IBM® DB2® Life Sciences Data Connect

Planning, Installation, and Configuration Guide

Version 8

GC27-1235-00

IBM® DB2® Life Sciences Data Connect

Planning, Installation, and Configuration Guide

Version 8

GC27-1235-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

Contents

About this book

This book contains:

- An introduction to DB2 Life Sciences Data Connect and how it fits into the IBM Life Sciences DiscoveryLink offering, a comprehensive set of software and services tailored to the life sciences
- Installation instructions for DB2 Life Sciences Data Connect
- Instructions for adding data sources to a federated system by registering wrappers. Wrappers are modules that enable you or an application to communicate with a data source using SQL statements.

Technical changes to the text are indicated by a vertical line to the left of the change.

Who should read this book

This book is for administrators who are setting up a federated database environment for life sciences research and development data, and for application programmers who are developing applications for such an environment.

What’s new in Version 8?

New features for DB2 Life Sciences Data Connect Version 8 include:

In General

- Wrapper library names have been updated.
- Costing nickname options for non-relational data sources has been added.

Enhanced query planning for non-relational data sources

DB2 Life Sciences Data Connect wrappers have been re-written to participate in the global query planning process, enhancing the access strategy developed for a query against one of the supported data sources. This new planning function increases the performance of queries sent to non-relational wrappers.

XML wrapper

The XML wrapper has been added. It provides federated access to XML data sources. XML joins the growing list of non-relational wrappers introduced since DB2 Universal Database Version 7 that includes BLAST, Documentum, Excel, and table-structured files.

Table-structured file wrapper

- The TYPE, VERSION, and NODE server options are no longer needed.
- The SORTED nickname option has been added.

Documentum wrapper

- The ALL_VALUES nickname option has been added.
- The following custom functions in Version 7 are now nickname pseudo columns:
  - GET_FILE
  - GET_FILE_DEL
  - GET_RENDITION
  - GET_RENDITION_DEL
  - HITS
  - SCORE
- The RENDITION_FORMAT custom function has been added.

Excel wrapper

- Only one wrapper library file is required for both Excel97 and Excel2000 data sources.
- The TYPE, VERSION, and NODE server options are no longer needed.

Online information

This section provides Web addresses and e-mail addresses related to this product.

http://www.ibm.com/software/data/db2/lifesciencesdataconnect/
DB2 Life Sciences Data Connect product website

http://www.ibm.com/solutions/lifesciences/discoverylink.html
DiscoveryLink website

http://www.ibm.com/solutions/lifesciences/
IBM Life Sciences website

ls@us.ibm.com
IBM Life Sciences email address

Conventions

This book uses these highlighting conventions:

Boldface type
Indicates commands and graphical user interface (GUI) controls (for example, names of fields, names of folders, menu choices).

`Monospace type`
Indicates examples of coding or of text that you type.

Italic type
Indicates variables that you should replace with a value. Italic type also indicates book titles and emphasizes words.

UPPERCASE TYPE
Indicates SQL keywords and names of objects (for example, tables, views, and servers).

How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined as follows:

Read the syntax diagrams from left to right and top to bottom, following the path of the line.

The ▶▶─── symbol indicates the beginning of a statement.

The ───▶ symbol indicates that the statement syntax is continued on the next line.

The ▶─── symbol indicates that a statement is continued from the previous line.

The ───▶◀ symbol indicates the end of a statement.

Required items appear on the horizontal line (the main path).

```
▶▶──STATEMENT──required item──────────────────────────────────────────────▶◀
```

Optional items appear below the main path.

```
▶▶──STATEMENT──┬───────────────┬──────────────────────────────────────────▶◀
               └─optional item─┘
```

If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

```
           ┌─optional item─┐
►►─STATEMENT─┴───────────────┴──────────────────────────────────────►◄
```

If you can choose from two or more items, they appear in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►─STATEMENT──┬─required choice1─┬──────────────────────────────────►◄
              └─required choice2─┘
```

If choosing none of the items is an option, the entire stack appears below the main path.

```
►►─STATEMENT──┬──────────────────┬──────────────────────────────────►◄
              ├─optional choice1─┤
              └─optional choice2─┘
```

If one of the items is the default, it will appear above the main path and the remaining choices will be shown below.

```
              ┌─default choice──┐
►►─STATEMENT──┼─────────────────┼───────────────────────────────────►◄
              ├─optional choice─┤
              └─optional choice─┘
```

An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

```
              ┌─────────────────┐
              ▼                 │
►►─STATEMENT────repeatable item─┴───────────────────────────────────►◄
```

If the repeat arrow contains a comma, you must separate repeated items with a comma.

```
              ┌─,───────────────┐
              ▼                 │
►►─STATEMENT────repeatable item─┴───────────────────────────────────►◄
```

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

Keywords appear in uppercase (for example, FROM). They must be spelled exactly as shown. Variables appear in lowercase (for example, column-name). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sometimes a single variable represents a set of several parameters. For example, in the following diagram, the variable parameter-block can be replaced by any of the interpretations of the diagram that is headed **parameter-block**:

```
►►──STATEMENT──┤ parameter-block ├──────────────────────────────────────►◄
```

parameter-block:

```
├──┬─parameter1──────────────────┬──────────────────────────────────────┤
   └─parameter2───┬─parameter3───┘
                  └─parameter4──┘
```

Adjacent segments occurring between “large bullets” (●) may be specified in any sequence.

```
►►──STATEMENT──item1──●──item2──●──item3──●──item4──────────────────────►◄
```

The above diagram shows that item2 and item3 may be specified in either order. Both of the following are valid:

```
STATEMENT item1 item2 item3 item4
STATEMENT item1 item3 item2 item4
```

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 documentation. You can use any of the following methods to provide comments:

- Send your comments from the Web. You can access the IBM Data Management online readers’ comment form at http://www.ibm.com/software/data/rcf
- Send your comments by e-mail to comments@vnet.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

Chapter 1. What is DB2 Life Sciences Data Connect?

This chapter introduces you to the DB2 Life Sciences Data Connect product, the IBM Life Sciences DiscoveryLink offering, and the general steps involved in setting up a system to query life sciences data.

DB2 Life Sciences Data Connect

IBM® DB2® Life Sciences Data Connect enables a DB2 federated system to integrate genetic, chemical, biological, and other research data from distributed sources. A DB2 federated system is a distributed computing system that consists of a DB2 Universal Database™ server and multiple data sources from which the DB2 Universal Database server retrieves data.

With a federated system, you or an application can use SQL statements to query, retrieve, and join data that can be located in several heterogeneous data sources, such as relational databases from IBM, Oracle, Sybase, and Microsoft, and non-relational data sources, such as table-structured files. Figure 1 illustrates a federated system using DB2 Life Sciences Data Connect to access multiple sources of research data.

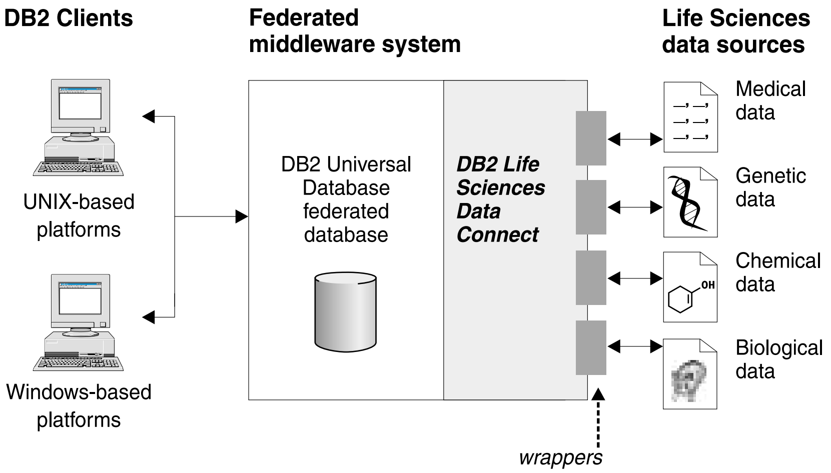

DB2 Clients
Federated middleware system
Life Sciences data sources
UNIX-based platforms
Windows-based platforms
DB2 Universal Database federated database
DB2 Life Sciences Data Connect
Medical data
Genetic data
Chemical data
Biological data
wrappers

Figure 1. Accessing life sciences data with DB2 Life Sciences Data Connect

A DB2 federated system includes clients, a database to which the clients submit queries (called a federated database), an interface through which the federated database communicates with data sources, and the data sources themselves.

The mechanism by which a federated server communicates with a data source is called a *wrapper*. To implement a wrapper, the server uses routines stored in a library called a *wrapper module*. These routines allow the server to perform operations such as connecting to a data source and retrieving data from it iteratively.

After a federated system is set up, the information in data sources can be accessed as though it is in one large database. Users and applications send queries to one federated database, which retrieves data from multiple data sources. Applications work with the federated database just like with any other DB2 database.

For more information on federated systems, see the *DB2 SQL Reference*.

Related concepts:

- "IBM Life Sciences DiscoveryLink" on page 2

IBM Life Sciences DiscoveryLink

The DiscoveryLink offering is a set of middleware software and services tailored specifically to life sciences research and development requirements for integrating data from multiple heterogeneous data sources.

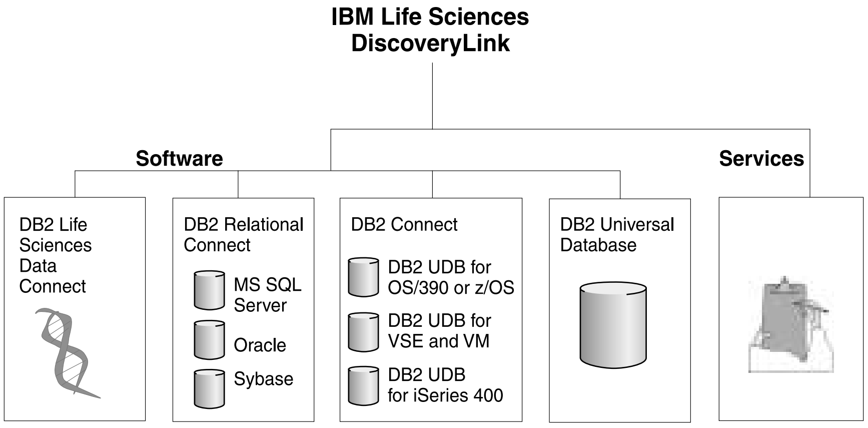


Figure 2. IBM Life Sciences DiscoveryLink

For example, with DiscoveryLink, you can use a single SQL statement to integrate protein sequence data from an Oracle database in Switzerland, chemical structure data from a Sybase database in Japan, and spectroscopic data stored in table-structured flat files on your local area network. The data appears as if it is in one virtual database.

Software components include:

DB2® Life Sciences Data Connect
For accessing life sciences data.

DB2 Relational Connect
For accessing Oracle, Sybase, and Microsoft® relational databases. For more information on DB2 Relational Connect, see the *Federated Systems Guide*.

DB2 Connect™
For accessing DB2 database servers on host systems. For more information on DB2 Connect, see the *DB2 Connect User's Guide*.

DB2 Universal Database™
To optimize queries and integrate results across multiple heterogeneous data sources. For more information on DB2 Universal Database, see the *DB2 Administration Guide*.

For more information on DiscoveryLink software and services, see ″Online information″ in the related links section below.

Related concepts:

- "DB2 Life Sciences Data Connect" on page 1

Chapter 2. Installing DB2 Life Sciences Data Connect

This chapter details the platforms supported for each wrapper, the installation instructions for Unix-based and Windows-based wrappers, and the wrapper libraries placed on your system after installation is complete.

Installing DB2 Life Sciences Data Connect

To use DB2 Life Sciences Data Connect to query and retrieve life sciences data, you must install the wrappers, then register each wrapper to add them to your federated system.

The wrappers have no special requirements beyond those required by DB2 Universal Database and run on any system configuration supported by DB2 Universal Database.

Table 1 shows the DB2 Life Sciences Data Connect wrappers on each operating system. Instructions for registering each life sciences wrapper are provided in the topics listed in the related links section below.

Table 1. DB2 Life Sciences Data Connect wrappers by operating system

Wrapper	**Windows**	**AIX**	**HP-UX**	**Linux**	**Solaris Operating Environment**
Table-structured files	X	X	X	X	X
Documentum	X	X			X
Excel	X				
BLAST	X	X		X	X
XML	X	X	X	X	X

During the installation process, there are three installable components for you to select from: Scientific, Structured files, and Applications. A list of each installable component and the wrappers included in each component is provided in Table 2 on page 6.

Table 2.

Installable component name	Description	Included wrappers
Scientific	Scientific data sources are developed exclusively for the life sciences industry, such as those containing genomic, proteomic, bioinformatic, and cheminformatic information.	BLAST
Structured files	Structured file data sources contain life sciences data stored in files with a defined, repeatable structure.	Table–structured file, Excel, XML
Applications	Application data sources use an application to access the underlying life sciences data. The raw data can be in a number of standard and nonstandard formats.	Documentum

Procedure:

To install DB2 Life Sciences Data Connect, follow these steps:

1. Before you install DB2 Life Sciences Data Connect.
2. Install DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers.
3. Install DB2 Life Sciences Data Connect on Windows servers
4. After you install DB2 Life Sciences Data Connect.

Related tasks:

- "Before installing DB2 Life Sciences Data Connect" on page 7
- "Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers" on page 7
- "Installing DB2 Life Sciences Data Connect on Windows servers" on page 8
- "After installing DB2 Life Sciences Data Connect" on page 10
- "Installing the CreateNicknameFile utility (Documentum wrapper)" on page 57

Before installing DB2 Life Sciences Data Connect

This task is part of the main task for *Installing DB2 Life Sciences Data Connect*.

Procedure:

Before you install DB2 Life Sciences Data Connect on your federated server:

- Confirm that you have DB2 Universal Database Enterprise Server Edition installed on the federated server.
- Make sure that the database has Federated Database System Support turned on. To check this setting, run the following command from the DB2 command line processor:

  ```
  GET DATABASE MANAGER CONFIGURATION
  ```

  This command displays all of the database parameters and their current settings. Confirm that the FEDERATED parameter is set to YES.

  If the FEDERATED parameter is set to NO, run the following command from the DB2 command line processor:

  ```
  UPDATE DATABASE MANAGER CONFIGURATION USING FEDERATED YES
  ```

The next task in this sequence of tasks is *Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers*.

Related tasks:

- “Installing DB2 Life Sciences Data Connect on Windows servers” on page 8
- “After installing DB2 Life Sciences Data Connect” on page 10
- “Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers” on page 7

Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers

This task is part of the main task for *Installing DB2 Life Sciences Data Connect*.

Prerequisites:

See ″Before installing DB2 Life Sciences Data Connect″ in the Related tasks section below.

Procedure:

To install DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment federated servers, use the db2setup utility.

Note: The screens that are displayed when you use the db2setup utility depend on what software products are installed on the federated server. These steps assume that DB2 Life Sciences Data Connect is not installed.

1. Log in as a user with root authority.
2. Insert and mount your DB2 Life Sciences Data Connect CD-ROM. For information on how to mount a CD-ROM, see *DB2 for UNIX Quick Beginnings*.
3. Change to the directory where the CD-ROM is mounted by entering the **cd** */cdrom* command, where *cdrom* is the mount point for your product CD-ROM.
4. Type the following command:

   ```
   ./db2setup
   ```

   The DB2 Setup window opens.
5. Follow the prompts in the setup program.

   When the installation is complete, DB2 Life Sciences Data Connect will be installed in the directory along with your other DB2 products.
   - On DB2 for AIX servers, the directory is /usr/opt/db2_08_01
   - On DB2 for Solaris Operating Environment servers, the directory is /opt/IBM/db2/V8.1
   - On DB2 for HP-UX servers, the directory is /opt/IBM/db2/V8.1
   - On DB2 for Linux servers, the directory is /opt/IBM/db2/V8.1

The next task in this sequence of tasks is *Installing DB2 Life Sciences Data Connect on Windows servers*.

Related tasks:

- "Before installing DB2 Life Sciences Data Connect" on page 7
- "Installing DB2 Life Sciences Data Connect on Windows servers" on page 8
- "After installing DB2 Life Sciences Data Connect" on page 10

Installing DB2 Life Sciences Data Connect on Windows servers

This task is part of the main task for *Installing DB2 Life Sciences Data Connect*.

Prerequisites:

See "Before installing DB2 Life Sciences Data Connect" in the Related tasks section below.

Procedure:

To install DB2 Life Sciences Data Connect on Windows federated servers, use the setup program.

1. Log on to the federated server with the user account that you created to perform the DB2 Universal Database installation.
2. Shut down any programs that are running so that the setup program can update files as required.
3. Invoke the setup program. You can invoke the setup program either automatically or manually. If the setup program fails to start automatically, or if you want to run the setup in a different language, invoke the setup program manually.
   - To automatically invoke the setup program, insert the DB2 Life Sciences Data Connect CD into the drive. The auto-run feature automatically starts the setup program. The system language is determined, and the setup program for that language is launched.
   - To manually invoke the setup program:
     a. Click **Start**, then click **Run**.
     b. In the **Open** field, type the following command:

        ```
        x:\setup /i language
        ```

        where:

        x: Represents your CD-ROM drive.

        language
        Represents the code for your language (for example, EN for English).
     c. Click **OK**.

   The installation launchpad opens.
4. Click **Install** to begin the installation process.
5. Follow the prompts in the setup program.

   When the installation is complete, DB2 Life Sciences Data Connect is installed in the installation directory with other DB2 products. The default installation directory is C:\Program Files\IBM\SQLLIB.

The next task in this sequence of tasks is *After installing DB2 Life Sciences Data Connect*.

Related tasks:

- "Before installing DB2 Life Sciences Data Connect" on page 7
- "After installing DB2 Life Sciences Data Connect" on page 10
- "Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers" on page 7

After installing DB2 Life Sciences Data Connect

This task is part of the main task for *Installing DB2 Life Sciences Data Connect*. After installation, wrapper library files are placed on your system. These libraries are used during the wrapper registration process.

Procedure:

To validate the installation, check the installation directories for the default wrapper libraries.

The default file name for each library, by supported operating system, is listed in Table 3 for Windows platforms, and in Table 4 for UNIX platforms.

Table 3. Default wrapper library names on Windows platforms

Wrapper	Windows
Table-structured files	db2lsfile.dll
Documentum	db2lsdctm.dll
Excel97 / Excel2000	db2lsxls.dll
BLAST	db2lsblast.dll
XML	db2lsxml.dll

Table 4 lists wrapper library name on the supported UNIX platforms.

Table 4. Default wrapper library names by UNIX platform

Wrapper	AIX	HP-UX	Linux	Solaris Operating Environment
Table-structured files	libdb2lsfile.a	libdb2lsfile.sl	libdb2lsfile.so	libdb2lsfile.so
Documentum	libdb2lsdctm.a			libdb2lsdctm.so
BLAST	libdb2lsblast.a		libdb2lsblast.so	libdb2lsblast.so
XML	libdb2lsxml.a	libdb2lsxml.sl	libdb2lsxml.so	libdb2lsxml.so

Note: For Documentum on all platforms except Windows, these libraries are created after they are link–edited to the Documentum client libraries. The files placed on your system after installation are named `libdb2lsSTdctmF.a` on AIX, and `libdb2lsSTdctmF.so` on Solaris Operating Environment.

There are no further tasks in this sequence of tasks.

Related tasks:

- "Before installing DB2 Life Sciences Data Connect" on page 7
- "Installing DB2 Life Sciences Data Connect on Windows servers" on page 8
- "Adding table-structured files to a federated system" on page 16
- "Installing DB2 Life Sciences Data Connect on AIX, HP-UX, Linux, and Solaris Operating Environment servers" on page 7
- "Adding Documentum to a federated system" on page 33
- "Adding Excel to a federated system" on page 71
- "Adding BLAST to a federated system" on page 90
- "Adding XML to a federated system" on page 115

Chapter 3. Table-structured files as data sources

This chapter explains what table-structured files are, how to add them as data sources to your federated system, and lists the error messages associated with the table-structured file wrapper.

What are table-structured files?

A table-structured file has a regular structure consisting of a series of records, where each record contains the same number of fields, separated by an arbitrary delimiter. Null values are represented by two delimiters next to each other.

The following example shows the contents of a file called DRUGDATA1.TXT. It contains three records, each with three fields, separated by commas:

```
234,DrugnameA,Manufacturer1
332,DrugnameB,Manufacturer2
333,DrugnameC,Manufacturer2
```

The first field is the drug's unique ID number. The second field is the name of the drug. The third field is the name of the manufacturer who produces the drug.

Related concepts:

- "Types of table-structured files" on page 13
- "How DB2 Life Sciences Data Connect works with table-structured files" on page 14
- "What is Documentum?" on page 31
- "What is Excel?" on page 69
- "What is BLAST?" on page 85
- "What is XML?" on page 111

Related tasks:

- "Adding table-structured files to a federated system" on page 16

Types of table-structured files

Table-structured files can be sorted or unsorted.

Sorted files

DRUGDATA1.TXT contains sorted records. The file is sorted by the first field, the drug's unique ID number. This field is the primary key because it is unique for each drug. Sorted files must be sorted in ascending order.

```
234,DrugnameA,Manufacturer1
332,DrugnameB,Manufacturer2
333,DrugnameC,Manufacturer2
```

Unsorted files

DRUGDATA2.TXT contains unsorted records. There is no order to the way the records are listed in the file.

