triggerObject="/EVENTS/INDEX_OPEN">
   <DbReadRef ref="INDEX" />
</RequestEvent>
```

For more information see How to define a Request Event 40.

16.15 <StandardResultSetProcessor />

Tag: `<StandardResultSetProcessor />`

Defines how to apply standard processing to the result set returned by a database read (`<DbRead>` 84).

Attributes:

Name	Type	Default value	Req?	Description
`nameColumn`	string	(none)	Y	Specifies the name of a column in the result set. For each row in the result set the value in this column is appended to the string defined by the `prefix` attribute of the enclosing `<DbRead>` tag, followed by the `<DbRead>` `postfix` string (if any). The resulting string forms the subject (symbol) name of the generated Update message.
`sendName`	boolean	`false`	N	If set to `true`, for each row in the result set, the value in the column specified by `nameColumn` is included as a field in the generated Update message.

This tag has no child tags.

Example:

```
<DbRead id="INDEXES" prefix="/INDEX" query="exec caplin_index_sls ?" >
   <SQLParam name="ParamListCode" sqltype="VARCHAR"/>
   <StandardResultSetProcessor nameColumn="IndexCode" sendName="true"/>
</DbRead>
```

For more information see How to define Standard Result Set Processing 55.

16.16 <SQLParam />

Tag: `<SQLParam />`

A parameter to be passed to a database read request (`<DbRead>` 84) or a database write request (`<DbWrite` 87`>`).

Attributes:

Name	Type	Default value	Req?	Description
`name`	string	(none)	Y	The name of the parameter.
`sqltype`	string	`"VARCHAR"`	N	The SQL type of the parameter. See the list of supported types below.

The supported SQL types are:

`"INTEGER"`

`"FLOAT"`

`"DOUBLE"`

`"DECIMAL"`

`"NUMERIC"`

`"CHAR"`

`"VARCHAR"`

`"LONGVARCHAR"`

`"BOOLEAN"`

Note: This is a *subset* of the standard list of SQL types as defined by the JDBC interface.
In particular there is no date/time data type; dates and times must be dealt with as text strings.

Example:

```
<DbWrite id="STOCK_CLOSE" query="exec stored_proc_b ?.?,?,?">
   <SQLParam name="MarketCode" sqltype="VARCHAR" />
   <SQLParam name="StockCode"  sqltype="VARCHAR" />
   <SQLParam name="ClosePrice" sqltype="NUMERIC" />
   <SQLParam name="CloseTime"  sqltype="VARCHAR" />
</DbWrite>
```

16.17 <TimedEvent>

Tag: `<TimedEvent>`

Defines a Timed Event. This is an event that repeats at a configured interval.

Attributes:

Name	Type	Default value	Req?	Description
`refreshPeriod`	long	`"5000"`	N	The interval between successive timed events in milliseconds.

The immediate children of the `<TimedEvent>` are zero or more `<DbReadRef>` 86 tags followed by an optional `<DbWriteRef>` 88 tag. There must be at least one `<DbReadRef>` tag, or a `<DbWriteRef>` tag.

You can define more than one timed event. There is no limit on the number of timed events that can be configured, other than system resources such as available memory.

Example:

```
<TimedEvent refreshPeriod = "5000">
   <DbReadRef ref="ALL_INDEXES">
      <Param name="ParamListCode" defaultValue="CAPLIN" />
   </DbReadRef>
</TimedEvent>
```

For more information see Configuring a timed event 21 in Setting up a simple configuration 13.

16.18 <UpdateEvent>

Tag: `<UpdateEvent>`

Defines an Update Event. An Update Event occurs when DS4DB receives an Update message from a DataSource peer (DataSource message type DS_DATAUPDATE).

Attributes:

Name	Type	Default value	Req?	Description
`triggerPeer`	int	(none)	N	To trigger this event the DataSource peer that sent the DataSource Update message must have this id number.
`triggerObject`	string	(none)	N	To trigger this event the incoming DataSource Update message must have this name in the object (subject field). The name is usually a symbol name.
`triggerField`	string	(none)	N	To trigger this event the incoming DataSource Update message must contain a field of this name.
`triggerValue`	string	(none)	N	To trigger this event the field specified by the `triggerField` in the incoming DataSource Update message must contain this value.
`peerRequest`	string	(none)	N	If this attribute is defined then, whenever a DataSource peer connects to DS4DB, DS4DB will send the newly connected peer a Request message whose subject (symbol) is the value of the attribute. *For more information on when to use this attribute (for example to create a more flexible market data platform) please consult Caplin Systems.*

The immediate children of the `<UpdateEvent>` are zero or more `<DbReadRef` 86`>` tags followed by an optional `<DbWriteRef>` 88 tag. There must be at least one `<DbReadRef>` tag, or a `<DbWriteRef>` tag.

You can specify any combination of attributes, the net result being a logical AND. For example, if all four attributes are specified this means that for the event to fire:

- the message must have come from the specified peer (`triggerPeer` attribute match), and
- the message must have the specified object name (`triggerObject` attribute match), and
- the specified field name must be present in the message (`triggerField` attribute match), and
- the `triggerField` must have the specified value (`triggerValue` attribute match).

If no attributes are specified then *any* incoming Update message will trigger the Update Event.

The `triggerObject` attribute can be a a regular expression. For example
`<UpdateEvent triggerObject="/INDEXCALC/.*">`
means that the Update Event will be triggered by any Update message that has an object name starting with "`/INDEXCALC/`".

Example:

```
<UpdateEvent triggerObject="/CLOSE/FX/.*" triggerField="bid">
   <DbWriteRef ref="FX_CLOSE">
   ...
   </DbWriteRef>
</UpdateEvent>
```

For more information see How to define an Update Event 41.

17 Alerts raised by DS4DB

When it encounters certain error conditions DataSource for Databases can send a DataSource alert message, which will be received by any DataSource peer that has subscribed to alert messages. For example Caplin Transformer has an alerts module that allows Transformer to receive alerts and forward them to interested parties by email.

Alerts are also recorded in the local DS4DB log file.

The following table lists the DataSource alerts that DS4DB can generate.

DS4DDB Alerts

Alert code	Error Severity	Alert title and message	Description
`DS4DB_RELOAD_FAILED`	ERROR	Title: `DataSourceForDatabases Configuration Change {TIMESTAMP}` Message: The exception that caused the failure in string form.	Sent when a configuration reload fails. This could be because the new configuration is invalid according to the XML schema. The old configuration remains in effect. Also see Dynamic configuration update 32.
`DS4DB_RELOAD_COMPLETED`	INFO	Title:`DataSourceForDatabases Configuration Change {TIMESTAMP}` Message: `DataSourceForDatabases configuration reload begun at {START_TIME}, completed at {END_TIME} (delayed by {DELAY}s)`	`{START_TIME}` is the time when DS4DB detected that the configuration file had changed. `{END_TIME}` is the time when the new configuration was applied. `{DELAY}` is the difference between the `{START_TIME}` and `{END_TIME}` in seconds. Also see Dynamic configuration update 32.
`DS4DB_RELOAD_DELAYED`	INFO	Title:`DataSourceForDatabases Configuration Change {TIMESTAMP}` Message: `Currently unable to reload DataSourceForDatabases configuration {TIMESTAMP}`	Sent when the XML defining a new configuration has been successfully parsed but cannot be applied as the system is busy processing events. These alerts are throttled such that a maximum of one per minute is sent while the reload attempt is delayed. A DS4DB_RELOAD_COMPLETED alert is always sent when the new configuration has finally been applied. Also see Dynamic configuration update 32.
`MAX_QUERY_RETRIES`	ERROR	Title: `Task aborted after {X} retries` where `{X}` is the number of retries. Message: String version of the Event that failed. This lists all nested `<DbRead>`s and `<DbWrite>`s, and their attributes.	The `MaxQueryRetries` attribute of the `<DataSourceForDatabases>` 82 configuration tag defines the maximum number of attempts to resubmit a query ("task") when the DBMS has returned deadlock error. If this limit is reached DS4DB aborts the event attempting to issue query and raises this alert.

Alert code	**Error Severity**	**Alert title and message**	**Description**
`DATABASE_CONNECTION_UNAVAILABLE`	ERROR	Title: `Task failed as database connection unavailable.` Message: String version of the Event that failed. This lists all nested `<DbRead>`s and `<DbWrite>`s, and their attributes.	Sent when a DbRead or DbWrite cannot connect to the required database. If any DbRead or DbWrite within an Event fails, then no DataSource messages are created as a result of any of the successful DbReads in the Event. Since events must complete in their entirety, the Event is put back on the dispatch queue and subsequently retried.

18 Glossary of terms and acronyms

This section contains a glossary of terms and acronyms relating to the DataSource for Databases product.

Term	Definition
API	Application Programming Interface
Active peer	A DataSource application that is configured to respond to DataSource Request messages received from one or more of its DataSource peers, by keeping track of which objects have been requested, and sending to the requesting peers updates just for those objects. This improves performance by reducing network bandwidth See also **broadcast peer**.
Broadcast peer	A DataSource application that is configured to broadcast (push) DataSource messages to all the peers connected to it. See also **active peer**.
DataSource peer	An application that can communicate with another application using the DataSource protocol. DataSource for Databases is a DataSource peer.
Database query	A programmatic request made to a **DBMS** to retrieve data from the database, or insert or modify values in the database. In many DBMS implementations database queries can be packaged as **stored procedures**.
DBMS	Database Management System
DS4DB	DataSource for(4) Databases
JDBC	Java Database Connectivity A Sun Microsystems standard defining how Java applications access data in a database.
JDK	Java Development Kit This includes the Java Runtime Environment (**JRE**)
JMX	Java Management Extensions. A facility within the Java Platform, Standard Edition (J2SE) that provides a standard way of monitoring and managing resources across a network.
JRE	Java Runtime Environment
Liberator	Caplin Liberator is a bidirectional streaming push server that delivers trade messages and market data to and from subscribers over any network, tunneling automatically through proxy servers and firewalls. It communicates with other components of the Caplin Platform using the DataSource protocol.
Peer	See **DataSource Peer**.
Result set	The logical database table (set of rows and columns) returned by a query on a database.
SDK	Software Development Kit

Term	Definition
SQL	Structured Query Language
Stored procedure	An encapsulation of a database query or queries that runs inside a database server. Because it runs on the server a stored procedure usually executes faster than when the client application submits the equivalent query to the server.
Transformer	Caplin Transformer is a real-time business rules engine. It uses the DataSource protocol to subscribe to real-time data from other components of the Caplin Platform, such as DataSource adapters. It then uses Transformer Modules to analyze the data and apply business rules to it, before publishing derived or added-value data in real time back to components such as **Liberator**.

Index

CAPLIN THE REAL TIME WEB COMPANY

Contact Us

Caplin Systems Ltd

Triton Court

14 Finsbury Square

London EC2A 1BR

Telephone: +44 20 7826 9600

Fax: +44 20 7826 9610

www.caplin.com