```
332,DrugnameB,Manufacturer2
234,DrugnameA,Manufacturer1
333,DrugnameC,Manufacturer2
```

The wrapper can search sorted data files much more efficiently than non-sorted files.

Related concepts:

- "What are table-structured files?" on page 13
- "How DB2 Life Sciences Data Connect works with table-structured files" on page 14

Related tasks:

- "Adding table-structured files to a federated system" on page 16

How DB2 Life Sciences Data Connect works with table-structured files

Using a module called a wrapper, DB2 Life Sciences Data Connect can process SQL statements that query data in a table-structured file as if it were contained in an ordinary relational table or view. This enables data in a table-structured file to be joined with relational data or data in other table-structured files. This process is illustrated in Figure 3 on page 15.

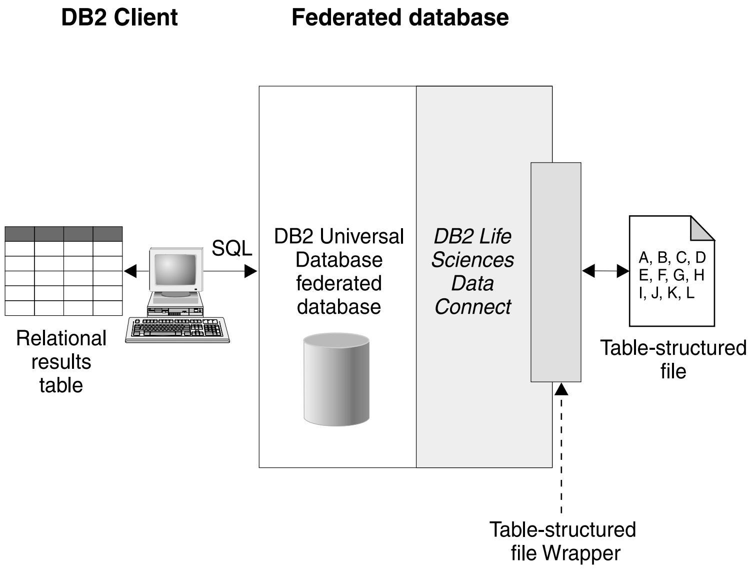


Figure 3. How the table–structured file wrapper works

For example, suppose that the table-structured file `DRUGDATA2.TXT` is located on your computer in your laboratory. Trying to query this data and match it up with other tables from other data sources that you use can be tedious.

After you register `DRUGDATA2.TXT` with DB2 Life Sciences Data Connect, the file behaves as if it is a relational data source. You can now query the file together with other relational and non-relational data sources and analyze the data together.

For example, you could run the following query:

```
SELECT * FROM DRUGDATA2 ORDER BY DCODE
```

This query produces the following results.

Dcode	Drug	Manufacturer
234	DrugnameA	Manufacturer1
332	DrugnameB	Manufacturer2
333	DrugnameC	Manufacturer2

Related concepts:

- “What are table-structured files?” on page 13
- “Types of table-structured files” on page 13

Related tasks:

- "Adding table-structured files to a federated system" on page 16

Adding table-structured files to a federated system

Procedure:

To add a data source for a table-structured file to a federated server:

1. Register the wrapper using the CREATE WRAPPER command.
2. Optional: Set the DB2_DJ_COMM environment variable to improve query performance.
3. Register the server using the CREATE SERVER command.
4. Register nicknames using the CREATE NICKNAME command for all table-structured files.

The commands can be run from the DB2 Command Line Processor.

Related tasks:

- "Registering the table-structured file wrapper" on page 16
- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Registering the server for table-structured files" on page 18
- "Registering nicknames for table-structured files" on page 19
- "Adding Documentum to a federated system" on page 33
- "Adding Excel to a federated system" on page 71
- "Adding BLAST to a federated system" on page 90
- "Adding XML to a federated system" on page 115

Registering the table-structured file wrapper

This task is part of the main task for *Adding table-structured files to a federated system*. You must register the wrapper in order to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from data sources. Wrappers are installed on your system as library files.

Procedure:

To register the wrapper, use the CREATE WRAPPER statement to specify which wrapper will be used to access table-structured files.

For example, to register a wrapper on AIX, run the following statement:

```
CREATE WRAPPER laboratory_flat_files LIBRARY 'libdb2lsfile.a'
  OPTIONS(DB2_FENCED 'N');
```

In this example, `laboratory_flat_files` is the name chosen for the wrapper. This name must be unique within the database in which the wrapper is being registered. The required library name for the table-structured file wrapper on AIX is `libdb2lsfile.a`.

The library name is installed as `libdb2lsfile.a` by default, but it might have been customized during installation. Check with your system administrator for the correct name.

For a table of default library filenames for the table-structured file wrapper by supported platform, see ″After installing DB2 Life Sciences Data Connect″ in the Related tasks section below. For more information on the CREATE WRAPPER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper*.

Related tasks:

- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Registering the Documentum wrapper" on page 36
- "Registering the Excel wrapper" on page 71
- "Registering the BLAST wrapper" on page 95
- "Registering the XML wrapper" on page 116

Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper

This task is part of the main task for *Adding table-structured files to a federated system*. To improve performance when table-structured files are accessed, set the DB2_DJ_COMM environment variable. This variable determines whether the federated server loads the wrapper upon initialization.

Procedure:

To set the DB2_DJ_COMM environment variable, submit the `db2set` command with the wrapper library that corresponds to the wrapper that you specified in the associated CREATE WRAPPER statement.

For example:

```
db2set DB2_DJ_COMM='libdb2lsfile.a'
```

Ensure that there are no spaces on either side of the equal sign (=).

There is overhead associated with loading the wrapper libraries during database startup. To avoid this overhead, only specify libraries you intend to access.

For more information about the DB2_DJ_COMM environment variable, see the *DB2 Administration Guide.*

The next task in this sequence of tasks is *Registering the server for table-structured files.*

Related tasks:

- “Registering the table-structured file wrapper” on page 16
- “Registering the server for table-structured files” on page 18
- “Setting the DB2_DJ_COMM environment variable for the Documentum wrapper” on page 37
- “Setting the DB2_DJ_COMM environment variable for the BLAST wrapper” on page 96
- “Setting the DB2_DJ_COMM environment variable for the XML wrapper” on page 116

Registering the server for table-structured files

This task is part of the main task for *Adding table-structured files to a federated system*. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the table-structured file server to the federated system, use the CREATE SERVER statement. For example:

```
CREATE SERVER biochem_lab WRAPPER laboratory_flat_files
```

In this example, `biochem_lab` is the name assigned to the table-structured file server. The name must be unique to the database in which the server is being registered.

For more information about the CREATE SERVER statement, see the *DB2 SQL Reference.*

The next task in this sequence of tasks is *Registering nicknames for table-structured files*.

Related tasks:

- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Registering nicknames for table-structured files" on page 19
- "Registering the server for Documentum data sources" on page 38
- "Registering the server for an Excel data source" on page 72
- "Registering the server for a BLAST data source" on page 97
- "Registering the server for an XML data source" on page 117

Registering nicknames for table-structured files

This task is part of the main task for *Adding table-structured files to a federated system*. After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to atable-structured file data source in a query.

Procedure:

To register a nickname, use the CREATE NICKNAME statement for each table-structured file that you want to access.

The syntax for the CREATE NICKNAME statement is:

```
                                    ┌─,─────────────────────────────────────────────┐
►►─CREATE NICKNAME─nickname─(───▼─column-name─┤ data-type ├─┤ column-option ├─┴──────►

►─)─FOR SERVER─server-name─OPTIONS─(─FILE_PATH─'path' ───────────────────────────────►


►─┬───────────────────────────────────────┬──┬──────────────────────┬────────────────►
  └─,─COLUMN_DELIMITER─'delimiter' ───────┘  └─,─SORTED─┬─'Y'─┬─────┘
                                                        └─'N'─┘


►─┬────────────────────────────────────────────┬─────────────────────────────────────►
  │  (1)                                       │
  └───────,─KEY_COLUMN─'key-column-name' ──────┘
►─┬─────────────────────────────────────────┬─)──────────────────────────────►◄
  │ (1)                                     │
  └────────,─VALIDATE_DATA_FILE──┬─'Y'─┬────┘
                                 └─'N'─┘
```

data-type:

```
├──┬─SMALLINT──────────────────────────────────────────┬──────────────────────────────┤
   ├─┬─INTEGER─┬───────────────────────────────────────┤
   │ └─INT─────┘                                       │
   ├─FLOAT──┬──────────────────┬───────────────────────┤
   │        └─(─integer─)──────┘                       │
   ├─REAL──────────────────────────────────────────────┤
   ├─DOUBLE──┬─────────────┬───────────────────────────┤
   │         └─PRECISION───┘                           │
   ├─┬─DECIMAL─┬──┬──────────────────────────────────┬─┤
   │ ├─DEC─────┤  └─(─integer──┬─────────────┬──)────┘ │
   │ ├─NUMERIC─┤               └─,─integer───┘         │
   │ └─NUM─────┘                                       │
   ├─┬─CHARACTER─┬──┬───────────────┬──────────────────┤
   │ └─CHAR──────┘  └─(─integer─)───┘                  │
   └─VARCHAR─(─integer─)───────────────────────────────┘
```

column-option:

```
├──┬────────────┬───────────────────────────────────────────────────────────────┤
   └─NOT NULL───┘
```

Notes:

1 Not allowed for unsorted files. Optional for sorted files.

nickname
: A unique nickname for the table-structured file to be accessed. It must be distinct from all other nicknames, tables, and views in the schema in which it is being registered.

column-name
: A unique name given to each field in the table-structured file. Follow each column name with its data type. Only columns of type CHAR, VARCHAR, SMALLINT, INTEGER, FLOAT, DOUBLE, REAL, and DECIMAL are supported.

SMALLINT
: For a small integer.

INTEGER or INT
: For a large integer.

FLOAT(*integer***)**
For a single or double precision floating-point number, depending on the value of *integer*. The value of *integer* must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double precision.

REAL For single precision floating-point.

DOUBLE or DOUBLE PRECISION
For double precision floating-point.

FLOAT
For double precision floating-point.

DECIMAL(*precision-integer, scale-integer***) or DEC(***precision-integer, scale-integer***)**
For a decimal number.

The first integer is the precision of the number; that is, the total number of digits. This value can range from 1 to 31.

The second integer is the scale of the number; that is, the number of digits to the right of the decimal point. This value can range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used.

The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

CHARACTER(*integer***) or CHAR(***integer***) or CHARACTER or CHAR**
For a fixed-length character string of length *integer*, which can range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR(*integer***)**
For a varying-length character string of maximum length *integer*, which can range from 1 to 32672.

NOT NULL
Prevents the column from containing null values.

server-name
Identifies the server you registered using the CREATE SERVER statement. For more information on the CREATE SERVER statement, see the Related Links section below. This server will be used to access the table-structured file.

'path' The fully qualified path to the table-structured file to be accessed, enclosed in single quotation marks. The data file must be a standard file or a symbolic link, rather then a pipe or another non-standard file

type. Data files must be readable by the DB2 instance owner. For more information on instance owners, see the *DB2 Administration Guide*.

SORTED

Specifies whether the data source file is sorted or unsorted. This option accepts either ’Y’, ’y’, ’n’, or ’N’. It has a default value of ’N’.

Note: If you specify that the data source is sorted, it is recommended you set VALIDATE_DATA_FILE to ’Y’.

’delimiter’

The delimiter used to separate columns of the table-structured file, enclosed in single quotation marks. Only single character delimiters are allowed. If no column delimiter is defined, the column delimiter defaults to the comma. A single quote cannot be used as a delimiter. The column delimiter cannot exist as valid data for a column. For example, a column delimiter of a comma cannot be used if one of the columns contains data with embedded commas.

’key-column-name’

The name of the column in the file that forms the key on which the file is sorted, enclosed in single quotation marks. Use this option for sorted files only.

Only single-column keys are supported. The value must be the name of a column defined in the CREATE NICKNAME statement. The column must be sorted in ascending order. If the value is not specified for a sorted nickname, it defaults to the first column in the nicknamed file. It is recommended that the key column be designated not nullable by adding the NOT NULL option to its definition in the nickname statement. For example:

```
CREATE NICKNAME tox (tox_id INTEGER NOT NULL, toxicity VARCHAR(100))
FOR SERVER tox_server1
  OPTIONS (FILE_PATH'/tox_data.txt', SORTED 'Y')

CREATE NICKNAME weights (mol_id INTEGER, wt VARCHAR(100) NOT NULL)
FOR SERVER wt_server
  OPTIONS (FILE_PATH'/wt_data.txt', SORTED 'Y', KEY_COLUMN 'WT')
```

Note: This option is case-sensitive. However, DB2 folds column names to upper case unless the column is defined with double quotes. The following example will not work properly because the `empno` column will be folded to uppercase by DB2, and the `empno` key column will be submitted in lowercase. Thus the column designated as the key will not be found.

```
CREATE NICKNAME depart (
 empno char(6) NOT NULL)
 FOR SERVER DATASTORE
 OPTIONS(FILE_PATH'data.txt', SORTED 'Y', KEY_COLUMN 'empno');
```

VALIDATE_DATA_FILE
For sorted files, this option specifies whether the wrapper verifies that the key column is sorted in ascending order and checks for NULL keys. The only valid values for this option are 'Y' or 'N', enclosed in single quotation marks. The check is done once at registration time. If this option is not specified, then no validation takes place.

The following example shows a CREATE NICKNAME statement for the table-structured file DRUGDATA1.TXT described in ″What are table-structured files?″ listed in the Related Links section below:

```
CREATE NICKNAME DRUGDATA1(Dcode Integer NOT NULL, Drug CHAR(20),
  Manufacturer CHAR(20))
FOR SERVER biochem_lab OPTIONS(FILE_PATH '/usr/pat/DRUGDATA1.TXT',
COLUMN_DELIMITER ',', SORTED 'Y', KEY_COLUMN 'DCODE', VALIDATE_DATA_FILE 'Y')
```

See the *DB2 SQL Reference* for more information about the CREATE NICKNAME statement. For more information about nicknames, see the *DB2 Administration Guide*.

There are no further tasks in this sequence of tasks.

Related tasks:
- "After installing DB2 Life Sciences Data Connect" on page 10
- "Registering the server for table-structured files" on page 18
- "Registering nicknames for Documentum data sources" on page 40
- "Registering nicknames for Excel data sources" on page 73
- "Registering nicknames for BLAST data sources" on page 98
- "Registering nicknames for XML data sources" on page 118
- Chapter 8, "Specifying costing nickname options" on page 133

Wrapper limitations and considerations for the table-structured file wrapper

- Passthru sessions are not allowed when using the wrapper.
- Multi-column keys are not allowed.
- Sorted files must be in ascending order only. Sorting in descending order is not supported.
- The wrapper does not enforce the NOT NULL constraint, but DB2 does. If you create a nickname and attach a NOT NULL constraint on a column and then select a row containing a null value for the column, DB2 will issue a SQL0407N error stating that you can't assign a NULL value to a NOT NULL column.

The exception to this rule is for sorted nicknames. The key column for sorted nicknames cannot be NULL. If a NULL key column is found for a sorted nickname, the SQL1822N error is issued, stating that the key column is missing.

- On DB2 Universal Database Enterprise Server Edition, any table-structured file for which a nickname has been created must be accessible with the same path name from each node. The file does not have to be on a DB2 Universal Database node as long as it can be accessed from any node with a common path.

Related reference:

- "File limitations and considerations for the table-structured file wrapper" on page 24
- "Limitations and considerations for the Documentum wrapper" on page 60
- "Wrapper limitations for the Excel wrapper" on page 77
- "Excel file limitations" on page 78
- "Limitations and considerations for the XML wrapper" on page 126

File limitations and considerations for the table-structured file wrapper

- Files are limited to one record per line.
- Each record must have an equal number of delimited columns.
- Each record must be terminated by the standard line-termination character(s) for the platform on which the wrapper is installed.
- The column delimiter must be consistent throughout the file.
- A null value is represented by two delimiters next to each other or a delimiter followed by a line terminator, if the NULL field is the last one on the line.
- The radix character is determined by the RADIXCHAR item of the LC_NUMERIC National Language Support category.
- Sorted data sources must be sorted in ascending order according to the collation sequence for the current locale, as defined by the settings in the LC_COLLATE National Language Support category.
- The database codepage must match the file's character set; otherwise, you could get unexpected results.
- Files containing multibyte characters are not supported.
- If a non-numeric field is too long for its column type, the excess data is truncated.
- If a decimal field in the file has more digits after the radix char than are allowed by the scale parameter of its column type, the excess data is truncated.
- The maximum line length is 32768.

Related reference:

- "Wrapper limitations and considerations for the table-structured file wrapper" on page 23
- "Limitations and considerations for the Documentum wrapper" on page 60
- "Excel file limitations" on page 78
- "Limitations and considerations for the XML wrapper" on page 126

File access control model for the table-structured file wrapper

The database management system will access table-structured files with the authority of the DB2 instance owner. The wrapper can only access files that can be read by this user ID (or group ID). The authorization ID of the application (the ID that establishes the connection to the federated database) is not relevant.

Related reference:

- "Access control for the Documentum wrapper" on page 62
- "File access control model for the Excel wrapper" on page 78

Optimization tips and considerations for the table-structured file wrapper

- The system can search sorted data files much more efficiently than non-sorted files.
- For sorted files, you can improve performance by specifying a value or range for the key column when submitting a query.
- Statistics for nicknames of table-structured files must be updated manually by updating the SYSSTAT and SYSCAT views. For more information on manually updating SYSSTAT and SYSCAT views, see the *DB2 Administration Guide*.

Related reference:

- "Optimization tips for the BLAST wrapper" on page 108

Messages for the table-structured file wrapper

This section lists and describes messages you might encounter while working with the wrapper for table-structured files. For more information on messages, see the *DB2 Message Reference*.

Table 5. Messages issued by the wrapper for table-structured files

Error Code	Message	Explanation
SQL0405N	The numeric literal ″<literal>″ is not valid because its value is out of range.	A column in the data file, or a predicate value in an SQL statement, contains a value that is out of the possible range for that data type. Correct the data file or redefine the column to a more appropriate type.
SQL0408N	A value is not compatible with the data type of it's assignment target. Target name is ″<column_name>″.	A column in the data file contains characters that are invalid for that data type. Correct the data file or redefine the column to a more appropriate type.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Data source path is NULL″.)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Key Column retrieval failure″.)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″STAT failed on data source. ERRNO = <error_number>″.)	Ensure that you have the proper directory permissions. Ensure that the file exists.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″No column info found″.)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unsupported operator″.)	Contact IBM Software Support.

Table 5. Messages issued by the wrapper for table-structured files (continued)

Error Code	Message	Explanation
SQL1816N	Wrapper ″<wrapper_name>″ cannot be used to access the ″type″ of data source (″<type>″ ″″) that you are trying to define to the federated database.	The server type was invalid. No server type should be specified in the CREATE SERVER statement. Remove the TYPE keyword and value and rerun it.
SQL1822N	Unexpected error code ″ERRNO = <error_number>″ received from data source ″<server_name>″. Associated text and tokens are ″Unable to read file″.	Check the value of the error number. Make sure that the file can be read by the DB2 instance owner. Then rerun the SQL command.
SQL1822N	Unexpected error code ″Data Error″ received from data source ″<server_name>″. Associated text and tokens are ″Data source is a non-standard file″.	The data source file is a directory, socket, or FIFO. Only standard files can be accessed as data source. Change the FILE_PATH option to point to a valid file and reissue the SQL command.
SQL1822N	Unexpected error code ″ERRNO = <error_number>″ received from data source ″<server_name>″. Associated text and tokens are ″File open error″.	The wrapper was unable to open the file. Check the error number to determine why the error occurred. Correct the problem with the data source and reissue the SQL command.
SQL1822N	Unexpected error code ″Data Error″ received from data source ″<server_name>″. Associated text and tokens are ″Key column missing″.	A record retrieved from the data source was missing the key field. The key column must not be null. Correct the data, or register the file with an unsorted nickname.
SQL1822N	Unexpected error code ″Data Error″ received from data source ″<server_name>″. Associated text and tokens are ″File not sorted″.	The file was not sorted on the key column. Do one of the following: change the KEY_COLUMN option to point to the correct column; resort the data file; or register the nickname as an unsorted nickname.

Table 5. Messages issued by the wrapper for table-structured files (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″Data Error″ received from data source ″<server_name>″. Associated text and tokens are ″Key exceeds definition size″.	The key column field read from the data source was larger than the DB2 column definition which could cause the wrapper search routines to function incorrectly. Correct the data or correct the nickname definition, and reregister the nickname.
SQL1822N	Unexpected error code ″Data Error″ received from data source ″<server_name>″. Associated text and tokens are ″Line in data file exceeds 32k″.	A line in the data file exceeded the maximum line length allowed by the wrapper. The line length cannot be greater than 32768. Shorten the length of the line in the data file.
SQL1823N	No data type mapping exists for data type ″<data_type>″ from server ″<server_name>″.	The nickname was defined with an unsupported data type. Redefine the nickname using only supported data types.
SQL1881N	″<option_name>″ is not a valid ″<component>″ option for ″<object_name>″.	The listed value is not a valid option for the listed object. Remove or change the invalid option then resubmit the SQL statement.
SQL1882N	The ″Nickname″ option ″COLUMN_DELIMITER″ cannot be set to ″<delimiter>″ for ″<nickname_name>″.	The column delimiter was more than one character long. Redefine the option with a single character. Then rerun the SQL statement command.
SQL1882N	The ″Nickname″ option ″KEY_COLUMN″ cannot be set to ″<column_name>″ for ″<nickname_name>″.	The column selected as the key column is not defined for this nickname. Correct the KEY_COLUMN option to be one of the sorted columns for this nickname, then reissue the SQL command.
SQL1882N	The ″Nickname″ option ″VALIDATE_DATA_FILE″ cannot be set to ″<option_value>″ for ″<nickname_name>″.	The option value was invalid. Valid values are ″Y″ or ″N″. Correct the option and register the nickname again.
SQL1883N	″<option_name>″ is a required ″<component>″ option for ″<object_name>″.	A required option for the wrapper was missing from the SQL statement. Add the required option and resubmit the SQL statement.

Table 5. Messages issued by the wrapper for table-structured files (continued)

Error Code	Message	Explanation
SQL30090N	Operation invalid for application execution environment. Reason code = ″21″.	You attempted a passthru session. The table-structured file wrapper does not support passthru sessions.

Related reference:

- "Messages for the Documentum wrapper" on page 62
- "Messages for the Excel wrapper" on page 78
- "Messages for the BLAST wrapper" on page 109
- "Messages for the XML wrapper" on page 127

Chapter 4. Documentum as a data source

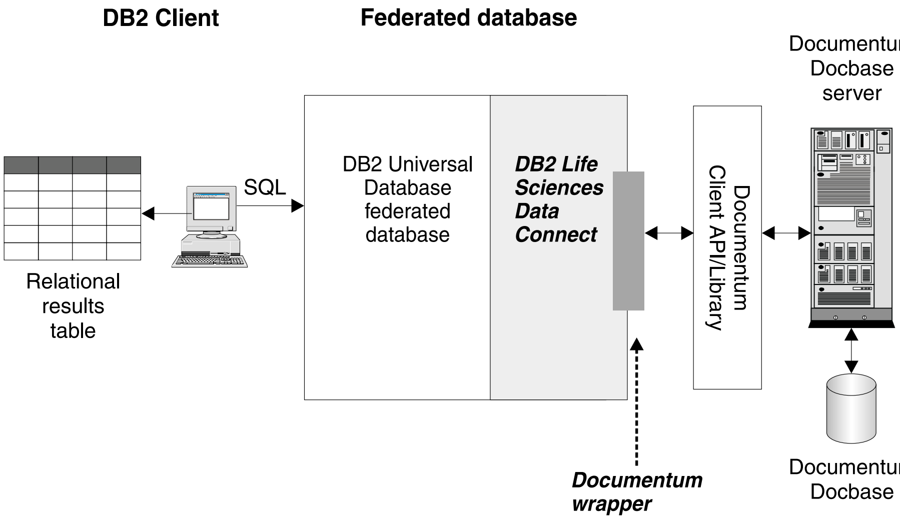
This chapter explains what Documentum is, how to add Documentum data sources to your federated system, and lists the error messages associated with the Documentum wrapper.

What is Documentum?

Documentum is document management software that provides management of document content and attributes such as check-in, check-out, workflow, and version management. The Documentum product is a three-tier, client-server system built on top of a relational database.

A Docbase is a Documentum repository that stores document content, attributes, relationships, versions, renditions, formats, workflow, and security. Documentum Query Language (DQL), an extended SQL dialect, is used to query Documentum data. A Docbase is the equivalent of an Oracle instance or a DB2® database plus document content files. The metadata is stored in the underlying relational database management system (RDBMS), and the content is stored as binary large objects (BLOBs) in the database or as files stored within the file-system of the server system. For more information on Documentum, refer to the Documentum manuals.

The wrapper for Documentum allows you to add a Documentum data source to a DB2 federated system. By adding the Documentum data source to a federated system, you can use SQL statements to access and query objects and registered tables in a Documentum Docbase. You can then integrate this data with other data sources in your federated system without having to move the data out of the native data source. The Documentum wrapper uses a client library to interface with the Documentum server. The Documentum wrapper provides access to two versions of the Documentum server: EDMS 98 (also referred to as version 3) and 4i. Figure 4 on page 32 illustrates how the Documentum wrapper works.



DB2 Client
Federated database
Documentum Docbase server
Relational results table
SQL
DB2 Universal Database federated database
DB2 Life Sciences Data Connect
Documentum Client API/Library
Documentum wrapper
Documentum Docbase

Figure 4. How the Documentum wrapper works

After the Documentum wrapper is registered, you can map Documentum Docbase objects and registered tables as relational tables. This is done by mapping Docbase attributes to column names in a DB2 relational table.

For example, Table 6 lists a subset of attributes for the Documentum Docbase default document type, dm_document, along with the associated data. You have determined that this attribute subset is important to you, and you would like to connect these attributes into your federated database system. You named this subset of data DrugAB_data.

Table 6. DrugAB_data

Title	Subject	Authors	Keywords
The effect of drug A on rabbits	Drug A	Curran, L.	rabbits, drug A
Toxicity results for drug A	Drug A	Abelite, P., McMurtrey, K.	toxicity, drug A
Drug B interactions	Drug B	DeNiro, R., Stone, S.	interactions, drug B
Chemical structure of drug B	Drug B	Boyslim, F.	structure, drug B

After you register the Documentum wrapper, the data can be queried using SQL statements.

The following query displays the titles and authors whose subject is Drug A. The result table is shown in Table 7 on page 33.

```
SELECT title, authors
 FROM drugAB_data
 WHERE subject = 'Drug A'
```

Table 7. Query results

Title	Authors
The effect of drug A on rabbits	Curran, L.
Toxicity results for drug A	Abelite, P., McMurtrey, K.

Related concepts:

- "What are table-structured files?" on page 13
- "What is Excel?" on page 69
- "What is BLAST?" on page 85
- "What is XML?" on page 111

Related tasks:

- "Adding Documentum to a federated system" on page 33

Adding Documentum to a federated system

Procedure:

To add the Documentum data source to a federated server:

1. Link to the Documentum client libraries.
2. Point to Documentum's client dmcl.ini file
3. Register the wrapper using the CREATE WRAPPER statement.
4. Optional: Set the DB2_DJ_COMM environment variable to improve query performance.
5. Register the server using the CREATE SERVER statement.
6. Give users access to the data source by using the CREATE USER MAPPING statement.
7. Register nicknames using the CREATE NICKNAME statement.
8. Create custom functions using the CREATE FUNCTION statement.

The statements can be run from the DB2 Command Line Processor. Once registered, you can run queries against the data source.

Related tasks:

- "Linking to Documentum client libraries (AIX and Solaris Operating Environment only)" on page 34

- "Pointing to Documentum's client dmcl.ini file" on page 35
- "Registering the Documentum wrapper" on page 36
- "Setting the DB2_DJ_COMM environment variable for the Documentum wrapper" on page 37
- "Registering the server for Documentum data sources" on page 38
- "Mapping users (Documentum wrapper)" on page 40
- "Registering nicknames for Documentum data sources" on page 40
- "Registering custom functions for Documentum data sources" on page 49
- "Adding table-structured files to a federated system" on page 16
- "Adding Excel to a federated system" on page 71
- "Adding BLAST to a federated system" on page 90
- "Adding XML to a federated system" on page 115

Linking to Documentum client libraries (AIX and Solaris Operating Environment only)

This task is part of the main task for *Adding Documentum to a federated system*. To enable access to Documentum data sources, the DB2 federated system must be link-edited to the client libraries. The link-edit process creates a wrapper library for each data source with which the federated server communicates. When you run the `djxlinkDctm` script you create the Documentum wrapper library.

Procedure:

To run the djxlinkDctm script:

1. Set the LSDC_DMCL environment variable to point to the directory where the Documentum client library is located.

   For example:

   ```
   export LSDC_DMCL=/usr/documentum/product/3.1.7
   ```
2. Type the following command as root:

   ```
   ksh djxlinkDctm
   ```

Note: The djxlinkDctm command must be rerun after applying a DB2 Universal Database FixPak.

The next task in this sequence of tasks is *Pointing to Documentum's client dmcl.ini file.*

Related tasks:

- "Pointing to Documentum's client dmcl.ini file" on page 35

Pointing to Documentum's client dmcl.ini file

This task is part of the main task for *Adding Documentum to a federated system*. Access to Documentum Docbases are controlled through the Documentum client's dmcl.ini file. A DB2 instance must have its environment variables set to the Documentum client's dmcl.ini file in order to gain access to a Documentum Docbase.

Procedure:

To set the environment variables:

1. Edit the db2dj.ini file, and set one of the following environment variables:

   ```
   DOCUMENTUM=<path>
   DMCL_CONFIG=<path>/dmcl.ini
   ```

   where <path> is the fully qualified directory that contains the dmcl.ini file that you want to use.

   The default path to the location of Documentum's dmcl.ini file is /pkgs/documentum. If both lines are included, DMCL_CONFIG is used.

   On AIX and Solaris Operating Environment, the db2dj.ini file is located in `$HOME/sqllib/cfg`.

   On Windows, the db2dj.ini file is in `x:\sqllib\cfg` where **x:** represents the drive on which the sqllib directory is located.

   Note: Ensure that the name of a docbroker, to which all accessible Docbases for the DB2 instance report, is specified in the dmcl.ini file as shown in Figure 5 on page 36.

```
################# DOCUMENTUM CLIENT CONFIGURATION FILE ######################
#
# Copyright Documentum 1994.
# Version 3.1 of the Documentum Server.
#
# A generated client init file for the Documentum Server.
#
# The only REQUIRED information in this file is the
# [DOCBROKER_PRIMARY] section and an entry for host.
# The host value should be the name of host on which
# your network wide DocBroker is running

[DOCBROKER_PRIMARY]
host = server16.comp2.big.com
```

Figure 5. Sample dmcl.ini file with docbroker name specified

2. Recycle the DB2 instance by issuing the following commands:
   ```
   db2stop
   db2start
   ```

The next task in this sequence of tasks is *Registering the Documentum wrapper*.

Related tasks:

- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Linking to Documentum client libraries (AIX and Solaris Operating Environment only)" on page 34
- "Setting the DB2_DJ_COMM environment variable for the Documentum wrapper" on page 37
- "Setting the DB2_DJ_COMM environment variable for the BLAST wrapper" on page 96
- "Setting the DB2_DJ_COMM environment variable for the XML wrapper" on page 116

Registering the Documentum wrapper

This task is part of the main task for *Adding Documentum to a federated system*. You must register the wrapper in order to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from data sources. Wrappers are installed on your system as library files.

Procedure:

To register the Documentum wrapper, submit the CREATE WRAPPER statement.

For example, to create a Documentum wrapper on AIX called `Dctm_Wrapper` from the default library file, `libdb2lsdctm.a`, submit the following statement:

```
CREATE WRAPPER Dctm_Wrapper LIBRARY 'libdb2lsdctm.a'
  OPTIONS(DB2_FENCED 'N');
```

For a table of default library filenames for the Documentum wrapper by supported platform, see ″After installing DB2 Life Sciences Data Connect″ in the Related tasks section below. For more information on the CREATE WRAPPER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Setting the DB2_DJ_COMM environment variable for the Documentum wrapper*.

Related tasks:

- “After installing DB2 Life Sciences Data Connect” on page 10
- “Registering the table-structured file wrapper” on page 16
- “Setting the DB2_DJ_COMM environment variable for the Documentum wrapper” on page 37
- “Registering the Excel wrapper” on page 71
- “Registering the BLAST wrapper” on page 95
- “Registering the XML wrapper” on page 116

Setting the DB2_DJ_COMM environment variable for the Documentum wrapper

This task is part of the main task for *Adding Documentum to a federated system*. To improve performance when Documentum data sources are accessed, set the DB2_DJ_COMM environment variable. This variable determines whether the federated server loads the wrapper upon initialization.

Procedure:

To set the DB2_DJ_COMM environment variable, submit the `db2set` command with the wrapper library that corresponds to the wrapper that you specified in the associated CREATE WRAPPER statement.

For example:

```
db2set DB2_DJ_COMM='libdb2lsdctm.a'
```

Ensure that there are no spaces on either side of the equal sign (=).

There is overhead associated with loading the wrapper libraries during database startup. To avoid this overhead, only specify libraries you intend to access.

For more information about the DB2_DJ_COMM environment variable, see the *DB2 Administration Guide*.

The next task in this sequence of tasks is *Registering the server for Documentum data sources*.

Related tasks:

- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Registering the Documentum wrapper" on page 36
- "Registering the server for Documentum data sources" on page 38
- "Setting the DB2_DJ_COMM environment variable for the BLAST wrapper" on page 96
- "Setting the DB2_DJ_COMM environment variable for the XML wrapper" on page 116

Registering the server for Documentum data sources

This task is part of the main task for *Adding Documentum to a federated system*. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the Documentum server to the federated system, use the CREATE SERVER statement.

For example, suppose there is a server called Dctm_Server1 for the Dctm_Wrapper wrapper created in the associated CREATE WRAPPER statement. Suppose that server contains a Docbase that runs on AIX and uses Oracle to store data. To register the server, submit the following statement:

```
CREATE SERVER Dctm_Server1
 TYPE   DCTM
 VERSION 3
 WRAPPER Dctm_Wrapper
 OPTIONS( NODE 'Dctm_Docbase',
    OS_TYPE 'AIX',
    RDBMS_TYPE 'ORACLE');
```

Arguments

TYPE Specifies the type of the data source. For Documentum, the type is DCTM. This argument is required.

VERSION
Specifies the version of the data source. For EDMS98, the value is ’3’. For 4i, the value is ’4’. This argument is required.

WRAPPER
Specifies the name of the wrapper associated with this server. This argument is required.

Options

CONTENT_DIR
Specifies the name of the locally-accessible root directory for storing content files retrieved by the GET_FILE, GET_FILE_DEL, GET_RENDITION, and GET_RENDITION_DEL pseudo columns. It must be writable by all users who can use these pseudo columns. Its default value is /tmp. This option is optional.

NODE
Specifies the actual name of the Documentum Docbase. This option is required.

OS_TYPE
Specifies the Docbase server’s operating system. Valid values are AIX, SOLARIS, and WINDOWS. This option is required.

RDBMS_TYPE
Specifies the RDBMS used by the Docbase. Valid values are DB2, INFORMIX, ORACLE, SQLSERVER or SYBASE. This option is required.

TRANSACTIONS
Specifies the server transaction mode. The valid values are:

- NONE — no transactions are enabled.
- QUERY — transactions are enabled only for Dctm_Query methods.
- ALL — transactions are enabled for the Dctm_Query method. ALL has the same function as QUERY in this release.

The default is QUERY. This option is optional.

For more information on the CREATE SERVER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Mapping users (Documentum wrapper)*.

Related tasks:

- “Registering the server for table-structured files” on page 18
- “Setting the DB2_DJ_COMM environment variable for the Documentum wrapper” on page 37

- “Mapping users (Documentum wrapper)” on page 40
- “Registering the server for an Excel data source” on page 72
- “Registering the server for a BLAST data source” on page 97
- “Registering the server for an XML data source” on page 117

Mapping users (Documentum wrapper)

This task is part of the main task for *Adding Documentum to a federated system*. You must map users to the previously defined servers to give them access to the data source.

Procedure:

To map users to your federated servers, use the CREATE USER MAPPING statement.

For example, the following CREATE USER MAPPING statement maps user `Chuck` to user `Charles` on the `Dctm_Server1` server.

```
CREATE USER MAPPING FOR Chuck SERVER Dctm_Server1
OPTIONS(REMOTE_AUTHID 'Charles', REMOTE_PASSWORD 'Charles_pw');
```

You can also define your own user mapping. In the following example, USER is a keyword meaning the current user, not a user named `USER`.

```
CREATE USER MAPPING FOR USER SERVER Dctm_Server1
OPTIONS(REMOTE_AUTHID 'Lisa', REMOTE_PASSWORD 'Lisa_pw')
```

For more information on the CREATE USER MAPPING statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Registering nicknames for Documentum data sources*.

Related tasks:

- “Registering the server for Documentum data sources” on page 38
- “Registering nicknames for Documentum data sources” on page 40

Registering nicknames for Documentum data sources

This task is part of the main task for *Adding Documentum to a federated system*. After you have registered a server and mapped the users to the server, you must register corresponding nicknames. Nicknames are used when you refer to a Documentum data source in a query.

Procedure:

To register nicknames, use the CREATE NICKNAME statement to create a nickname for each Docbase for each object type or registered table of interest.

The syntax for the CREATE NICKNAME statement for Documentum is:

```
                                      ┌─,──────────────────────────────────────────┐
                                      ▼                                            │
►►──CREATE NICKNAME──nickname──(──────column-name──┤ column-information ├──────────┴──)────────►

►──FOR SERVER──server-name──OPTIONS──(──┬──────────────────────────────────┬────────────────────►
                                        └─ALL_VERSIONS──┬─'Y'─┬──,─────────┘
                                                        └─'N'─┘

►──┬───────────────────────────────────┬──┬────────────────────────────────────┬────────────────►
   └─FOLDERS──'folder_string'──,───────┘  └─IS_REG_TABLE──┬─'Y'─┬──,───────────┘
                                                          └─'N'─┘

►──REMOTE_OBJECT──'remote_object_type'──)──────────────────────────────────────────────────────►◄
```

column-information:

```
├──┤ data-type ├──┤ column-option ├──┬──────────────────────────────────┬──────────────────────┤
                                     └──┤ nickname-column-options ├─────┘
```

data-type:

```
├──┬─SMALLINT──────────────────────────────────┬───────────────────────────────────────────────┤
   ├─┬─INTEGER─┬───────────────────────────────┤
   │ └─INT─────┘                               │
   ├─DOUBLE──┬────────────┬────────────────────┤
   │         └─PRECISION──┘                    │
   ├─┬─CHARACTER─┬──┬────────────────────┬─────┤
   │ └─CHAR──────┘  └─(──integer──)──────┘     │
   ├─VARCHAR──(──integer──)────────────────────┤
   ├─DATE──────────────────────────────────────┤
   └─TIMESTAMP─────────────────────────────────┘
```

column-option:

```
├──┬──────────┬────────────────────────────────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

nickname-column-options:

```
├──OPTIONS──(──┬────────────────────────────────┬─┬──────────────────────────┬──────►
               └─REMOTE_NAME──'attribute_name' ─,┘ └─DELIMITER──'delimiter' ─,┘

►─┬──────────────────────────┬─┬────────────────────────┬───────────────────────────┤
  └─IS_REPEATING──┬─'Y'─┬──,─┘ └─ALL_VALUES──┬─'Y'─┬────┘
                  └─'N'─┘                    └─'N'─┘
```

For more information on the CREATE NICKNAME statement, see the *DB2 SQL Reference*.

Column options

NOT NULL
All single-valued columns except those defined as TIMESTAMP and DATE must be defined as NOT NULL. Repeating attributes must not be defined as NOT NULL in nicknames.

Nickname column options

Nickname column option values must be enclosed in single quotation marks.

ALL_VALUES
Specifies that all values of a repeating attribute will be returned, separated by the specified delimiter. If this option is missing or is 'N', then only the last value of a repeating attribute will be returned. As noted under DELIMITER, ALL_VALUES may only be specified for VARCHAR columns for which the IS_REPEATING option is 'Y' (and is invalid when IS_REG_TABLE = 'Y').

DELIMITER
Specifies the delimiter string to be used when concatenating multiple values of a repeating attribute. The delimiter can be one or more characters. The default delimiter is a comma. This option is only valid for attributes of objects with data type VARCHAR where the IS_REPEATING option is set to 'Y'. This option is optional.

IS_REPEATING
Indicates if the column is multi-valued. Valid values are 'Y' and 'N'. The default is 'N'. This option is optional.

REMOTE_NAME
Specifies the name of the corresponding Documentum attribute or column. This option maps remote attribute or column names to local DB2 column names. It defaults to the DB2 column name. This option is optional.

Nickname options

Nickname option values must be enclosed in single quotation marks.

ALL_VERSIONS

Specifies whether all object versions will be searched. The valid values are 'y', 'Y', 'n', and 'N'. The default value of 'N' means that only the current object versions are included in query processing. This option is invalid when IS_REG_TABLE = 'Y'. This option is optional.

FOLDERS

Specifies a string that contains one or more logically-combined and syntactically-correct Documentum FOLDER predicates. Specifying FOLDER predicates restricts the set of documents represented by this nickname to those in the designated folders.

When you specify this option, enclose the entire value of the FOLDERS option in single quotes and use double quotes in place of the single quotes within the string.

For example, if you want to insert:

```
FOLDER('/Tools',DESCEND) OR FOLDER('/Cars')
```

Specify the following FOLDERS option:

```
FOLDERS 'FOLDER("/Tools",DESCEND) OR FOLDER("/Cars")'
```

This option is invalid when IS_REG_TABLE = 'Y'. This option is optional.

IS_REG_TABLE

Specifies whether the object specified by the REMOTE_OBJECT option is a Documentum registered table. The valid values are 'y', 'Y', 'n', and 'N'. The default value is 'N'. This option is optional.

Note: You cannot change a nickname from a Documentum object to a registered table (or back) by changing this option with the ALTER NICKNAME statement. Instead, you must DROP and re-CREATE the nickname.

REMOTE_OBJECT

Specifies the name of the Documentum object type associated with the nickname. The name can be any Documentum object type or registered table. In the case of a registered table, it should be prefixed by the table owner's name. If the registered table belongs to the Docbase owner, dm_dbo can be used for the owner name. This option is required.

Note: Using ALTER NICKNAME to change the value of the REMOTE_OBJECT option will result in errors if the structure of the new object is not similar to that of the original object.

Understanding pseudo columns

The CREATE NICKNAME statement also defines 6 pseudo columns. These columns are used to access object content and other information

The pseudo-columns and their definitions are listed in Table 8.

Table 8. Pseudo column names and definitions.

Pseudo column name	**Definition**
GET_FILE	VARCHAR (255)
GET_FILE_DEL	VARCHAR (255)
GET_RENDITION	VARCHAR (255)
GET_RENDITION_DEL	VARCHAR (255)
HITS	INTEGER
SCORE	DOUBLE

Table 9 lists pseudo columns for SELECT clauses.

Table 9. Pseudo columns for SELECT clauses

Pseudo column name	**Description**

GET_FILE	Retrieves the content file for the current row in addition to the column values.
	
	The extension for the content file is its Documentum format name. If a file of the same name exists, it will be overwritten.
	
	GET_FILE attempts to get the object’s base format. Its value in the row is the object’s a_content_type. Its value is the string ″no_content″ if the object has no content file.
	
	For example:
	`SELECT object_name, DCTM.GET_FILE`
	`FROM ...`
	
	The content file is placed in the server directory that is specified by the Server’s CONTENT_DIR option. It is also placed in a subdirectory named with the user’s DB2 local name. The subdirectory will be created if it doesn’t exist.
	
	It’s extension will be its DOS extension defined in the Docbase for the document’s format type. For example, ″.doc″, for MS Word documents.
	
	Returns the string ″no_content″ or the fully-qualified name of the file.

Table 9. Pseudo columns for SELECT clauses (continued)

Pseudo column name	Description
GET_FILE_DEL	This function is the same as GET_FILE except GET_FILE_DEL first deletes the file retrieved for the previous row, if any, in that query. Returns the string ″no_content″ or the fully-qualified name of the file.

GET_RENDITION	Retrieves the content file of that rendition, a copy of the original document in a different format, for the current row in addition to the column values.
	
	The extension for the content file is its Documentum format name. If a file of the same name exists, it will be overwritten.
	
	To specify the rendition format, a predicate of the form DCTM.RENDITION_FORMAT(<format) = 1 must be specified in the WHERE clause.
	
	For example:
	`SELECT object_name, get_rendition`
	`FROM ...`
	`WHERE DCTM.RENDITION_FORMAT('pdf')=1`
	
	GET_RENDITION attempts to get the named rendition of the object. Its value in the row is the object’s a_content_type, except that its value is the string ″no_content″ if the object has no content file, or the string ″not_found″ if the rendition does not exist.
	
	The content file is placed in the server directory that is specified by the Server’s CONTENT_DIR option. It is also placed in a subdirectory named with the user’s DB2 local name. The subdirectory will be created if it doesn’t exist.
	
	It’s extension will be its DOS extension defined in the Docbase for the document’s format type. For example, ″.doc″, for MS Word documents.
	
	Returns the string ″no_content″, ″not found″, or the fully-qualified name of the file.

GET_RENDITION_DEL	This function is the same as GET_RENDITION except GET_RENDITION_DEL first deletes the file retrieved for the previous row, if any, in that query. Returns the string ″no_content″, ″not found″, or the fully-qualified name of the file.

Table 10 on page 46 lists pseudo columns for SELECT clauses in queries that contain search clauses.

Table 10. Pseudo columns for SELECT clauses in queries that contain search clauses

Pseudo column name	Description
HITS	Returns an integer number that represents the number of places in the document in which the search criteria was matched.
	For example:
	`SELECT r_object_id, object_name, hits`
	`  FROM std_doc`
	`  WHERE DCTM.SEARCH_WORDS ('''workflow'' OR ''flowchart''')=1`
	For each document returned, the number of occurrences of the words ″workflow″ and ″flowchart″ within the document's content are summed and returned as the HITS value.
	The HITS pseudo column is appropriate when the documents have only one content file. This is the typical case. This pseudo column can be used in a WHERE clause qualification for a SELECT statement. However, it must also be specified in the SELECT clause.
SCORE	Returns a document's relevance ranking.
	Use this pseudo column in conjunction with the Documentum's ACCRUE concept operator. Both return a number that indicates how many of the specified words were found in each returned document.
	For example:
	`SELECT object_name, score`
	` FROM std_doc`
	` WHERE`
	`DCTM.SEARCH_TOPIC('<ACCRUE>("document","management","workflow")')=1`
	`  AND SCORE >=75`
	The statement returns all documents that have either two or three of the specified words in their content. If a document has only one of the words, it is assigned a score of 50 and therefore fails the WHERE clause criteria and is not returned. If two of the three words are found, a document is assigned a score of 75. If all three words are found, the document's score is 88.
	The SCORE pseudo column is used for documents that have one content file. This is the typical case.
	SCORE can be in a SELECT clause only if the WHERE contains a SEARCH_WORDS() or SEARCH_TOPIC() function. In a WHERE clause, it is used in conjunction with the ACCRUE concept operator.
	For information on the ACCRUE concept operator, see the Documentum documentation.

CREATE NICKNAME example

The following CREATE NICKNAME statement defines the nickname std_doc. Std_doc is associated with a Documentum Docbase with an object type of dm_document. Table 11 maps the Documentum attributes and data types to DB2 relational column names and data types that are then used to construct the CREATE NICKNAME statement.

Table 11. Mapping of Documentum attributes to DB2 columns for the std_doc nickname

Documentum attribute name	Documentum data type	DB2 column name	DB2 data type	Repeats?	Nullable?
object_name	string(255)	object_name	varchar	No	No
r_object_id	ID	object_id	char(16)	No	No
r_object_type	string(32)	object_type	varchar	No	No
title	string(255)	title	varchar	No	No
subject	string(128)	subject	varchar	No	No
authors	string(32)	author	varchar	Yes	Yes
keywords	string(32)	keyword	varchar	Yes	Yes
r_creation_date	time	creation_date	timestamp	No	Yes
r_modify_date	time	modified_date	timestamp	No	Yes
a_status	string(16)	status	varchar	No	No
a_content_type	string(32)	content_type	varchar	No	No
r_content_size	double	content_size	integer	No	No
owner_name	string(32)	owner_name	varchar	No	Yes

Table 12 describes each Documentum attribute used in the nickname.

Table 12. Description of Documentum attributes for the std_doc nickname

Documentum attribute name	Description
object_name	The user-defined name of the object.
r_object_id	The unique object identifier for this object, set at creation time.
r_object_type	The object’s type, set when the object is created.
title	The user-defined title of the object.
subject	The user-defined subject of the object.
authors	The user-defined list of the authors for the object.
keywords	The list of user-defined keywords for the object.
r_creation_date	The date and time that the object was created.

Table 12. Description of Documentum attributes for the std_doc nickname (continued)

Documentum attribute name	**Description**
r_modify_date	The date and time that the object was last modified.
a_status	Set by server when a router task is forwarded. The value is taken from the values assigned to attached_task_status in the router object.
a_content_type	The file format of the object's content.
r_content_size	The number of bytes in the content. For multi-page documents, this attribute records the size of the first content associated with the document.
owner_name	The name of the object's owner (the user who created the object).

Table 11 on page 47 translates into the following CREATE NICKNAME statement.

```
CREATE NICKNAME std_doc (
  object_name varchar(255) not null,
  object_id char(16) not null OPTIONS(REMOTE_NAME 'r_object_id'),
  object_type varchar(32) not null OPTIONS(REMOTE_NAME 'r_object_type'),
  title varchar(255) not null,
  subject varchar(128) not null,
  author varchar(32) OPTIONS(REMOTE_NAME 'authors', IS_REPEATING 'Y'),
  keyword varchar(32) OPTIONS(REMOTE_NAME 'keywords', IS_REPEATING 'Y'),
  creation_date timestamp OPTIONS(REMOTE_NAME 'r_creation_date'),
  modifed_date timestamp OPTIONS(REMOTE_NAME 'r_modify_date'),
  status varchar(16) not null OPTIONS(REMOTE_NAME 'a_status'),
  content_type varchar(32) not null OPTIONS(REMOTE_NAME 'a_content_type'),
  content_size integer not null OPTIONS(REMOTE_NAME 'r_content_size'),
  owner_name varchar(32))
FOR SERVER Dctm_Server2 OPTIONS (REMOTE_OBJECT 'dm_document', IS_REG_TABLE 'N')
```

After you submit the CREATE NICKNAME statement, you can use the nickname std_doc to query your federated system. You can also join the std_doc nickname with other nicknames and tables in the federated system.

Note: In the catalog, the number of columns for this nickname is 6 more than what is being specified in the CREATE NICKNAME statement due to the pseudo columns.

You can use the CreateNicknameFile utility to automatically map Documentum types to DB2 types and to create an initial CREATE NICKNAME statement. For more information on the CreateNicknameFile utility, see the Related links section below.

The next task in this sequence of tasks is *Registering custom functions for Documentum data sources.*

Related tasks:

- "Registering nicknames for table-structured files" on page 19
- "Mapping users (Documentum wrapper)" on page 40
- "Registering custom functions for Documentum data sources" on page 49
- "Registering nicknames for Excel data sources" on page 73
- "Registering nicknames for BLAST data sources" on page 98
- "Registering nicknames for XML data sources" on page 118
- Chapter 8, "Specifying costing nickname options" on page 133

Registering custom functions for Documentum data sources

This task is part of the main task for *Adding Documentum to a federated system*. You must use the CREATE FUNCTION statement to register several custom functions. You can use these functions to access some of the unique capabilities of Documentum, such as full-text searching and retrieving document content within queries.

Custom functions for predicates are listed in Table 13 on page 50.

DB2 does not support a BOOLEAN data type. Therefore, to create valid SQL statements, the value of each custom function must be explicitly tested. The wrapper implementation only supports the semantics for ″DCTM.<function>(<args>) = 1″ regardless of the test comparison operator specified.

Note: References to the TOPIC function are to Documentum function provided as part of its third-party full-text indexing system from Verity, Inc.

Procedure:

To register custom functions, use the CREATE FUNCTION statement.

All custom functions must be registered with the schema name DCTM. The fully-qualified name of each function is DCTM.<function_name>.

The following example registers the ANY_EQ custom function.

```
CREATE FUNCTION DCTM.ANY_EQ (CHAR(), CHAR()) RETURNS INTEGER
 AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
```

You must register each custom function one time for each DB2 database that has the Documentum wrapper installed.

To assist you in registering custom functions, the sample file, `create_function_mappings.ddl`, is provided in the `sqllib/samples/lifesci` directory. This file contains definitions for each custom function. You can run this ddl file to register the custom functions for each DB2 database that has the Documentum wrapper installed.

Custom function string argument rules

All arguments passed as strings must adhere to the following rules:

- Each string is enclosed in single quotes.
- Single quotes within strings are expressed by two single quotes.

Using custom functions in queries

The following examples illustrate the use of the custom functions in queries.

To display the object name and author from the std_doc nickname for documents that have one or more authors named ’Dave Winters’:

```
SELECT object_name,authors FROM std_doc
WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1
```

To display the object name and author from the std_doc nickname for documents that have one or more authors named ’Dave Winters’ or ’Jon Doe’:

```
SELECT object_name,authors FROM std_doc
WHERE DCTM.ANY_IN(authors,'Dave Winters','Jon Doe')=1
```

To display the object name and r_object_id, and to retrieve the content file, from the std_doc nickname for documents containing strings like ’Dave Win%’ in the authors column:

```
SELECT object_name, r_object_id, get_file FROM std_doc
WHERE DCTM.ANY_LIKE(authors,'Dave Win%')=1
```

Custom function table

Table 13 lists the custom functions for predicates.

Table 13. Custom functions for predicates

Function name	Description
ANY_EQ(arg1, arg2)	Tests a repeating attribute for any value equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1`

Table 13. Custom functions for predicates (continued)

Function name	Description
ANY_NE(arg1, arg2)	Tests a repeating attribute for any value not equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_NE(authors,'Dave Winters')=1`
ANY_LT(arg1, arg2)	Tests a repeating attribute for any value less than the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_LT(num_approvers,4)=1`
ANY_GT(arg1, arg2)	Tests a repeating attribute for any value greater than the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_GT(num_approvers,3)=1`
ANY_LE(arg1, arg2)	Tests a repeating attribute for any value less than or equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_LE(num_approvers,2)=1`

Table 13. Custom functions for predicates (continued)

Function name	Description
ANY_GE(arg1, arg2)	Tests a repeating attribute for any value greater than or equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_GE(num_approvers,1)=1`
ANY_IN(arg1, arg2 – arg11)	Tests a repeating attribute for any of ten values in a specified list of values. Takes 3–11 arguments of the same data type: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2–arg11** Specifies a comma-separated list of values to be compared. For example: `... WHERE DCTM.ANY_IN(authors,'Crick','Watson')=1`

ANY_LIKE(arg1, arg2)	Tests a repeating attribute for any value like the specified value. Takes two required arguments:
	arg1 Specifies the name of a column that represents a repeating attribute.
	arg2 Specifies the pattern being compared with sub-strings in single quotes.
	For example:
	`... WHERE DCTM.ANY_LIKE(authors,'Dave Win%')=1`
	`OR DCTM.ANY_LIKE(keywords,'%\_%')=1`
	Note: The escape clause is not supported in ANY_LIKE() predicates.

Table 13. Custom functions for predicates (continued)

Function name	Description
ANY_NOT_LIKE(arg1, arg2)	Tests a repeating attribute for any value not like the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the pattern being compared with sub-strings in single quotes. For example: `... WHERE DCTM.ANY_NOT_LIKE(authors,'Dave Win%')=1 OR DCTM.ANY_NOT_LIKE(keywords,'%\_%')=1` **Note:** The escape clause is not supported in ANY_NOT_LIKE() predicates.
ANY_NULL(arg)	Tests a repeating attribute for IS NULL. Takes one required argument that is the name of the repeating attribute or single-valued DATE or TIMESTAMP attribute. For example: `... WHERE DCTM.ANY_NULL(authors)=1`
ANY_NOT_NULL(arg)	Tests a repeating attribute for IS NOT NULL. Takes one required argument that is the name of the repeating attribute. For example: `... WHERE DCTM.ANY_NOT_NULL(authors)=1`
ANY_SAME_INDEX(arg1 – arg10)	Tests repeating attributes for values at the same index of each attribute. Takes two to ten of the other ANY_xx() functions. The following example checks whether a document has at least one author named Ken who is not affiliated with UCD. `... WHERE DCTM.ANY_SAME_INDEX( ANY_EQ(author_name,'Ken'), DCTM.ANY_NE(author_affiliation,'UCD'))=1`

Table 13. Custom functions for predicates (continued)

Function name	Description

CABINET(arg) and CABINET_TREE(arg)	Takes one required argument that is the fully-qualified name of a Docbase cabinet.
	
	For example:
	`... WHERE DCTM.CABINET('/Tools')=1`
	`... WHERE DCTM.CABINET_TREE('/MyDocs')=1`
	
	Use multiple instances of CABINET and CABINET_TREE to specify multiple cabinets.
	
	For example:
	`... WHERE DCTM.CABINET('/Tools')=1`
	`OR DCTM.CABINET_TREE('/Parts')=1`

FOLDER(arg) and FOLDER_TREE(arg)	Takes one required argument that is the fully-qualified name of a Docbase folder or cabinet.
	
	For example:
	`... DCTM.FOLDER('/Tools/Drills')=1`
	`... DCTM.FOLDER_TREE('/MyDocs/WhitePapers')=1`
	
	Use multiple instances of FOLDER and FOLDER_TREE to specify multiple folders.
	
	For example:
	`... DCTM.FOLDER('/Tools/Drills')=1`
	`OR DCTM.FOLDER_TREE('/Animals/Horses')=1`

RENDITION_FORMAT (format)	Works with the GET_RENDITION and GET_RENDITION_DEL pseudo columns to establish the format of the rendition to be retrieved. Takes a single character string argument specifying the format.
	
	The following example retrieves a document in PDF format:
	`SELECT get_rendition`
	`FROM ....`
	`WHERE DCTM.RENDITION_FORMAT('pdf')=1`

USER(1)	Compares a value to the Documentum author ID of the current user. Takes a dummy argument that must be 1. For example: `... WHERE approver = DCTM.USER(1)` **Note:** To make the Documentum author ID correspond to the DB2 author ID, use the CREATE USER MAPPING statement. For more information on user mapping, see the Related Links section below.

Table 13. Custom functions for predicates (continued)

Function name	Description

SEARCH_WORDS(arg)	Takes one required string argument that is a list of individual words enclosed in single quotes, separated by AND, OR, or NOT, and using parentheses to control precedence. Words cannot contain white space and must be enclosed in single quotes.
	
	For example:
	`... DCTM.SEARCH_WORDS('''yeast''`
	`AND (''bread'' OR ''cake'')`
	`AND NOT ''wedding''' )=1`

SEARCH_TOPIC(arg)	Takes one required string argument which is a Verity TOPIC query statement that is to be passed to Documentum and Verity verbatim. For example: `... WHERE DCTM.SEARCH_TOPIC('"quick"')=1`

For more information on the CREATE FUNCTION statement, see the *DB2 SQL Reference*.

There are no further tasks in this sequence of tasks.

Related tasks:

- "Registering nicknames for Documentum data sources" on page 40

Running queries against Documentum data sources

After you register the wrapper, you can run SQL queries against the Documentum data source. This section provides several example queries.

Procedure:

To run queries, you use the nickname and the defined nickname columns in your SQL statements in the same manner as you would use a regular table name and table columns.

The following query displays all of the Docbase documents for documents named 'Test Document':

```
SELECT object_name
FROM std_doc
WHERE object_name='Test Document';
```

The following query uses the custom function ANY_EQ to display all the documents where one of the authors is ’Joe Doe’.

```
SELECT object_name
FROM std_doc
WHERE DCTM.ANY_EQ(author,'Joe Doe')=1
```

The following query uses the FOLDER_TREE function and the SEARCH_WORDS function to find all documents in the Approved cabinet that contain the text ″protein″.

```
SELECT object_name
FROM std_doc
WHERE DCTM.FOLDER_TREE('/Approved')=1
      AND DCTM.SEARCH_WORDS('protein')=1
```

The following query uses the GET_FILE pseudo column and the FOLDER_TREE and ANY_IN custom functions to retrieve the name of the files, on the DB2 server, into which the content has been placed for all documents in the Approved cabinet that have any of the authors listed.

```
SELECT object_name, object_id, get_file
FROM std_doc
WHERE DCTM.FOLDER_TREE('/Approved')=1
      AND DCTM.ANY_IN(author, 'Mary Black', 'Joe Carson', 'Peter Miller')=1
```

Related tasks:

- “Running queries against Excel data sources” on page 74
- “Running queries against XML data sources” on page 125

What is the CreateNicknameFile utility for the Documentum wrapper?

You can use a Docbasic utility named CreateNicknameFile, available for free download, to create an ASCII file that contains a complete definition of any Docbase object or registered table. You can edit the output file to:

- Define custom local names for columns and attributes. The local and remote names are initially the names as they are known in the Docbase.
- Delete unwanted columns and attributes. The only predefined Documentum document type (dm_document) has 59 attributes in EDMS98 and 76 attributes in 4i. Most of these contain metadata for low-level document management and application development. Deleting the attributes that are not of interest can make SELECT * SQL statements more useful without impacting performance.
- Add a value for the FOLDERS option to restrict searches against this nickname to particular Documentum folders.
- Change DATE mappings to TIMESTAMP if that is desired. The utility generates a mapping from DQL DATE to DB2® DATE because that seems the most useful.

- Change CHAR mappings to VARCHAR or vice-versa depending on application insight.

You must install the utility in a Docbase and run it from a Documentum Windows® graphical user interface. The files that the utility generates are specific to the Docbase in which it is installed.

Related tasks:

- "Installing the CreateNicknameFile utility (Documentum wrapper)" on page 57
- "Configuring the CreateNicknameFile utility (Documentum wrapper)" on page 58
- "Mapping the DM_ID object type in Documentum registered tables" on page 59

Installing the CreateNicknameFile utility (Documentum wrapper)

The CreateNicknameFile utility can assist you in writing CREATE NICKNAME statements for your Documentum data sources.

Procedure:

To install the utility:

1. Download the CreateNicknameFile utility from the download section of the DB2 Life Sciences Data Connect product website at: http://www.ibm.com/software/data/db2/lifesciencesdataconnect/
2. Use the EDMS98 Workspace graphical user interface or the 4i Desktop Client to import the utility, named CreateNicknameFile.txt. You can import the utility as a procedure type into any Docbase cabinet or folder, and you can give it any name you want.
3. Check the **Can be run by user** box on the properties dialog for the newly imported CreateNicknameFile.txt object.

Related concepts:

- "What is the CreateNicknameFile utility for the Documentum wrapper?" on page 56

Related tasks:

- "Configuring the CreateNicknameFile utility (Documentum wrapper)" on page 58
- "Mapping the DM_ID object type in Documentum registered tables" on page 59

Configuring the CreateNicknameFile utility (Documentum wrapper)

The CreateNicknameFile utility can assist you in writing CREATE NICKNAME statements for your Documentum data sources.

Prerequisites:

You must install the CreateNicknameFile utility before it can be configured. For more information on installing the CreateNicknameFile utility, see ″Installing the CreateNicknameFile utility (Documentum wrapper)″ in the Related tasks section below.

Procedure:

To configure the utility after you install it:

1. Double-click on the utility's icon to run it.
2. Type the Documentum Document/object-type name. The default is dm_document.

   Note: Specify dm_registered as the name if you need to create a nickname file for a registered table. If you specify dm_registered, you will also be prompted for the fully-qualified table name in <owner>.<table_name> format. You can use dm_dbo for the owner name if the table is owned by the Docbase owner (the typical case).

   The utility assumes a naming convention for the names of nicknames for registered tables. The convention is to prefix the table name with ″rt_″ to indicate ″registered table″. You can change the nickname proposed by the utility if you don't want to use this convention.
3. Type the server name associated with the nickname you are creating.
4. Type the name of the nickname.

   The names of nickname should be self-explanatory and must be unique within the DB2 instance. The utility assumes a naming convention of <server_name>.<object_type> because the same <object_type> might need to be defined to multiple servers. You can change the nickname proposed by the utility if you don't want to follow this convention.
5. Type the name of the output file.

   The default is C:\Temp\nickname.txt. The directory to receive the output file must already exist and be writeable by the user running the utility.

After you answer the prompts, the nickname file is created and opens in a text editor.

Related concepts:

- "What is the CreateNicknameFile utility for the Documentum wrapper?" on page 56

Related tasks:

- "Registering nicknames for Documentum data sources" on page 40
- "Installing the CreateNicknameFile utility (Documentum wrapper)" on page 57

Mapping the DM_ID object type in Documentum registered tables

The column definitions created by the CreateNicknameFile utility are compliant with the requirements of the Documentum wrapper, including the correct mapping of each data type to the corresponding DB2 data type. The only exception is that Documentum does not support the DM_ID data type in registered tables. The utility assumes that a column in a registered table is used to contain an object ID if it is defined as a string, is 16 characters long, and has a name ending with ″_id″. In the case of the DM_ID data type, the utility maps the column to the DB2 CHAR(16) data type. In all other cases, all string/varchar columns are mapped to the DB2 VARCHAR data type.

Procedure:

To ensure proper data type mapping:

1. Examine the column data type definitions in the output file created by the CreateNicknameFile utility.
2. If the utility mapped a data type of a Documentum column to an incorrect DB2 data type, change the DB2 data type before using the file to register the nickname to DB2.

Related concepts:

- "What is the CreateNicknameFile utility for the Documentum wrapper?" on page 56

Related tasks:

- "Installing the CreateNicknameFile utility (Documentum wrapper)" on page 57
- "Configuring the CreateNicknameFile utility (Documentum wrapper)" on page 58

Dual defining repeating attributes (Documentum wrapper)

To maximize the query capabilities of the wrapper, each attribute must be defined as its true equivalent DB2 data type. That is, Documentum integers must be defined as DB2 integers and so forth. However, these definitions prevent the return of multiple values for non-VARCHAR repeating attributes. For such columns, only the value at index[0] is returned.

This restriction exists because, whenever possible, the wrapper returns only one row of results per Docbase object. This restriction is an issue only when repeating attributes are selected. However, you can define a second column for the same remote repeating attribute but with a data type of VARCHAR.

This column name would be used in the SELECT list to return all values as a delimiter-separated list of all its values. (Each column's DELIMITER option specifies the delimiter to be used.)

You should standardize the local names of the multi-value columns. You can standardize the local names of each multi-value column by adding a prefix of ″m_″ to the local name of the column that is defined as its true data type.

For example, suppose you have a nickname column of a Documentum repeating attribute called approval_dates defined with the data type TIMESTAMP. You can create a second nickname column called m_approval_dates and define it as a VARCHAR data type. You can then use m_approval_dates in a SELECT list to return all approval dates in a delimiter-separated list.

You do not need to use dual definitions for repeating attributes whose true data type is VARCHAR.

Limitations and considerations for the Documentum wrapper

This section contains a list of limitations and considerations associated with the use of the Documentum wrapper.

- Limitations concerning returning repeating attributes values: Only the last value is returned for
  - non-VARCHAR repeating attributes
  - VARCHAR columns when ALL_VALUES 'N' is specified

  To overcome this limitation, you can create a dual definition for the repeating attribute column. For more information on creating dual definitions for repeating attributes, see the Related Links Section below.

Also, multiple values of repeating attributes defined as VARCHAR are returned as one delimiter—separated string. The delimiter depends on the setting of the DELIMITER nickname option set in the CREATE NICKNAME statement.

- The Passthru capability is not supported.
- For each connection to a DB2 database made by a DB2 application, the Documentum wrapper can support a maximum of 10 simultaneous Documentum sessions, and each such session can simultaneously manage up to 10 Documentum queries. A single DB2 application can have several queries in progrss simultaneously; the lifetime of a query begins when it is submitted to DB2 and ends when the corresponding cursor over the result set is closed. At any given time, across the entire set of queries in progress at that time, the following restrictions must hold:
  - All the nicknames referenced by all the queries must reside at no more than 10 different Documentum servers.
  - No more than 10 nicknames from one Documentum server may be referenced.

  Nicknames mentioned in more than one query, or referenced multiple times in a single query, must be counted once for each time they appear.
- The Documentum wrapper uses Version 3.1.7a for AIX of the client library. If you are using Documentum 4i , you will need to acquire the older version of the client library from Documentum (if it is not already installed).
- Because DB2 does not support the Boolean type, most of the custom functions (except for USER) used in the WHERE clause must do a check for ″=1″ because these functions are defined to return an integer.

  For example,

  ```
  "... WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1"
  ```
- Due to a limitation of DB2, the custom function USER is defined with an integer argument that is not used.
- All servers running against the same instance of DB2 must share the same Documentum dmcl.ini configuration parameters.
- The maximum number of values in an ANY_IN custom function for repeating attributes is 10 for a single statement. However, multiple statements can be OR'd.
- For the ANY_SAME_INDEX custom function the maximum number of tests for values at the same index of repeating attributes is 10. The tests must be AND tests that are evaluated left to right.
- The wrapper has no capabilities that are specific to a particular code page.

Related reference:

- "Wrapper limitations and considerations for the table-structured file wrapper" on page 23
- "File limitations and considerations for the table-structured file wrapper" on page 24
- "Wrapper limitations for the Excel wrapper" on page 77
- "Excel file limitations" on page 78
- "Limitations and considerations for the XML wrapper" on page 126

Access control for the Documentum wrapper

Queries are subject to the user's permissions in the Docbase. Only those documents to which the user has at least read access are included in query results.

Related reference:

- "File access control model for the table-structured file wrapper" on page 25
- "File access control model for the Excel wrapper" on page 78

Messages for the Documentum wrapper

This section lists and describes messages you might encounter while working with the wrapper for Documentum. For more information on messages, see the *DB2 Message Reference*.

Table 14. Messages issued by the wrapper for Documentum

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″sqlno_crule_save_plans [100]:rc (-2144272209) Empty plan list detect″.)	The SQL query submitted to DB2 could not be processed by the wrapper. Correct the syntax and resubmit.

Table 14. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″dmAPI exec failed: [DM_QUERY_E_BAD_QUAL] error: ″The attribute qualifier, A0, for attribute <column_name>, is not a valid qualifier.″″.)	An incorrect Documentum type or registered table was entered for the REMOTE_OBJECT nickname option. Change the nickname to use the correct Documentum object type or registered table.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Invalid null column specified″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Nickname specification is empty″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″The Output object is empty or incomplete″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unexpected number of columns requested″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″No column information found″.)	Internal programming error. Contact IBM Software Support.

Table 14. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unsupported column type requested″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Incorrect Column definition″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Inconsistent type; DB2 request != nickname type″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Output parameter is not NULL″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Query output variable is not NULL″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Invalid timestamp length″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Inconsistent number of columns″.)	Internal programming error. Contact IBM Software Support.

Table 14. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Could not access data when converting values″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Failed to initialize the DMCL client″.)	The Documentum client cannot initialize. Contact your system administrator.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Get_User returned NULL″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Get_Local_User returned NULL″.)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Begin Transaction failed″.)	Documentum reported that begintrans failed. Contact your system administrator.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Input parameter was not NULL″.)	Internal programming error. Contact IBM Software Support.
SQL901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Dctm functions must be like DCTM.function(...) =1″.)	The user did not use =1 as the RHS of the predicate for a Dctm function. Correct the syntax and run the query again.

Table 14. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Invalid column number requested″.)	Internal programming error. Contact IBM Software Support.
SQL1881N	″DELIMITER″ is not a valid ″COLUMN″ option for ″<column-name>″	The DELIMITER option was specified for column <column-name>, but the IS_REPEATING option was not specified.
SQL1882N	The ″SERVER″ option ″RDBMS_TYPE″ cannot be set to ″<option-value>″ for ″<server-name>″.	The value specified for the RDBMS_TYPE server option is invalid. It must be one of the following: DB2, INFORMIX, ORACLE, SQLSERVER or SYBASE.
SQL1882N	The ″SERVER″ option ″TRANSACTIONS″ cannot be set to ″<option-value>″ for ″<server-name>″.	The value specified for the TRANSACTIONS server option is invalid. It must be one of the following: NONE, QUERY, PASSTHRU or ALL.
SQL1882N	The ″NICKNAME″ option ″IS_REG_TABLE″ cannot be set to ″<option-value>″ for ″<nickname>″.	The value specified for the IS_REG_TABLE nickname option is invalid. It must be one of the following: ’Y’ or ’N’.
SQL1882N	The ″NICKNAME″ option ″ALL_VERSIONS″ cannot be set to ″<option-value>″ for ″<nickname>″.	The value specified for the ALL_VERSIONS nickname option is invalid. It must be one of the following: ’Y’ or ’N’.
SQL1882N	The ″SERVER″ option ″OS_TYPE″ cannot be set to ″<option-value>″ for ″<server-name>″	The value specified for the OS_TYPE server option is invalid. It must be: AIX, HPUX, SOLARIS or WINDOWS.
SQL1882N	The ″NICKNAME″ option ″FOLDERS″ cannot be set to ″<option-value>″ for ″<nickname>″	The value specified for the FOLDERS nickname option is invalid. It cannot be specified for a table where IS_REG_TABLE is ’Y’.
SQL1882N	The ″NICKNAME″ option ″VERSIONS″ cannot be set to ″<option-value>″ for ″<nickname>″	The value specified for the VERSIONS nickname option is invalid. It must be one of the following: ’Y’ or ’N’. Moreover, VERSIONS ’Y’ cannot be specified for a table where IS_REG_TABLE is ’Y’.

Table 14. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL30090N	Operation invalid for application execution environment. Reason code = ″Invalid column name, IS_REG_TABLE, or IS_REPEATING specified in nickname″	Check the nickname statement for the correct specification of the IS_REG_TABLE, IS_REPEATING, REMOTE_NAME options, and column names.
SQL30090N	Operation invalid for application execution environment. Reason code = ″db2dj.ini missing DOCUMENTUM or DMCL_CONFIG env var″	The required environment variables are not set. Set them in the db2dj.ini file.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Failed to open log file for debugging″	The log file used for troubleshooting is not accessible. Contact your system administrator.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Only one search condition may be specified″	Only one custom search function may be specified per query.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Failed to create content directory″	Make sure the destination directory is writable by the DB2 agent.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Failed to change permissions on content file″	Make sure the target content directory is writable by the db2 agent.

Related reference:

- “Messages for the table-structured file wrapper” on page 25
- “Messages for the Excel wrapper” on page 78
- “Messages for the BLAST wrapper” on page 109
- “Messages for the XML wrapper” on page 127

Chapter 5. Excel as a data source

This chapter explains what Excel is, how to add Excel data sources to your federated system, and lists the error messages associated with the Excel wrapper.

What is Excel?

An Excel spreadsheet or workbook is a file created using the Microsoft® (MS) Excel application and has a file extension of xls. DB2 Life Sciences Data Connect supports spreadsheets from Excel 97 and Excel 2000. Figure 6 illustrates how the Excel wrapper connects your spreadsheets to your federated system.

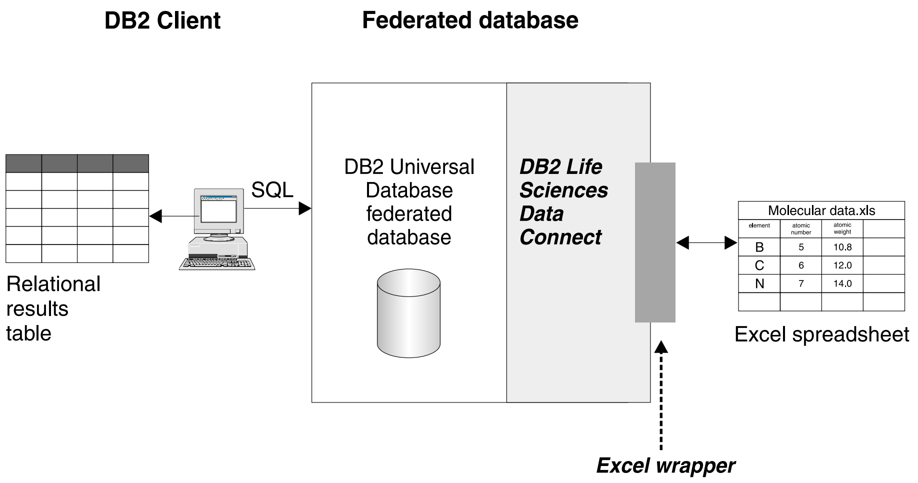


Figure 6. How the Excel wrapper works

The Excel wrapper uses the CREATE NICKNAME statement to map the columns in your Excel spreadsheet to columns in your DB2® federated system. Table 15 shows sample spreadsheet data that is stored in a file called Compound_Master.xls.

Table 15. Sample spreadsheet for Compound_Master.xls

	A	B	C	D
1	compound_A	1.23	367	tested
2	compound_G		210	

Table 15. Sample spreadsheet for Compound_Master.xls (continued)

	A	B	C	D
3	compound_F	0.000425536	174	tested
4	compound_Y	1.00256		tested
5	compound_Q		1024	
6	compound_B	33.5362		
7	compound_S	0.96723	67	tested
8				
9	compound_O	1.2		tested

This information is usually not available to you through standard SQL commands. When the Excel wrapper is installed and registered, you can access this information as if it were a standard relational data source. For example, if you wanted to know all the compound data where the molecular count is greater than 100, you would run the following SQL query:

```
SELECT * FROM compound_master WHERE mol_count > 100
```

The results of the query are shown in Table 16.

Table 16. Query results

COMPOUND_NAME	WEIGHT	MOL_COUNT	WAS_TESTED
compound_A	1.23	367	tested
compound_G		210	
compound_F	0.000425536	174	tested
compound_Q		1024	

Related concepts:

- "What are table-structured files?" on page 13
- "What is Documentum?" on page 31
- "What is BLAST?" on page 85
- "What is XML?" on page 111

Related tasks:

- "Adding Excel to a federated system" on page 71

Related reference:

- "Prerequisite for the Excel wrapper" on page 71

Prerequisite for the Excel wrapper

The prerequisite for utilizing the Excel data source wrapper is:

- The MS Excel application must be installed on the server where DB2 Life Sciences Data Connect is installed before an Excel wrapper can be utilized.

Related tasks:

- "Adding Excel to a federated system" on page 71
- "Registering the Excel wrapper" on page 71

Adding Excel to a federated system

Procedure:

To add the Excel data source to a federated system:

1. Register the wrapper using the CREATE WRAPPER statement.
2. Register the server using the CREATE SERVER statement.
3. Register nicknames using the CREATE NICKNAME statement for each Excel spreadsheet you want to access.

The commands can be run from the DB2 Command Line Processor.

Related tasks:

- "Registering the Excel wrapper" on page 71
- "Registering the server for an Excel data source" on page 72
- "Registering nicknames for Excel data sources" on page 73
- "Adding table-structured files to a federated system" on page 16
- "Adding Documentum to a federated system" on page 33
- "Adding BLAST to a federated system" on page 90
- "Adding XML to a federated system" on page 115

Registering the Excel wrapper

This task is part of the main task for *Adding Excel to a federated system*. You must register the wrapper in order to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from data sources. Wrappers are installed on your system as library files.

Procedure:

To register the Excel data source wrapper, submit a CREATE WRAPPER statement.

To create an Excel wrapper for Excel 97 called `Excel_9x_Wrapper` using the library file `db2lsxls.dll`, submit the following statement:

```
CREATE WRAPPER Excel_9x_Wrapper LIBRARY 'db2lsxls.dll'
  OPTIONS(DB2_FENCED 'N');
```

For more information on the CREATE WRAPPER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Registering the server for an Excel data source*.

Related tasks:

- "Registering the server for an Excel data source" on page 72

Related reference:

- "Prerequisite for the Excel wrapper" on page 71

Registering the server for an Excel data source

This task is part of the main task for *Adding Excel to a federated system*. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the Excel server to the federated system, use the CREATE SERVER statement.

For example, to create a server called `biochem_lab`, with a node name of `biochem_node1` that registers the server for the Excel_2000_Wrapper wrapper created using the CREATE WRAPPER statement, submit the following statement:

```
CREATE SERVER biochem_lab WRAPPER Excel_2000_Wrapper;
```

Argument definitions

WRAPPER
Specifies the name of the wrapper that you registered in the associated CREATE WRAPPER statement. This argument is required.

For more information on the CREATE SERVER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Registering nicknames for Excel data sources*.

Related tasks:

- “Registering the server for table-structured files” on page 18
- “Registering the server for Documentum data sources” on page 38
- “Registering the Excel wrapper” on page 71
- “Registering nicknames for Excel data sources” on page 73
- “Registering the server for a BLAST data source” on page 97
- “Registering the server for an XML data source” on page 117

Registering nicknames for Excel data sources

This task is part of the main task for *Adding Excel to a federated system*. After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to an Excel data source in a query.

Procedure:

To map the Excel data source to relational tables, create a nickname using the CREATE NICKNAME statement.

CREATE NICKNAME syntax (for Excel)

```
                                     ┌─,──────────────────────────────────────────────┐
                                     ▼                                                │
►►──CREATE NICKNAME──nickname──(─────column-name──┤ data-type ├──┤ column-option ├───┴─────►

►──)──FOR SERVER──server-name──OPTIONS──(──FILE_PATH──'path' ──)───────────────────────────►◄
```

data-type:

```
├──┬─┬─INTEGER─┬──────────┬──────────────────────────────────────────────────────────────┤
   │ └─INT─────┘          │
   ├─FLOAT─┬──────────────┤
   │       └─(──integer──)┘
   ├─VARCHAR──(──integer──)──┤
   └─DATE─────────────────┘
```

column-option:

```
├──┬──────────┬──────────────────────────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

For more information on the CREATE NICKNAME statement, see the *DB2 SQL Reference.*

FOR SERVER
Identifies the server that you registered in the associated CREATE SERVER statement. This server is used to access the Excel spreadsheet. Specify the server name.

Option definitions

FILE_PATH
Specifies the fully qualified directory path and file name of the Excel spreadsheet that you want to access.

The statement in the following example creates the Compounds nickname from the Excel spreadsheet file named `CompoundMaster.xls`. The file contains three columns of data that are being defined to the federated system as `Compound_ID`, `CompoundName`, and `MolWeight`.

```
CREATE NICKNAME Compounds (
 Compound_ID INTEGER,
 CompoundName VARCHAR(50),
 MolWeight FLOAT)
FOR SERVER biochem_lab
OPTIONS(FILE_PATH 'C:\My Documents\CompoundMaster.xls');
```

There are no further tasks in this sequence of tasks.

Related tasks:
- "Registering nicknames for table-structured files" on page 19
- "Registering nicknames for Documentum data sources" on page 40
- "Registering the server for an Excel data source" on page 72
- "Registering nicknames for BLAST data sources" on page 98
- "Registering nicknames for XML data sources" on page 118
- Chapter 8, "Specifying costing nickname options" on page 133

Running queries against Excel data sources

This section lists several sample Excel spreadsheet queries using the example nickname `Compounds`.

Procedure:

To run queries, you use the nickname and the defined nickname columns in your SQL statements in the same manner as you would use a regular table name and table columns.

The following query displays all `compound_ID`'s where the molecular weight is greater than 2000:

```
SELECT compound_ID
FROM Compounds
WHERE MolWeight > 200;
```

The following query displays all records where the compound name or molecular weight is null:

```
SELECT *
FROM Compounds
WHERE CompoundName IS NULL
OR MolWeight IS NULL;
```

The following query displays all records where the compound name contains the string `ase` and the molecular weight is greater than or equal to 300:

```
SELECT *
FROM Compounds
WHERE CompoundName LIKE '%ase%
AND MolWeight >=300;
```

Related tasks:

- "Running queries against Documentum data sources" on page 55
- "Sample Excel wrapper scenario" on page 75
- "Running queries against XML data sources" on page 125

Sample Excel wrapper scenario

This section demonstrates a sample implementation of the Excel_2000 wrapper accessing an Excel 2000 spreadsheet located in the `C:\Data` directory. The scenario registers the wrapper, registers a server and registers one nickname, that will be used to access the spreadsheet. The statements shown in the scenario are entered using the DB2 Command Line Processor. After the wrapper is registered, you can run queries against the spreadsheet.

The scenario starts with a compound spreadsheet, called `Compund_Master.xls`, with 4 columns and 9 rows. The fully-qualified path name to the file is `C:\Data\Compound_Master.xls`. The contents are show in Table 17.

Table 17. Sample spreadsheet Compound_Master.xls

	A	B	C	D
1	compound_A	1.23	367	tested
2	compound_G		210	
3	compound_F	0.000425536	174	tested
4	compound_Y	1.00256		tested
5	compound_Q		1024	

Table 17. Sample spreadsheet Compound_Master.xls (continued)

	A	B	C	D
6	compound_B	33.5362		
7	compound_S	0.96723	67	tested
8				
9	compound_O	1.2		tested

Procedure:

To access an Excel 2000 spreadsheet using the Excel wrapper:

1. Register the Excel_2000 wrapper:
   ```
   db2 => CREATE WRAPPER Excel_2000 LIBRARY 'db2lsxls.dll'
      OPTIONS(DB2_FENCED 'N')
   ```
2. Register the server:
   ```
   db2 => CREATE SERVER biochem_lab WRAPPER Excel_2000
   ```
3. Register a nickname that refers to the Excel spreadsheet:
   ```
   db2 => CREATE NICKNAME Compound_Master (compound_name VARCHAR(40),
   weight FLOAT, mol_count INTEGER, was_tested VARCHAR(20))
   FOR biochem_lab
   OPTIONS ( FILE_PATH 'C:\Data\Compound_Master.xls')
   ```

The registration process is complete. The Excel data source is now part of the federated system, and can be used in SQL queries.

The following examples show sample SQL queries and results obtained using the Excel data source.

- Sample SQL query: ″Give me all the compound data where mol_count is greater than 100″
  ```
  SELECT * FROM compound_master WHERE mol_count > 100
  ```
  Result: All fields for rows 1, 2, 3, 5, and 7.
- Sample SQL query: ″Give me the compound_name and mol_count for all compounds where the mol_count has not yet been determined.
  ```
  SELECT compound_name, mol_count FROM compound_master
  WHERE mol_count IS NULL
  ```
  Result: Fields compound_name & mol_count of rows 4, 6 and 9 from the spreadsheet.
- Sample SQL query: ″Count the number of compounds that have not been tested and the weight is greater than 1.″
  ```
  SELECT count(*) FROM compound_master
  WHERE was_tested IS NULL AND weight > 1
  ```

Result: The record count of 1 which represents the single row 6 from the spreadsheet that meets the criteria.

- Sample SQL query: ″Give me the compound_name and mol_count for all compounds where the mol_count has been determined and is less than the average mol_count.″

```
SELECT compound_name, mol_count
FROM compound_master
WHERE mol_count IS NOT NULL
AND mol_count < (SELECT AVG(mol_count) FROM compound_master
                WHERE mol_count IS NOT NULL AND was_tested IS NOT NULL)
```

The sub-query returns the average 368 to the main query which then returns Table 18:

Table 18. Query results

COMPOUND_NAME	**MOL_COUNT**
compound_A	367
compound_G	210
compound_F	174
compound_S	67

Related tasks:

- “Adding Excel to a federated system” on page 71
- “Running queries against Excel data sources” on page 74

Wrapper limitations for the Excel wrapper

- The Excel wrappers are only available for Microsoft Windows operating systems that support DB2 Universal Database Enterprise Server Edition.
- Passthru sessions are not allowed with the Excel wrappers.
- Excel spreadsheet data can only be read not written.
- The wrapper supported date range of the DATE data type is January 1, 1970 to January 18, 2038.

Related reference:

- “Wrapper limitations and considerations for the table-structured file wrapper” on page 23
- “File limitations and considerations for the table-structured file wrapper” on page 24
- “Limitations and considerations for the Documentum wrapper” on page 60
- “Excel file limitations” on page 78
- “Limitations and considerations for the XML wrapper” on page 126

Excel file limitations

- Data types must be consistent within each column and the column data types must be described correctly during the register nickname process.
- The Excel wrappers can only access the primary spreadsheet within an Excel workbook.
- Blank cells in the spreadsheet are interpreted as NULL.
- Up to 10 consecutive blank rows can exist in the spreadsheet and be included in the data set. More than 10 consecutive blank rows are interpreted as the end of the data set.
- Blank columns can exist in the spreadsheet. However, these columns must be registered and described as valid fields even if they will not be used.

Related reference:

- "Wrapper limitations for the Excel wrapper" on page 77

File access control model for the Excel wrapper

The database management system accesses Excel files with the authority of the LOG ON AS property of the DB2 database service. This setting can be viewed in the LOG ON properties page for the DB2 instance. The properties page is accessed through the Windows NT Services control panel.

Related reference:

- "File access control model for the table-structured file wrapper" on page 25
- "Access control for the Documentum wrapper" on page 62

Messages for the Excel wrapper

This section lists and describes messages you might encounter while working with the wrapper for Excel. For more information on messages, see the *DB2 Message Reference*.

Table 19. Messages issued by the wrapper for Excel

Error Code	Message	Explanation
SQL1817N	The CREATE SERVER statement does not identify the ″VERSION″ of data source that you want defined to the federated database.	The VERSION parameter was not specified during the CREATE SERVER statement. Correct the SQL statement and run it again.

Table 19. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″-1000.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Memory allocation error″	Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1001.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Unknown option″.	The option specified in the DDL statement is not supported. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1002.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Creation of DELTA object failed″.	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1100.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Wrapper options are not supported″	Wrapper OPTIONS are not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1200.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″<option> is an unsupported Server option″.	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1201.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error obtaining server name″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1209. <internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″ Error converting VARCHAR data″	An internal program error has occurred. Contact IBM Software Support.

Table 19. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″-1211.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″ Error converting INTEGER data″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1212.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″ Error converting FLOAT data″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1400.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″<option> is an unsupported User option″	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1401.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Creation of USER Delta object failed″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1500.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″<option> is an unsupported Nickname option″	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1501.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Required option PATH not specified″	The PATH option is required to register the NICKNAME. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1502.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Creation of NICKNAME Delta object failed″	An internal program error has occurred. Contact IBM Software Support.

Table 19. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″-1503.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error obtaining Nickname column type″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1504.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error obtaining Nickname column type name″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1505.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″received from data source ″Excel Wrapper″.	The specified <data type> is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code ″-1506.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error obtaining Nickname column info″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1507.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″<option> option cannot be dropped″	The specified option cannot be dropped because it is a required option.
SQL1822N	Unexpected error code ″-1508.VANI″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Column names cannot be altered″	The altering of column names is not permitted by the Excel wrapper.
SQL1822N	Unexpected error code ″-1701.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error parsing SQL″	An internal program error has occurred. Contact IBM Software Support.

Table 19. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″-1702.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error accessing NICKNAME object″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1703.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error building data storage area″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1704.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error linking SQL to Nickname Data″	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1705.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Excel application startup failed″	The startup of the Excel application failed. Confirm that Excel is installed on the system and has been registered with the correct version of the wrapper. Check the LOG ON AS property for the DB2 instance in the Windows NT Services control panel. The Excel application will be accessed using this authority. Confirm that this user has appropriate rights or change this property to an authorized account, then restart DB2 and run the SQL query again.
SQL1822N	Unexpected error code ″-1706.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error opening source spreadsheet″	A problem occurred while opening the spreadsheet referenced by the nickname in the SQL query. Ensure that the file exists in the PATH specified during the CREATE NICKNAME statement during registration.

Table 19. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″-1707.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error accessing DL output storage area″	An internal program error occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code ″-1708.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Excel application end failed″	An internal program error occurred. If this error persists after repeated queries, contact IBM Software Support.
SQL1822N	Unexpected error code ″-1711.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Error during fetch, possible data/col type mismatch″	The data fetched during the SQL query was of a different data type than the data type specified during the registration of the nickname. Correct the data in the source spreadsheet or correct the registered data type in the nickname. If this does not correct the problem, contact IBM Software Support.
SQL1822N	Unexpected error code ″-1900.<internal program code>″ received from data source ″Excel Wrapper″. Associated text and tokens are ″Memory allocation error″	An internal program error has occurred. Contact IBM Software Support.

Related reference:

- “Messages for the table-structured file wrapper” on page 25
- “Messages for the Documentum wrapper” on page 62
- “Messages for the BLAST wrapper” on page 109
- “Messages for the XML wrapper” on page 127

Chapter 6. BLAST as a data source

This chapter explains what BLAST is, how to add BLAST data sources to your federated system, and lists the error messages associated with the BLAST wrapper.

What is BLAST?

BLAST (Basic Local Alignment Search Tool) is a utility that is maintained by the National Center for Biotechnology Information (NCBI). BLAST is used to scan a nucleotide or amino acid sequence database for ″hits.″ A BLAST hit contains one or more high-scoring segment pairs (HSPs). A HSP is a pair of sequence fragments, whose alignment is locally maximal, and whose similarity score exceeds some threshold value. NCBI provides an executable, blastall, that is used to perform BLAST searches on BLAST-able data sources, such as GenBank and SWISS-PROT.

The BLAST wrapper supports all five types of BLAST searches: BLASTn, BLASTp, BLASTx, tBLASTn, and tBLASTx. These are described in Table 20.

Table 20. BLAST search types supported by the BLAST wrapper

BLAST search type	Description
BLASTn	A type of BLAST search in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTp	A type of BLAST search in which an amino acid sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTx	A type of BLAST search in which a nucleotide sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence. The query sequence is translated in all six reading frames, and each of the resulting sequences is used to search the sequence database.

Table 20. BLAST search types supported by the BLAST wrapper (continued)

BLAST search type	Description
tBLASTn	A type of BLAST search in which an amino acid sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence. The sequences in the sequence database are translated in all six reading frames, and the resulting sequences are searched for regions homologous to regions of the query sequence.
tBLASTx	A type of BLAST search in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence. In a tBLASTx search, both the query sequence and the sequence database are translated in all six reading frames, and the resulting sequences are compared to discover homologous regions.

Figure 7 shows how BLAST works with your federated system.

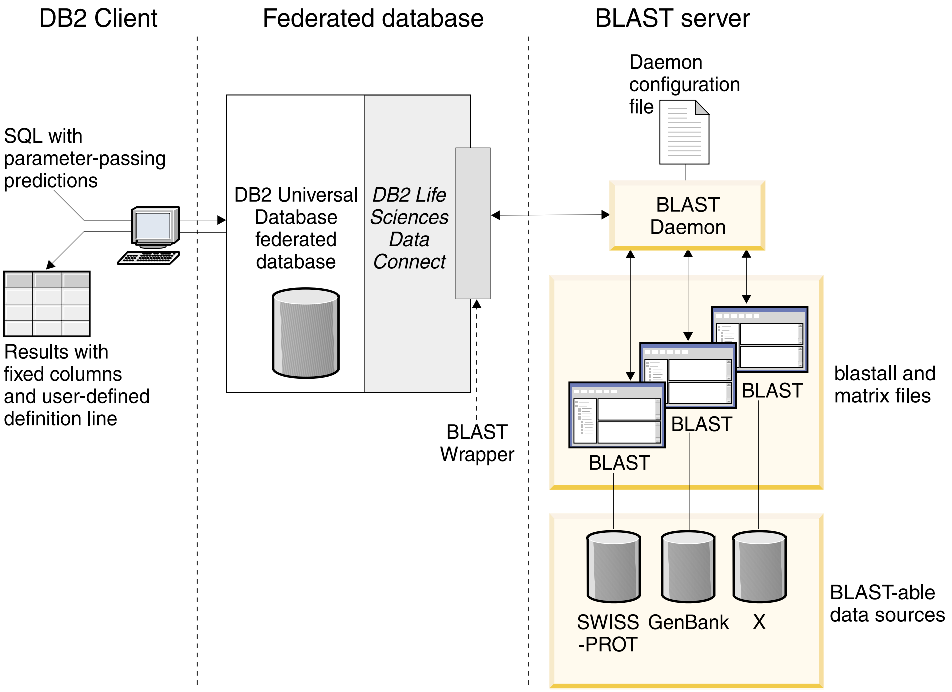

DB2 Client
Federated database
BLAST server
SQL with parameter-passing predictions
Results with fixed columns and user-defined definition line
DB2 Universal Database federated database
DB2 Life Sciences Data Connect
BLAST Wrapper
Daemon configuration file
BLAST Daemon
BLAST
BLAST
BLAST
blastall and matrix files
SWISS -PROT
GenBank
X
BLAST-able data sources

Figure 7. How the BLAST wrapper works

On the client side, users or applications submit SQL statements with BLAST-specific parameter-passing predicates that map to standard BLAST options. The SQL statements with the input predicates are sent to your DB2® Universal Database federated database system with the BLAST wrapper installed.

The BLAST wrapper transforms the query into a format understandable by the BLAST application and sends the transformed query to your BLAST server. This server can be a separate machine from the machine with the federated system. A special daemon program runs on your BLAST server. This daemon, using information from a daemon configuration file, receives the query request from the federated system and sends it to the BLAST application. The BLAST application then runs against a BLAST-able data source in the usual manner.

The results are returned to BLAST and then to the daemon. The daemon returns the retrieved data to the BLAST wrapper. The wrapper transforms the data into a relational table format, and returns this table to the user or application. The returned data contains two parts:

- A series of standard, fixed columns familiar to BLAST users, and
- User-configured definition line information.

The following example illustrates how relational information is extracted from BLAST-able data sources. Data moves from raw fasta file format to a BLAST-able data set to a relational table that can be joined with other data sources in your federated system.

Figure 8 on page 88 is a sample fasta file containing four definition line and nucleotide sequence records.

```
>7:4986 PMON5744
GTTCTTCCCAGTGCCCAAGTCCATTCTGACATCAATGAAGAAGGTAAAATCCCTGCGTGATCCCTCTGCC
AAGATGTCGAAATCAGACCCGGATAAACTAGCTGCTGTCAGAATAACAGACAGCCCGGAGGAGATCGTGC
AGAAGTTCCGCAAGGCTGTGACGGACTTCACCTCGGAGGTCACCTACGACCCGGCCAGGCGAGGAGGCGT
GTCCAACTTGGTGGCCATCCACGCGGCAGTGACCGGACTCCCGGTGGAGGAGGTGGTCCGCCGAAGTGCT
GGCATCAACACCGCTGGCTACAAGTTGGTGGTGGCGGAGGCTGTGATTGAGAGATTTGCACCAATTAAGA
GTGAAATTGAAAAACTGAAGAGGAACAAGGACCACCTAGAGAAGGTTTTACAAGTTGGGTCGGCAAAAGC
CAAAGAATTAGCATATCCCGTGTGCCAGGAGGTGAAGAAATTGGTGGGGTTTCTATAGGCAGTCTCACCT
AGTCCCAGAAAATGTTTTTTATCTTGTGGTCTGCTTGCACACTCAGTCTAATAAAGGCAGCTTTCCTAAG
ACGCCAACAATTCCAGTTTGGGGATGCTTAGTTTACT
>8:9747 PMON5699
AAGAAGTTCTTGTTAGAACTTTCCACCTCCGGCTTCCCCTCCACCTCTCTTACTGTCCCAACCTTCTGAG
ACGCTTTTTCTCCTCCCGAGGATTTATCTCTTTCTCTCTCTCTCTCTCTCTCTTTTTTTTTTTTCCCCT
TTTCCCCCCCCCGAGGCTGGTTTTGCTTTGGGGAGGGGGGGTTTTTTAAAGGGGCCGGGGGGGCCCCCTTT
CTCCCCCCTAATGGGGTTAATTAATAATGGGGGGGGGGGTTTTTTTTTTTAAACCCCTATTTGGTCCGG
CCCGGGGATTTCCCCCCCCCCCCCCTTGCCCGGTTCCGGGGCCCGGAGGAGGGGGGGAAAAGGGCGGGAA
CCTTTGGTAGTTTCCCCTCGGAAAAAAATTTTTCGGGGGGGAAAACCTCCCT
>13:6512 PMON5498
GATAAGAGGCAGAATAGAAGACTGGACTACTTCTCTCCTAAAAACACATTTAAAACTAAGCCTGAGCAAT
CTCCACCCAAATGGACCGGAAACCTTAAAAAAGAATCCTACTCCTGAAGAAAAAGAGGAGGACACATCAA
GAGGTAGAAGGGGCGATTTCATGATATAAACAACCCCATACCTCCAGAGTGGGAAGCTCCACAGACTGAA
AACTAACTGGTTCACAGAAACTCACCTACAGGAGTGAGCCCCACATCAAACCCTCGAATGTGGGGATCTG
GCACTGGTAGAAAGAGCCCCTGGAGCATCTGGCATTGAAGGCCAGTGGGGCTTGTGTGCAGGAGATCCAC
AGGACTAGGGGAAACGGAGACCCCCATTCTTAAAAGGTGCACACAGACTTTTACGTGCACTGGGTCCCAG
TGCAAAGCAAAGTCTCCATAGGAATCTGGGTCAAACCTGACTGCAGTTCTTGGAGGACCTCCTGGGAAAG
CAAGGGTGAATGTGGCTTCTTGTGGGGAAAGGACATTGGAAGCAAAGCTCTTGGGAATATTCATCAGTGT
GC
>15:8924 PMON5426
GGAGAAACTGACTCCTGAGCAGCTGCAATTCATGCGGCAGGTGCAGCTCGCCCAGTGGCAGAAGACGCTG
CCACAGCGGCGGACCCGGAACATCGTGACCGGCCTGGGCATCGGGGCGCTGGTGTTGGCAATTTGTATCC
GTTTGGACTGTAGACTCAGGGAGACCGCATTTAGGGGAACAGGAAGGGCAGCAGGGGCGTGTAGGAGGGC
AGTGTGGGGGTGGTAGAAGGAGCCCGAGATATGAAAACCTTGGCTCCTTTTAACTCTGAATCAAGCGTTT
GGTGTACCTTACGTTGTCATTTTAAAGGTGTATTTTAGTATAATTGATTAATGATTACGGAGTCGGGTGA
GGGCTCCCAGGAGCAGACGGCAGAAGATCGAATTTGGGAGGATGATCAGCAGCGGTGGTTGAGCAAGTGT
GGGAAAAGGGAATGCGCACATTCCACGTGGTTTCCTGAACCCACCTCCCCAGATGGTTACACCTTCTACT
CGGTGTCCCAGGAGCGTTTCTTGGATGAGCTGGAGGATGAGGCCAAAGCTGCTC
```

Figure 8. Sample fasta file, nucleo1

The standard formatdb application transforms the fasta file to a BLAST-able data set. The data is now ready for querying by SQL through a federated system with the BLAST wrapper installed and registered.

The following query, sent by the user or an application at the client end, is transformed by the BLAST wrapper. It then runs against the BLAST-able data set.

```
SELECT Unique_ID, Experiment_Number, Organism_Number, HSP_Info, Score
FROM nucleo1
WHERE BlastSeq = 'ACATTCTTATAGAGTATTGCTACTCCTCCAGGATAGAGTCATCTCT
 GGTCTCCAGAGCCACCGCTGGCTACAAGTTGGTGGTGGCGGAGGCTGTGATTGAGAGATTTG
 CACCAATACAGAAACTCACCTACAGGAGTGAGCGGGTGGTAGAAGGAGCCCGAGATATGAAA
 ACCTTGTTTCAAGACCCCATTGTCACCGGGG';
```

The results of the query are transformed by the BLAST wrapper into a relational table format shown in Table 21.

Table 21. BLAST returns results in relational table form when integrated into your federated system

Unique ID	Experiment number	Organism number	HSP_INFO	SCORE
PMON5744	4986	7	Identities = 57/201 (28%), Positives = 57/201 (28%), Gaps = 0/201 (0%)	+1.13487000000000E+002
PMON5426	8924	15	Identities = 35/201 (17%), Positives = 35/201 (17%), Gaps = 0/201 (0%)	+6.98754000000000E+001
PMON5498	6512	13	Identities = 26/201 (13%), Positives = 26/201 (13%), Gaps = 0/201 (0%)	+5.20342000000000E+001

The data is in a fully relational form and can be joined with data from other data sources used by your laboratory. Combining the results of several data sources can lead to insights not readily or efficiently discovered prior to the implementation of your federated system.

Related concepts:

- "What are table-structured files?" on page 13
- "What is Documentum?" on page 31
- "What is Excel?" on page 69
- "What is XML?" on page 111

Related tasks:

- "Adding BLAST to a federated system" on page 90

Adding BLAST to a federated system

Procedure:

To add the BLAST data source to a federated server:

1. Verify that the correct version of the blastall executable and matrix files are installed.
2. Configure the BLAST daemon.
3. Start the BLAST daemon.
4. Register the wrapper using the CREATE WRAPPER statement.
5. Optional: Set the DB2_DJ_COMM environment variable to improve query performance.
6. Register the server using the CREATE SERVER statement.
7. Register nicknames using the CREATE NICKNAME statement.

The statements can be run from the DB2 command line processor. After the BLAST wrapper is added to your federated system, you can run queries against the BLAST data source.

Related tasks:

- "Verifying that the correct version of the blastall executable and matrix files are installed" on page 91
- "Configuring the BLAST daemon" on page 91
- "Starting the BLAST daemon" on page 94
- "Registering the BLAST wrapper" on page 95
- "Setting the DB2_DJ_COMM environment variable for the BLAST wrapper" on page 96
- "Registering the server for a BLAST data source" on page 97
- "Registering nicknames for BLAST data sources" on page 98
- "Adding table-structured files to a federated system" on page 16
- "Adding Documentum to a federated system" on page 33
- "Adding Excel to a federated system" on page 71
- "Adding XML to a federated system" on page 115

Verifying that the correct version of the blastall executable and matrix files are installed

This task is part of the main task for *Adding BLAST to a federated system*. Verify that you have the latest version of the blastall executable and BLOSUM62, BLOSUM80, PAM30, and PAM70 matrix files installed on your BLAST server machine. If you don't, you must install the binary files and the matrix files. The matrix files must be in the same directory as the blastall executable.

Procedure:

To check the version level of your blastall executable and matrix files:

1. Run a BLAST search from the command line and note the version number located in the output file.
2. If you do not have the lastest version of the blastall executable and matrix files, download the files from the NCBI website: ftp://ftp.ncbi.nih.gov/blast/executables.

The next task in this sequence of tasks is *Configuring the BLAST daemon*.

Related tasks:

- "Configuring the BLAST daemon" on page 91

Configuring the BLAST daemon

This task is part of the main task for *Adding BLAST to a federated system*. The BLAST wrapper requires a BLAST daemon to be running on your UNIX-based machine accessible via TCP/IP from your DB2 Universal Database federated system. The daemon runs separately from the wrapper and DB2 Universal Database and listens for BLAST job requests from the wrapper. The daemon executable file, `db2blast_daemon`, can reside in any directory on the BLAST server machine.

During DB2 Universal Database installation, the daemon executable is placed in the `/usr/opt/db2_08_01/bin` directory on AIX, and in the `/opt/IBM/db2/V8.1/bin` directory on the other Unix platforms, of the machine on which the federated server is being installed. If, in your environment, BLAST runs on a different machine, you must copy the daemon to a location of your choice on that machine.

The BLAST daemon must have:

- Execute access to the blastall binary file so that it can run BLAST searches.
- Write access to a directory in which it can write temporary files.

- Read access to at least one BLAST-able data source against which BLAST searches can be run. The blastall executable must have read access to both the data file and the BLAST index files generated by the formatdb program.

The BLAST daemon requires a configuration file. A sample daemon configuration file, named `BLAST_DAEMON.config`, is placed in the directory `DB2PATH/samples/lifesci`, where `DB2PATH` is the directory in which DB2 Universal Database is installed. `BLAST_DAEMON.config` is the default name for the file.

Copy the configuration file to any location accessible to the daemon, rename it if you want, and edit it to work with your data source. By default the blast_daemon looks for its configuration information in the working directory from which it was started.

Procedure:

To configure the daemon, specify the following options in the configuration file. For options requiring paths, you can specify relative paths. Relative paths are relative to the directory from which the daemon process was started.

DAEMON_PORT
: This is the network port on which the daemon will listen for BLAST job requests submitted by the wrapper.

MAX_PENDING_REQUESTS
: This is the maximum number of BLAST job requests that can be blocking on the daemon at any one time. This number does not represent the number of BLAST jobs that are running concurrently, only the number of job requests that can block at one time. It is recommended that you set this to a number greater than five. The BLAST daemon does not restrict the number of BLAST jobs that can run concurrently.

DAEMON_LOGFILE_DIR
: This is the directory in which the daemon will create its log file. This file will contain useful status and error information generated by the BLAST daemon.

Q_SEQ_DIR_PATH
: This is the directory in which a temporary query sequence data file will be created by the daemon. This temporary file is cleaned up once the BLAST job completes.

BLAST_OUT_DIR_PATH
: This is the directory in which the daemon will create the temporary file to store the BLAST output data. Data will be read from this file

and passed back to the wrapper via the network connection, at which point the daemon cleans up the temporary file.

BLASTALL_PATH
This is the fully-qualified name of the BLAST executable file on the machine running the daemon.

database specification entry
Specifies the location of a BLAST-able data source. For the daemon to function properly, you must specify each entry name used in the configuration file in the DATASOURCE option of the CREATE NICKNAME statement when you create the nickname for the data source. For more information on the CREATE NICKNAME statement, see ″Registering nicknames for BLAST data sources″ in the Related tasks section below.

The configuration file must contain at least one database specification entry in the following form:

```
entry_name = path to BLAST-able_data_source
```

For example, to specify the GenBank BLAST-able data source, you would add the following line to the daemon configuration file:

```
genbank=/dsk/1/nucl_data/genbank
```

The path indicated in a database specification entry must contain the following files:

- The original fasta-formatted data
- The three index files.
  - For nucleotide data sources, the index files have these extensions:
    - .nhr
    - .nin
    - .nsq
  - For amino acid data sources, the index files have these extensions:
    - .phr
    - .pin
    - .psq

The database specification entry must indicate the file name of the file that contains the original Fasta-formatted data. The three index files must have the same root file name as the file containing the original Fasta-formatted data.

The configuration file must end with a newline character.

Example:

The following example shows the contents of a sample configuration file, with the required options and BLAST-able data source specification for GenBank and SWISS-PROT.

```
DAEMON_PORT=4007
MAX_PENDING_REQUESTS=10
DAEMON_LOGFILE_DIR=./
Q_SEQ_DIR_PATH=./
BLAST_OUT_DIR_PATH=./
BLASTALL_PATH=./blastall
genbank=/dsk/1/nucl_data/genbank
swissprot=/dsk/1/prot_data/swissprot
```

The next task in this sequence of tasks is *Starting the BLAST daemon*.

Related tasks:

- "Verifying that the correct version of the blastall executable and matrix files are installed" on page 91
- "Starting the BLAST daemon" on page 94
- "Registering nicknames for BLAST data sources" on page 98

Starting the BLAST daemon

This task is part of the main task for *Adding BLAST to a federated system*. Before you can access BLAST data sources, you must have the BLAST daemon running.

Prerequisites:

Before you start the BLAST daemon, you must have write access to all paths listed under the DAEMON_LOGFILE_DIR, BLAST_OUT_DIR_PATH, and Q_SEQ_DIR_PATH entries in the configuration file.

Procedure:

To start the BLAST daemon if you are in the daemon installation directory, did not change the name of the daemon configuration file, and the configuration file is in the same directory as the daemon executable file, type the following command at the command line:

```
db2blast_daemon
```

The executable starts a new process in which the BLAST daemon runs.

To start the BLAST daemon if you changed the name of the daemon configuration file or are not in the directory in which the daemon configuration file is located, you must use the -c option on the wrapper daemon command to point the daemon executable to the new name or location.

For example, the following command causes the wrapper daemon to look for its configuration information in a file called `BLAST_D.config` in the subdirectory `cfg`.

```
db2blast_daemon -c cfg/BLAST_D.config
```

The next task in this sequence of tasks is *Registering the BLAST wrapper*.

Related tasks:

- "Configuring the BLAST daemon" on page 91
- "Registering the BLAST wrapper" on page 95

Registering the BLAST wrapper

This task is part of the main task for *Adding BLAST to a federated system*. You must register the wrapper in order to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from data sources. Wrappers are installed on your system as library files.

Procedure:

To register the BLAST wrapper, submit the CREATE WRAPPER statement.

For example, to create a BLAST wrapper on AIX called `my_blast` from the default library file, `libdb2lsblast.a`, submit the following statement:

```
CREATE WRAPPER my_blast LIBRARY 'libdb2lsblast.a'
  OPTIONS(DB2_FENCED 'N');
```

For a table of default library filenames for the BLAST wrapper by supported platform, see ″After installing DB2 Life Sciences Data Connect″ in the Related tasks section below. For more information on the CREATE WRAPPER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Setting the DB2_DJ_COMM environment variable for the BLAST wrapper*.

Related tasks:

- "Registering the table-structured file wrapper" on page 16
- "Registering the Documentum wrapper" on page 36

- “Registering the Excel wrapper” on page 71
- “Starting the BLAST daemon” on page 94
- “Setting the DB2_DJ_COMM environment variable for the BLAST wrapper” on page 96
- “Registering the XML wrapper” on page 116

Setting the DB2_DJ_COMM environment variable for the BLAST wrapper

This task is part of the main task for *Adding BLAST to a federated system*. To improve performance when BLAST data sources are accessed, set the DB2_DJ_COMM environment variable. This variable determines whether the federated server loads the wrapper upon initialization.

Procedure:

To set the DB2_DJ_COMM environment variable, submit the `db2set` command with the wrapper library that corresponds to the wrapper that you specified in the associated CREATE WRAPPER statement.

For example:

```
db2set DB2_DJ_COMM='libdb2lsblast.a'
```

Ensure that there are no spaces on either side of the equal sign (=).

There is overhead associated with loading the wrapper libraries during database startup. To avoid this overhead, only specify libraries you intend to access.

For more information about the DB2_DJ_COMM environment variable, see the *DB2 Administration Guide.*

The next task in this sequence of tasks is *Registering the server for a BLAST data source.*

Related tasks:

- “Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper” on page 17
- “Setting the DB2_DJ_COMM environment variable for the Documentum wrapper” on page 37
- “Registering the BLAST wrapper” on page 95
- “Registering the server for a BLAST data source” on page 97
- “Setting the DB2_DJ_COMM environment variable for the XML wrapper” on page 116

Registering the server for a BLAST data source

This task is part of the main task for *Adding BLAST to a federated system*. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the BLAST server to the federated system, use the CREATE SERVER statement.

For each machine on which the BLAST executable and daemon are installed in your environment, you must register one server for each type of BLAST search you want to run using that instance of the BLAST executable and daemon.

For example, to register a server called blast_server1 for the my_blast wrapper created using the CREATE WRAPPER statement that will be used for BLASTn searches, submit the following statement:

```
CREATE SERVER blast_server1
 TYPE blastn
   VERSION 2.1.2
   WRAPPER my_blast
   OPTIONS (NODE 'big_rs.company.com', PORT '4007')
```

Arguments

TYPE Determines the type of BLAST search performed using the given server. This argument is required. It must be set to one of the following values: blastn, blastp, blastx, tblastn, tblastx.

VERSION
Specifies the version of the server that you are using. It should be set to the version of blastall that you are running. This argument is required.

WRAPPER
Specifies the name of the wrapper that you registered using the CREATE WRAPPER statement. This argument is required.

Options

NODE
Specifies the host name of the system on which the BLAST daemon process is running. This option is required.

PORT Specifies the port number on which the daemon will listen for BLAST job requests. The port number must be the same number specified in the daemon_port option of the daemon configuration file. The default is 4007. This option is optional.

For more information on the CREATE SERVER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Registering nicknames for BLAST data sources*.

Related tasks:

- "Registering the server for table-structured files" on page 18
- "Registering the server for Documentum data sources" on page 38
- "Registering the server for an Excel data source" on page 72
- "Setting the DB2_DJ_COMM environment variable for the BLAST wrapper" on page 96
- "Registering nicknames for BLAST data sources" on page 98
- "Registering the server for an XML data source" on page 117

Registering nicknames for BLAST data sources

This task is part of the main task for *Adding BLAST to a federated system*. After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to a BLAST data source in a query.

Procedure:

To register a BLAST nickname, use the CREATE NICKNAME statement. . Since each type of BLAST search is handled by a separate server, you must define a separate nickname for each type of BLAST search that you want to run against a given BLAST-able data source.

The nickname specifies column information for the definition line portion of the data source. All other columns are fixed. For more information on definition line parsing, see "Definition line parsing" on page 100. For more information on fixed columns, see "Fixed columns" on page 101.

The syntax for the CREATE NICKNAME statement for BLAST is:

```
                                        ┌─,──────────────────────────────────┐
                                        ▼                                    │
►►──CREATE NICKNAME──nickname──(────column-name──┤ column-information ├──┴──)──────────►

►──FOR SERVER──server-name──OPTIONS──(──DATASOURCE──'data_source_name' ──,──────────────►
►─TIMEOUT─'timeout_duration'─)──────────────────────────────────────────►◄
```

column-information:

```
├──┤ data-type ├──┤ column-option ├──┤ nickname-column-options ├──────────┤
```

data-type:

```
├──┬─┬─INTEGER─┬─────────────────────────┬─────────────────────────────────┤
   │ └─INT─────┘                         │
   ├─┬─FLOAT──┬───────────────────┬─┬────┤
   │ │        └─(──integer──)─────┘ │    │
   │ └─DOUBLE─┬─────────────┬───────┘    │
   │          └─PRECISION───┘            │
   └─VARCHAR──(──integer──)──────────────┘
```

column-option:

```
├──┬────────────┬──────────────────────────────────────────────────────────┤
   └─NOT NULL───┘
```

nickname-column-options:

```
├──OPTIONS──(──INDEX──'index_number'──,──DELIMITER──'delimiter'──────────►

►──┬──────────────────────────────────┬─)─────────────────────────────────┤
   └─DEFAULT──'new_default_value'─────┘
```

For more information on the CREATE NICKNAME statement, see the *DB2 SQL Reference*.

Nickname column options

Nickname column option values must be enclosed in single quotation marks.

INDEX
The ordinal number of the column on which this option appears in the group of definition line columns. This option is required. For more information on definition line parsing, see "Definition line parsing" on page 100.

DELIMITER
The delimiter characters that should be used to determine the end point of the definition line information for the column on which this option appears. If more than one character appears in this option's value, then the first occurrence of any one of the characters will signal the end of this field's information. The default is end of line. This

option is required, except for the last column specified if you want that column to contain the remainder of the definition line. For more information on definition line parsing, see “Definition line parsing”.

DEFAULT
Specifies a new default value for the following input fixed columns:
- E_value
- QueryStrands
- GapAlign
- NMisMatchPenalty
- NMatchReward
- Matrix
- FilterSequence
- NumberOfAlignments
- GapCost
- ExtendedGapCost
- WordSize
- ThresholdEx

This new value overrides the pre-set default values. The new default value must be of the same type as the value indicated for a given column. For more information on input fixed columns, see “Input fixed columns” on page 101. This option is optional.

Nickname options

Nickname option values must be enclosed in single quotation marks.

DATASOURCE
The name of the data source against which the BLAST search will be run. The exact string used here must be present in the configuration file of the BLAST daemon. For more information on the configuration file, see ″Configuring the BLAST daemon″ in the Related tasks section below. This option is required.

TIMEOUT
The maximum time, in minutes, that the BLAST wrapper will wait for results from the daemon. The default is 60. This option is optional.

Definition line parsing

The definition line, also called the defline, is like a key for each sequence in the BLAST-able data source and is returned as part of each BLAST hit.

If you are interested in including the definition line information in your results table, you must specify the definition line columns in the CREATE NICKNAME statement. Each column specification must specify an INDEX option. The DELIMITER option must be specified for each column, except for the last column specified if you want that column to contain the remainder of the definition line.

For more information on the INDEX and DELIMITER options, see "Nickname column options" on page 99.

The definition line fields must be of type integer, float, double, or varchar.

Note: If data are found in the Accession Number field of a BLAST hit, these data are inserted before data in the Definition field of that BLAST hit. The resulting definition line that contains the Accession Number data followed by the Definition field data is parsed by the wrapper.

For an example showing how to specify definition line columns in the CREATE NICKNAME statement, see "CREATE NICKNAME example" on page 104.

Fixed columns

The CREATE NICKNAME statement automatically creates fixed columns. The fixed columns do not appear in the CREATE NICKNAME statement, but are part of the nickname definition and can be referenced in SQL queries. There are two types of fixed columns, input and output.

Input fixed columns

Input fixed columns are used as parameter-passing predicates in SQL queries. They pass standard BLAST switches to BLAST. BLAST then runs against the specified data source using these switches. Input fixed columns can also be referenced in the query select list and returned as part of the results table. Input fixed columns are listed in Table 22.

Table 22. Input fixed columns

Name	Data type	Description
BlastSeq	varchar(32000)	Passes the query sequence to the BLAST wrapper.
E_Value	double	Both an input and an output parameter. As an input parameter, this column indicates to the BLAST wrapper the upper limit of expect values that should be returned from blastall.
QueryStrands	integer	Specifies which strands should be compared when performing a BLASTn search. A value of 1 indicates that the top strand should be used, 2 indicates the bottom strand, and 3 indicates that both strands should be compared.

Table 22. Input fixed columns (continued)

Name	Data type	Description
GapAlign	char(1)	Indicates to the wrapper whether gapped alignments are permitted in the BLAST output.
Matrix	varchar(50)	Determines which substitution matrix is used by blastall to determine the degree of similarity between pairings of amino acids. Only those BLAST search types that compare amino acids to amino acids use this predicate.
NMisMatchPenalty	integer	Specifies the value that blastall deducts from the score of an alignment if one of the pairs of nucleotides in the homologous region does not match. Only those BLAST search types that compare nucleotides to nucleotides use this predicate.
NMatchReward	integer	Specifies the value that blastall adds to the score of an alignment for each of the pairs of nucleotides in the homologous region that do match. Only those BLAST search types that compare nucleotides to nucleotides use this predicate.
FilterSequence	char(1)	Indicates to blastall whether to perform filtering to remove biologically uninteresting segments from the query sequence. If the search type is BLASTn, the filter used is DUST. Otherwise, filtering is performed by SEG.
NumberOfAlignments	integer	Specifies how many HSP alignments to include in the BLAST output.
GapCost	integer	Specifies the value that blastall deducts from the score of an alignment if a gap must be introduced in either the query sequence or the hit sequence to allow the length of the alignment to grow.
ExtendedGapCost	integer	Specifies the value that blastall deducts from the score of an alignment if a gap that was already introduced in either the query sequence or the hit sequence must be extended by one nucleotide or amino acid to allow the length of the alignment to grow.
WordSize	integer	Indicates to blastall the length of the initial hits that blastall initially searches in the database.

Table 22. Input fixed columns (continued)

Name	Data type	Description
ThresholdEx	integer	Indicates the score threshold below which BLAST does not attempt to extend a hit any further.

The supported BLAST search types and switches for each input fixed column are listed in Table 23

Table 23. BLAST search types and switches supported by the input fixed columns

Name	BLAST search types	BLAST switch	Req?	Default
BlastSeq	n, p, x, tn, tx	–l	Y	N/A
E_Value	n, p, x, tn, tx	–e	N	10
QueryStrands	n	S	N	3
GapAlign	n, p, x, tn, tx	–g	N	T
Matrix	p, x, tn, tx	–n	N	BLOSUM62
NMisMatchPenalty	n	–q	N	–3
NMatchReward	n	–r	N	1
FilterSequence	n, p, x, tn, tx	–F	N	T
NumberOfAlignments	n, p, x, tn, tx	–b	N	250
GapCost	n, p, x, tn, tx	–G	N	11
ExtendedGapCost	n, p, x, tn, tx	–E	N	1
WordSize (for Blastn, a value less than 7 is invalid)	n, p, x, tn, tx	–W	N	11 –BLASTn 3 –BLASTp
ThresholdEx	n, p, x, tn, tx	–f	N	0

Output fixed columns

Output fixed columns are returned in the query results table and can be used as predicates. Output fixed columns are listed in Table 24 on page 104.

Table 24. Output fixed columns

Name	Data type	Description
Score	double	The computed score for an HSP as reported in the BLAST results.
E_value	double	Both an input and an output parameter. As an output parameter, this column provides the computed score for an HSP as reported in the BLAST results.
Length	integer	The length of the hit sequence as reported in the BLAST results.
HSP_Info	varchar(100)	The information string for the given HSP, as reported by BLAST. This string contains information about the number of nucleotides or amino acids that matched between the query sequence and the hit sequence.
HSP_Q_Start	integer	The numeric position of the first homologous nucleotide or amino acid on the query sequence.
HSP_Q_End	integer	The numeric position of the last homologous nucleotide or amino acid on the query sequence.
HSP_Q_Seq	varchar(32000)	The segment of the query sequence beginning at HSP_Q_Start and ending at HSP_Q_End.
HSP_H_Start	integer	The numeric position of the first homologous nucleotide or amino acid on the hit sequence.
HSP_H_End	integer	The numeric position of the last homologous nucleotide or amino acid on the hit sequence.
HSP_H_Seq	varchar(32000)	The segment of the hit sequence beginning at HSP_H_Start and ending at HSP_H_End.
HSP_Midline	varchar(32000)	The string output by BLAST that indicates the degree of homology between the amino acids or nucleotides at each position in the homologous regions of the query and hit sequences.

CREATE NICKNAME example

The following CREATE NICKNAME statement defines the nickname genbank.

It assumes the definition field in a BLAST result contains the following information:

```
>276342 15:8924 PMON5426
```

where:

276342 The accession field of the BLAST result.

15:8924 PMON5426
The definition field in a BLAST result containing an organism number followed by an experiment number and then a unique identifier.

With this information, the following nickname is created:

```
CREATE NICKNAME genbank (
     acc_num integer   OPTIONS(INDEX '1', DELIMITER ' '),
     org_num integer   OPTIONS(INDEX '2', DELIMITER ':'),
     exp_num  integer   OPTIONS(INDEX '3', DELIMITER ' '),
     u_id  varchar(10)  OPTIONS(INDEX '4'))
     FOR SERVER blast_server1
       OPTIONS(DATASOURCE 'genbank', TIMEOUT '300');
```

The column `acc_num` would contain `276342`, the column `org_num` would contain 15, the column `exp_num` would contain `8924`, and the column `u_id` would contain `PMON5426`.

After you submit the CREATE NICKNAME statement, you can use the nickname genbank to query your federated system. You can also join the genbank nickname with other nicknames and tables in your federated system.

There are no further tasks in this sequence of tasks.

Related tasks:

- "Registering nicknames for table-structured files" on page 19
- "Registering nicknames for Documentum data sources" on page 40
- "Registering nicknames for Excel data sources" on page 73
- "Configuring the BLAST daemon" on page 91
- "Registering the server for a BLAST data source" on page 97
- "Registering nicknames for XML data sources" on page 118
- Chapter 8, "Specifying costing nickname options" on page 133

Constructing BLAST SQL queries

SQL for BLAST data sources must contain only special input predicates used to pass standard BLAST switches to the blastall executable file.

Restrictions:

To be valid, every query passed to the BLAST wrapper must contain at least the `BlastSeq` input predicate. All other predicates are optional.

Procedure:

To construct a BLAST query, use the input predicates in the WHERE clause of your SQL statement.

The following example shows three input predicates: `BlastSeq`, `GapCost`, and `NMisMatchPenalty`.

```
Select * from blast b where
BlastSeq = 'GTCCAGCC...' AND
GapCost = -10 AND
NMisMatchPenalty = -4;
```

For a list of data types, descriptions, BLAST switches, and search types supported for each input predicate, see ’Registering nicknames for BLAST data sources’ in the Related tasks section below.

Related tasks:

- “Sample BLAST queries” on page 106

Sample BLAST queries

Several sample BLAST queries are provided below to illustrate how queries are constructed for BLAST data sources.

Procedure:

To run queries, use the examples below as a guide.

In these queries, the name used for each nickname indicates the type of BLAST search and the data source. This is done so that the registration statements do not need to be listed with each sample query. Also, some of the queries make use of other hypothetical data sources so that these examples can illustrate the behavior of the wrapper when joined with other data sources.

Query 1

```
select *
from blastn_genbank
where BlastSeq =
 'caacccctccagccgagttgtcaatggcgaggaagctgttccccac';
```

When this SQL statement is executed, the wrapper will perform a BLASTn search of GenBank using the indicated sequence. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 2

```
select *
from blastn_genbank
where BlastSeq =
 'caacccctccagccgagttgtcaatggcgaggaagctgttccccac'
 and GapCost = 8 and NmisMatchPenalty = -4;
```

When this SQL statement is executed, the wrapper will perform a BLASTn search of GenBank using the indicated sequence. In addition, the wrapper will pass the two indicated parameters to the daemon, and they will be passed to the blastall command line. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 3

```
select blp.*
from blastp_swissprot blp, protein_db prdb
where prdb.keyword = 'malic enzyme'
 and blp.BlastSeq = prdb.sequence;
```

When this SQL statement is executed, the wrapper will perform zero or more BLASTp searches of SWISS-PROT, depending on the number of sequences returned from a hypothetical protein sequence database. This statement will be broken into two separate queries by DB2, and one BLASTp search will be run for each row that is returned from the hypothetical protein database. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 4

```
select Score, E_Value, HSP_Info, HSP_Q_Seq, HSP_H_Seq, HSP_Midline
from blastx_swissprot
where BlastSeq = 'gagttgtcaatggcgagg'
 and GapCost = 8;
```

When this SQL statement is executed, the wrapper will perform a BLASTx search of SWISS-PROT using the indicated sequence. In this case, blastall will translate the input sequence in all six reading frames and perform the homology search using each of the six newly created protein sequences. The HSPs in the results will contain amino acid-amino acid alignments, rather than nucleotide-nucleotide alignments. The supplied parameter will be passed to the daemon and then to blastall via the command line. The wrapper will return only those columns that are specifically requested in the query.

Query 5

```
select tblx.Score, tblx.E_Value, tblx.HSP_Info tblx.HSP_Q_Seq,
 HSP_H_Seq, HSP_Midline
from tblastx_genbank tblx, gen_exp_database gedb
where tblx.BlastSeq = gedb.sequence
 and gedb.organism = 'interesting organism'
 and GapCost = 8
 and FilterSequence = 'F';
```

When this SQL statement is executed, the wrapper will perform zero or more tBLASTx searches of GenBank, depending on the number of sequences returned from a hypothetical gene expression database. The statement will be broken into two separate queries by DB2, and one tBLASTx search will be run for each row that is returned from the hypothetical gene expression database. In this case, blastall will translate the input sequence and all of the sequences in GenBank in all six reading frames and perform the homology search using each of the six newly created query protein sequences and all of the newly created database protein sequences. The HSPs in the results will contain amino acid-amino acid alignments, rather than nucleotide-nucleotide alignments. The supplied parameters will be passed to the daemon and then to blastall via the command line. The wrapper will return only those columns that are specifically requested in the query.

Related tasks:

- “Running queries against Documentum data sources” on page 55
- “Running queries against Excel data sources” on page 74
- “Running queries against XML data sources” on page 125

Optimization tips for the BLAST wrapper

Running both the wrapper and the daemon on the same server can eliminate potential network communication bottlenecks.

Related reference:

- “Optimization tips and considerations for the table-structured file wrapper” on page 25

Messages for the BLAST wrapper

This section lists and describes messages that you might encounter when working with the wrapper for BLAST. For more information on messages, see the *DB2 Message Reference*.

Table 25. Messages issued by the wrapper for BLAST

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″sqlno_crule_save_plans [100]:rc (–2144272209) Empty plan list detect″.)	The SQL query submitted to DB2 could not be processed by the wrapper. Correct the syntax and resubmit.
SQL1816N	Wrapper ″BLAST_WRAPPER″ cannot be used to access the ″type″ of data source (″<server type>″ ″″) that you are trying to define to the federated database.	The CREATE SERVER statement used an invalid TYPE. The type must be one of the supported BLAST types.
SQL1817N	The CREATE SERVER statement does not identify the ″version″ of data source that you want defined to the federated database.	The CREATE SERVER statement did not specify the version.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″Unable to connect to daemon″.	The blast wrapper was not able to connect to the daemon. The daemon might not be running. It might be misconfigured. The machine that it is running on might be unreachable.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″Blast daemon timeout expired″.	No results were received from the daemon before the timeout as specified on the CREATE NICKNAME statement elapsed. Increase the timeout or check to see if there is a problem with the daemon.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″Blast Daemon Failed″.	The daemon stopped communicating or the results returned were not properly formatted.

Table 25. Messages issued by the wrapper for BLAST (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″Unknown error from the blast daemon″.	The blast wrapper received an error code from the daemon that it doesn't recognize. The daemon version might not be compatible with the wrapper version.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″Column rename not allowed″.	An ALTER NICKNAME statement was issued trying to rename one of the columns. Renaming a column is not allowed.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Blast Wrapper″. Associated text and tokens are ″XML parser error″.	The Xerces parser is in an invalid state or has thrown an exception.
SQL1823N	No data type mapping exists for data type ″<data type name>″ from server ″<server name>″.	The data type specified is not supported by this column.
SQL1881N	″DEFAULT″ is not a valid ″COLUMN″ option for ″<column-name>″	The DEFAULT option was used on a column that does not support it. Output only columns and definition line columns do not have default values.
SQL1882N	The ″COLUMN″ option ″DEFAULT″ cannot be set to ″<option-value>″ for ″<column-name>″.	The value specified for the DEFAULT option is of an incompatible type for the column or is incorrectly formatted.

Related reference:

- "Messages for the table-structured file wrapper" on page 25
- "Messages for the Documentum wrapper" on page 62
- "Messages for the Excel wrapper" on page 78
- "Messages for the XML wrapper" on page 127

Chapter 7. XML as a data source

This chapter explains what XML is, how to add XML data sources to your federated system, and lists the error messages associated with the XML wrapper.

What is XML?

The Extensible Markup Language (XML) is a universal format for structured documents and data. XML files have a file extension of xml. Like HTML, XML makes use of tags (words bracketed by '<' and '>') for structuring data in the document. A sample XML document is shown in Figure 9.

```
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc)
```

Figure 9. Sample XML document

The XML wrapper enables use of SQL to query external XML documents stored in files. Figure 10 on page 112 shows how the XML works with your federated system.

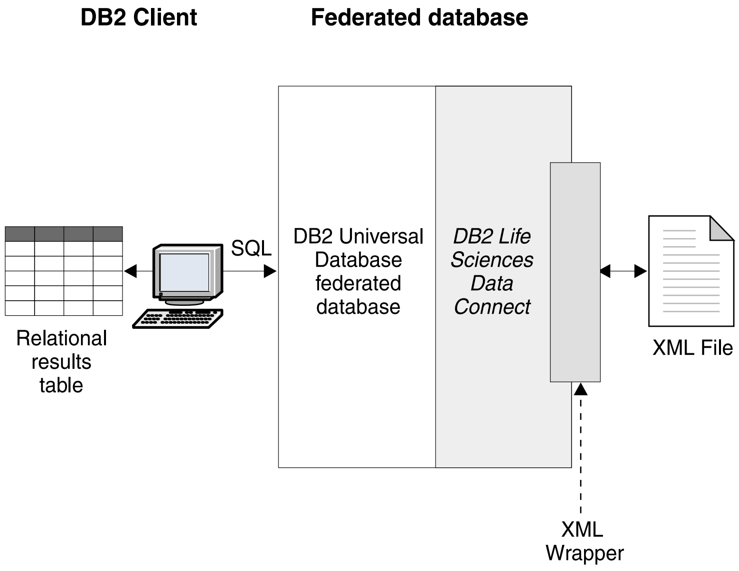


Figure 10. How the XML wrapper works

The XML Wrapper allows mapping of XML data from an external data source into a relational schema composed of a set of nicknames. The structure of an XML document is logically equivalent to a relational schema where the nested and repeating elements are modeled as separate tables with foreign keys.

The nicknames corresponding to an XML document are organized into a tree where the child nicknames model elements that are nested in the element corresponding to the parent nickname.

There are basically two cases when a nested element should be modeled as a separate nickname:

- Repeating elements
- Elements with distinct identity and rich structure

Child and parent nicknames are connected by primary/foreign keys generated by the wrapper.

XPath expressions are used to map an XML document into a relational schema composed of a set of nicknames. XPath is an addressing mechanism for identifying the parts of an XML file – for example, the groups of nodes and attributes within an XML document tree. The basic XPath syntax is similar to file system addressing.

Each nickname is defined by an XPath expression that identifies the XML elements representing individual tuples, and a set of XPath expressions specifying how to extract the column values from each element.

An Example:

The following example illustrates how the sample XML document, shown in Figure 9 on page 111, is mapped into a set of nicknames, how parent and child relationships are modeled using primary and foreign keys, how XPath expressions are used to define individual tuples and columns within each element of the document, and how a query can be run against the XML document once it is registered to your federated system.

The sample XML document contains a set of `customer` elements each enclosing several `order` and `payment` elements.

The `order` elements enclose several `item` elements.

The relationship between the elements is shown in Figure 11.

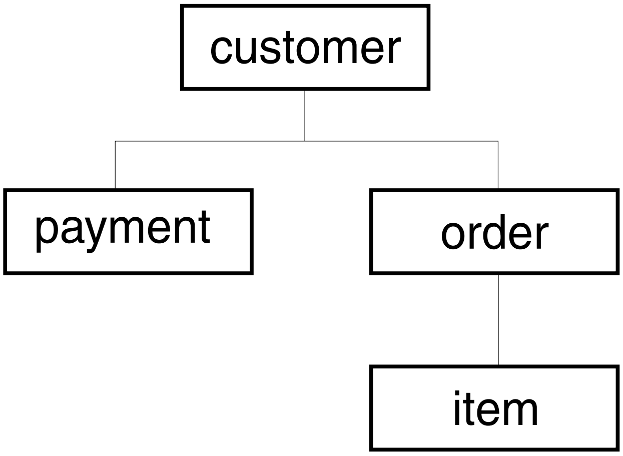


Figure 11. Tree structure of the sample XML document

From this structure, the XML document can be mapped, using the CREATE NICKNAME statement, into a relational schema using four nicknames:

- customers
- orders
- payments
- items

The relationships between the nicknames are defined by specifying each nickname as a parent or child nickname using special primary and foreign key nickname column options, respectively. Each parent nickname must have a special column designated with the a primary key column option. Children of a parent are defined by having a special column that references the primary key column of a parent nickname using a foreign key column option. The designated primary and foreign nickname columns do not correspond to data in your XML document as these nickname columns will contain keys generated by the wrapper. A nickname can have multiple children while it must have exactly one parent, except the root that has no parent.

For the sample XML document the `customers` nickname would have a primary key defined, and the `orders`, `payments`, and `items` nicknames would each have a foreign key defined that points to their parent nickname. The `orders` and `payments` nicknames would have foreign keys pointing to `customers`, and the `items` nickname would have a foreign key pointing to `orders`.

To identify the XML elements representing individual tuples, one XPath expression is created. In this example, all the customer elements can be referenced using the `//customer` XPath expression and all the order elements can be referenced using the `.//order` XPath expression.

A set of XPath expressions are created to specify how to extract the column values from each element. In this example, the `id` attribute of the customer elements, now a column defined in the nickname, can be referenced using the `./@id` XPath expression. The name element of the customer elements can be referenced by using the `.//name` XPath expression, and the address element of the customer elements can be referenced by using the `.//address/@street` XPath expression.

Once the XML document is mapped into a set of nicknames using the CREATE NICKNAME statement, each nickname defined as a parent or child using primary and foreign keys, with XPath expressions defining individual tuples and columns within each element of the document, you can run SQL queries against the XML document.

For more detailed information on how to create nicknames and for the syntax of the CREATE NICKNAME statement, see ’Registering nicknames for XML data sources’ in the Related tasks section below.

Related concepts:

- “What are table-structured files?” on page 13
- “What is Documentum?” on page 31
- “What is Excel?” on page 69

- “What is BLAST?” on page 85

Related tasks:

- “Adding XML to a federated system” on page 115
- “Registering nicknames for XML data sources” on page 118

Adding XML to a federated system

Procedure:

To add an XML data source to a federated server:

1. Register the wrapper using the CREATE WRAPPER statement.
2. Optional: Set the DB2_DJ_COMM environment variable to improve query performance.
3. Register the server using the CREATE SERVER statement.
4. Register nicknames using the CREATE NICKNAME statement.
5. Create views for non-root nicknames

The statements can be run from the DB2 command line processor. After the XML wrapper is added to your federated system, you can run queries against an XML data source.

Related tasks:

- “Registering the XML wrapper” on page 116
- “Setting the DB2_DJ_COMM environment variable for the XML wrapper” on page 116
- “Registering the server for an XML data source” on page 117
- “Registering nicknames for XML data sources” on page 118
- “Creating federated views for non-root nicknames (XML wrapper)” on page 123
- “Adding table-structured files to a federated system” on page 16
- “Adding Documentum to a federated system” on page 33
- “Adding Excel to a federated system” on page 71
- “Adding BLAST to a federated system” on page 90

Registering the XML wrapper

This task is part of the main task for *Adding XML to a federated system*. You must register the wrapper in order to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from data sources. Wrappers are installed on your system as library files.

Procedure:

To register the XML wrapper, submit the CREATE WRAPPER statement.

For example, to create an XML wrapper on AIX called `my_xml`from the default library file, `libdb2lsxml.a`, submit the following statement:

```
CREATE WRAPPER my_xml LIBRARY 'libdb2lsxml.a'
  OPTIONS(DB2_FENCED 'N');
```

For a table of default library filenames for the XML wrapper by supported platform, see 'After installing DB2 Life Sciences Data Connect' in the Related tasks section below. For more information on the CREATE WRAPPER statement, see the *DB2 SQL Reference*.

The next task in this sequence of tasks is *Setting the DB2_DJ_COMM environment variable for the XML wrapper*.

Related tasks:

- "Registering the table-structured file wrapper" on page 16
- "Registering the Documentum wrapper" on page 36
- "Registering the Excel wrapper" on page 71
- "Registering the BLAST wrapper" on page 95
- "Setting the DB2_DJ_COMM environment variable for the XML wrapper" on page 116

Setting the DB2_DJ_COMM environment variable for the XML wrapper

This task is part of the main task for *Adding XML to a federated system*. To improve performance when XML documents are accessed, set the DB2_DJ_COMM environment variable. This variable determines whether the federated server loads the wrapper upon initialization.

Procedure:

To set the DB2_DJ_COMM environment variable, submit the `db2set` command with the wrapper library that corresponds to the wrapper that you specified in the associated CREATE WRAPPER statement.

```
db2set DB2_DJ_COMM='libdb2lsxml.a'
```

Ensure that there are no spaces on either side of the equal sign (=).

There is overhead associated with loading the wrapper libraries during database startup. To avoid this overhead, only specify libraries you intend to access.

For more information about the DB2_DJ_COMM environment variable, see the *DB2 Administration Guide*.

The next task in this sequence of tasks is *Registering the server for an XML data source*.

Related tasks:

- "Setting the DB2_DJ_COMM environment variable for the table-structured file wrapper" on page 17
- "Setting the DB2_DJ_COMM environment variable for the Documentum wrapper" on page 37
- "Setting the DB2_DJ_COMM environment variable for the BLAST wrapper" on page 96
- "Registering the XML wrapper" on page 116
- "Registering the server for an XML data source" on page 117

Registering the server for an XML data source

This task is part of the main task for *Adding XML to a federated system*. After the wrapper is registered using, you must register a corresponding server.

Procedure:

To register the XML server to the federated system, use a CREATE SERVER statement:

```
CREATE SERVER xml_server WRAPPER my_xml
```

where:

WRAPPER
Specifies the name of the wrapper that you registered with the associated CREATE WRAPPER statement. This argument is required.

Note: The XML wrapper does not use the TYPE and VERSION keywords. An error is issued if these keywords are used in the CREATE SERVER statement.

The next task in this sequence of tasks is *Registering nicknames for XML data sources*.

Related tasks:

- "Registering the server for table-structured files" on page 18
- "Registering the server for Documentum data sources" on page 38
- "Registering the server for an Excel data source" on page 72
- "Registering the server for a BLAST data source" on page 97
- "Setting the DB2_DJ_COMM environment variable for the XML wrapper" on page 116
- "Registering nicknames for XML data sources" on page 118

Registering nicknames for XML data sources

This task is part of the main task for *Adding XML to a federated system*. You must create nicknames that model the tree structure of your XML data source. A parent nickname must be created to model the parent or root of the tree. Child nicknames must be created to model elements that are nested in the element corresponding to the parent nickname.

Parent and child nicknames are connected by primary and foreign keys specified on the CREATE NICKNAME statement.

Each nickname is defined by XPath expressions that:

- identify the XML elements representing individual tuples
- specify how to extract the column values from each element.

Nicknames are associated with your XML documents in one of two ways:

- in a fixed manner (using the FILE_PATH nickname option). When this option is used, the nickname represents data from a specific XML document.
- with a filename specified at query time (using the DOCUMENT nickname column option). When this option is used, the nickname can be used to represent data from any XML document whose schema matches the nickname definition.

More information on the these options is provided in the procedure section below.

Procedure:

To map the XML data source to relational tables, you must create nicknames using the CREATE NICKNAME statement.

```
►►──CREATE NICKNAME──nickname──(──column-name──┤ column-information ├──)──────────────►

►─FOR SERVER──server-name──OPTIONS──(──┬──────────────────────┬─────────────────────►
                                       └─FILE_PATH──'path' ─,─┘

►─XPATH──'xpath_expression' ─)──────────────────────────────────────────────────────►◄
```

column-information:

```
├──┤ data-type ├──┤ column-option ├──┤ nickname-column-options ├──────────────────────┤
```

data-type:

```
├──┬─SMALLINT──────────────────────────────────────────────┬──────────────────────────┤
   ├─┬─INTEGER─┬───────────────────────────────────────────┤
   │ └─INT─────┘                                           │
   ├─REAL──────────────────────────────────────────────────┤
   ├─DOUBLE──┬───────────┬─────────────────────────────────┤
   │         └─PRECISION─┘                                 │
   ├─┬─DECIMAL─┬──┬─────────────────────────────────────┬──┤
   │ ├─DEC─────┤  └─(──integer──┬─────────────┬──)──────┘  │
   │ ├─NUMERIC─┤                └─,──integer──┘            │
   │ └─NUM─────┘                                           │
   ├─┬─CHARACTER─┬──┬─────────────────┬────────────────────┤
   │ └─CHAR──────┘  └─(──integer──)───┘                    │
   └─VARCHAR──(──integer──)────────────────────────────────┘
```

nickname-column-options:

```
├──OPTIONS──(──┬─DOCUMENT──'value' ───────────────┬──)────────────────────────────────┤
               ├─XPATH──'xpath_expression' ───────┤
               ├─PRIMARY_KEY──'YES' ──────────────┤
               └─FOREIGN_KEY──'parent_nickname' ──┘
```

column-option:

```
├──┬──────────┬───────────────────────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

Nickname options

FILE_PATH
Specifies the file path of the XML document. If this nickname option is specified then no DOCUMENT nickname column option should be specified. This option is accepted only for the root nickname (the nickname identifying the elements at the top level of the XML document).

XPATH

Specifies an XPath expression that identifies the XML elements representing individual tuples. The XPATH nickname option for a child nickname is evaluated in the context of the path specified by the XPATH nickname option of its parent. This XPath expression is used as a context for evaluating column values identified by the XPATH nickname column options.

Nickname column options

DOCUMENT

Specifies the kind of XML data. Currently, the XML wrapper only supports FILE. This option is accepted only for the root nickname (the nickname identifying the elements at the top level of the XML document). Only one column can be specified with the DOCUMENT option per nickname. The column associated with the DOCUMENT option has to be of data type VARCHAR.

Using the DOCUMENT nickname column option, instead of the FILE_PATH nickname option, implies that the document corresponding to this nickname will be supplied during query execution. If the DOCUMENT option has the ″FILE″ value, it means that what will be supplied during query execution is the name of a file containing a document. The following CREATE NICKNAME example illustrates the use of the DOCUMENT nickname column option:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'FILE'),
   name     VARCHAR(16)    OPTIONS(XPATH './/name'),
   address  VARCHAR(30)    OPTIONS(XPATH './/address/@street'),
   cid      VARCHAR(16)    OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '//customer');
```

The following query, specifying the location of the XML document in the WHERE clause, can now be run against the customers nickname:

```
SELECT * FROM customers WHERE doc='/home/db2user/Customers.xml';
```

XPATH

Specifies the XPath expression in the XML document where the data corresponding to this column can be found. This XPath expression is applied after evaluating the XPath expression specified in the XPATH nickname option.

PRIMARY_KEY

Indicates that this is a parent nickname. The column data-type should always be VARCHAR(16). A nickname can have at most one PRIMARY_KEY column option. ’YES’ is the only legal value. The column designated with this option holds a key generated by the wrapper. The column's value cannot be retrieved in a SELECT list and the XPATH option must not be specified for this column. The column can only be used to join parent and child nicknames.

FOREIGN_KEY
Indicates that this is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for this option is case sensitive. The column designated with this option holds a key generated by the wrapper. The column's value cannot be retrieved in a SELECT list and the XPATH option must not be specified for this column. The column can only be used to join parent and child nicknames.

Nickname examples

The following examples illustrate the procedure for creating nicknames for XML data sources using the sample XML file shown in for the sample document in Figure 12.

```
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc)
```

Figure 12. Sample XML file

To create the parent nickname, customers, specify the following statement:

```
CREATE NICKNAME customers
(
   id          VARCHAR(5)   OPTIONS(XPATH './@id')
   name        VARCHAR(16)  OPTIONS(XPATH './/name'),
   address     VARCHAR(30)  OPTIONS(XPATH './/address/@street'),
   cid         VARCHAR(16)  OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS( XPATH '//customer',
            FILE_PATH '/home/db2user/Customers.xml');
```

To create the nicknames for the children of customer –orders, payments, and items –specify the following three nickname statements:

For orders:

```
CREATE NICKNAME orders
(
   amount   INTEGER       OPTIONS(XPATH './amount'),
   date     VARCHAR(10)   OPTIONS(XPATH './date'),
   oid      VARCHAR(16)   OPTIONS(PRIMARY_KEY 'YES'),
   cid      VARCHAR(16)   OPTIONS(FOREIGN_KEY 'CUSTOMERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './/order');
```

For payments:

```
CREATE NICKNAME payments
(
   number   INTEGER       OPTIONS(XPATH './number'),
   date     VARCHAR(10)   OPTIONS(XPATH './date'),
   cid      VARCHAR(16)   OPTIONS(FOREIGN_KEY 'CUSTOMERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './/payment');
```

For items:

```
CREATE NICKNAME items
(
   name        VARCHAR(20)   OPTIONS(XPATH './name'),
   quantity    INTEGER       OPTIONS(XPATH './@quant'),
   oid         VARCHAR(16)   OPTIONS(FOREIGN_KEY 'ORDERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './/item');
```

The next task in this sequence of tasks is *Creating federated views for non-root nicknames (XML wrapper).*

Related tasks:

- “Registering nicknames for table-structured files” on page 19
- “Registering nicknames for Documentum data sources” on page 40
- “Registering nicknames for Excel data sources” on page 73

- “Registering nicknames for BLAST data sources” on page 98
- “Registering the server for an XML data source” on page 117
- Chapter 8, “Specifying costing nickname options” on page 133

Creating federated views for non-root nicknames (XML wrapper)

This task is part of the main task for *Adding XML to a federated system*. It is recommended that you define federated views over the hierarchy of nicknames that describe an XML document. Defining federated views ensures that queries that join pieces of an XML nickname hierarchy not including the root and queries that join on columns other than the special PRIMARY_KEY and FOREIGN_KEY columns execute properly.

Procedure:

To define federated views that include all required predicates and a full path to the root, perform the following steps:

1. Define a view for each non-root nickname as a join of all the nicknames on the path to the root.
2. In the WHERE clause, make the join predicates over the PRIMARY_KEY and FOREIGN_KEY columns.
3. In the SELECT list, include all the columns of the non-root nickname except the one designated with the FOREIGN_KEY nickname column option.
4. In the SELECT list, include the column of the parent nickname designated with the PRIMARY_KEY option.

View examples:

The following example illustrates the use of views. In this example, assume that the nicknames of the sample file shown in Figure 13 on page 124 were previously created as `customers`, `orders`, `payments` and `items`.

```
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc)
```

Figure 13. Sample XML file.

The views for the non-root nicknames `order`, `payment` and `item` are then:

For `order`:

```
CREATE FEDERATED VIEW order_view AS
SELECT o.amount, o.date, o.oid, c.cid
FROM customers c, orders o
WHERE c.cid = o.cid;
```

For `payment`:

```
CREATE FEDERATED VIEW payment_view AS
SELECT p.amount, p.date, c.cid
FROM customers c, payments p
WHERE c.cid = p.cid;
```

For `item`:

```
CREATE FEDERATED VIEW item_view AS
SELECT it.quantity, it.name, o.oid
FROM customers c, orders o, items i
WHERE c.cid = o.cid AND o.oid = i.oid;
```

Queries submitted to these views are processed correctly because the join path to the root is present.

For example, the following query pairs the amounts of customer's orders and payments from the same date:

```
SELECT o.amount, p.amount
FROM order_view o, payment_view p
WHERE p.date = o.date AND
   p.cid = o.cid;
```

There are no further tasks in this sequence of tasks.

Related tasks:

- "Registering nicknames for XML data sources" on page 118

Running queries against XML data sources

This section lists several sample queries using the nicknames `customers`, `orders`, and `items` created with the CREATE NICKNAME statement.

Procedure:

To run queries, use the following examples as a guide.

The following query displays all customer names:

```
SELECT name FROM customers;
```

The following query displays all records where the customer name is 'Smith':

```
SELECT * FROM customers where name='Smith';
```

The following query displays the amounts and customer names for each order of each customer:

```
SELECT c.name, o.amount FROM customers c, orders o where c.cid=o.cid;
```

The join `c.cid=o.cid` is required to indicate the child / parent relationship between the orders and the customers nicknames.

The following query selects the order amounts, the item names and the customer addresses for each order and item of each customer:

```
SELECT c.address, o.amount, i.name FROM customers c, orders o, items i
WHERE c.cid=o.cid AND o.oid=i.oid;
```

Again the two joins are required to maintain the child / parent relationships.

The following two examples show how to write queries using a nickname that does not specify a FILE_PATH nickname option, but does specify a

DOCUMENT nickname column option. The corresponding CREATE NICKNAME statement used to create the nickname customers, is shown below:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'FILE'),
   name     VARCHAR(16)    OPTIONS(XPATH './/name'),
   address  VARCHAR(30)    OPTIONS(XPATH './/address/@street'),
   cid      VARCHAR(16)    OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '//customer');
```

The following query selects all the data from the XML file Customers.xml with a file path of /home/db2user/Customers.xml:

```
SELECT * FROM customers WHERE doc='/home/db2user/Customers.xml';
```

The following query selects names of customers and dates of their orders for orders over the amount of 1000 from the XML file Customers.xml located at /home/db2user/Customers.xml:

```
SELECT c.name, o.date FROM customers c, orders o
WHERE c.doc='/home/db2user/Customers.xml' AND o.amount > 1000;
```

Related tasks:

- "Running queries against Documentum data sources" on page 55
- "Running queries against Excel data sources" on page 74

Limitations and considerations for the XML wrapper

This section contains a list of limitations and considerations associated with the use of the XML wrapper.

- The passthru capability is not supported.
- XML documents can only be read.

Related reference:

- "Wrapper limitations and considerations for the table-structured file wrapper" on page 23
- "File limitations and considerations for the table-structured file wrapper" on page 24
- "Limitations and considerations for the Documentum wrapper" on page 60
- "Wrapper limitations for the Excel wrapper" on page 77
- "Excel file limitations" on page 78

Messages for the XML wrapper

This section lists and describes messages that you might encounter when working with the wrapper for XML. For more information on messages, see the *DB2 Message Reference*.

Table 26. Messages issued by the wrapper for XML

Error Code	Message	Explanation
SQL0405N	The numeric literal ″<column_name>″ is not valid because its value is out of range.	The specified numeric literal is not in the acceptable range. Check data type of column in the CREATE NICKNAME statement.
SQL0408N	A value is not compatible with the data type of its assignment target. Target name is ″<column_name>″.	The data type of the value to be assign to the column is incompatible with the declared data type of the assignment target. Check data type of column in the CREATE NICKNAME statement.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Error creating wrapper object″.)	An error occurred when creating a new wrapper object. Contact DB2 support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Xerces initialization error″.)	An exception occurred during the initialization of the Xerces parser. Contact DB2 support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″<xalan_error_message>″.)	An error occurred when calling a Xalan function. Check the XML document. If the document is well structured, refer to Xalan documentation for more details on the error message.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″XalanDOMException: exception code is <exception_code>″.)	A XalanDOMException exception occurred. Refer to the Xalan documentation to see the meaning of the exception code.

Table 26. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Error getting node value″.)	Xalan tried to access an invalid node. Contact DB2 support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Error parsing XML document″.)	An error occurred when parsing the XML document. Check the XML document.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Error getting root element of XML document″.)	After parsing the XML document, Xalan tried to retrieve the root element but failed. Check the XML document.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unspecified exception while evaluating XPath expression″.)	An unspecified exception was generated by Xalan when evaluating an XPath expression. Check XML document and refer to Xalan documentation.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unspecified exception while getting node value″.)	An unspecified exception was generated by Xalan when retrieving a node value. Check XML document and refer to Xalan documentation.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Unspecified exception while building DOM tree″.)	An unspecified exception was generated by Xalan when building the DOM tree for the XML document. Check XML document and refer to Xalan documentation.

Table 26. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Memory allocation error″.)	An error occurred when allocating memory.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Column data type not supported″.	A nickname column has an unsupported data type. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″TYPE clause not supported″.	The CREATE SERVER statement contains a TYPE clause. This clause is not supported by the XML wrapper. Remove it.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″VERSION clause not supported″.	The CREATE SERVER statement contains a VERSION clause. This clause is not supported by the XML wrapper. Remove it.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid use of predicate with DOCUMENT column″.	The query contains a predicate with incorrect operands. Check the predicates in the query.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid use of predicate with FOREIGN_KEY column″.	The query contains a predicate with incorrect operands. Check the predicates in the query.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid use of predicate with PRIMARY_KEY column″.	The query contains a predicate with incorrect operands. Check the predicates in the query.

Table 26. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″XPATH and DOCUMENT options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″XPATH and FOREIGN_KEY options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″XPATH and PRIMARY_KEY options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″DOCUMENT and FOREIGN_KEY options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″DOCUMENT and PRIMARY_KEY options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″FOREIGN_KEY and PRIMARY_KEY options not compatible″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.

Table 26. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Column option missing″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″DOCUMENT column option not unique″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″FOREIGN_KEY column option not unique″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″PRIMARY_KEY column option not unique″.	The CREATE NICKNAME statement is not correct as specified. Check its syntax.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid DOCUMENT option value″.	The value of the DOCUMENT option specified in the CREATE NICKNAME statement is not valid: it can only be ’FILE’. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid PRIMARY_KEY option value″.	The value of the PRIMARY_KEY option specified in the CREATE NICKNAME statement is not valid.: it can only be ’YES’. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″Invalid FOREIGN_KEY option value″.	The value of the FOREIGN_KEY option specified in the CREATE NICKNAME statement is not valid: no parent nickname matching the option value could be found. Check the CREATE NICKNAME statement.

Table 26. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″XML wrapper″. Associated text and tokens are ″FILE_PATH and DOCUMENT options not compatible″.	The CREATE NICKNAME statement is not correct as specified: the FILE_PATH and DOCUMENT options cannot be specified at the same time. Check the CREATE NICKNAME syntax.
SQL1881N	″<option_name>″ is not a valid ″<option_type>″ option for ″<object_name>″.	The specified option might not exist or might not be valid for this particular data source. Check the CREATE NICKNAME statement.
SQL1883N	″<option_name>″ is a required ″<option_type>″ option for ″<object_name>″.	A required DB2 option has not been specified. Check the CREATE NICKNAME statement.

Related reference:

- “Messages for the table-structured file wrapper” on page 25
- “Messages for the Documentum wrapper” on page 62
- “Messages for the Excel wrapper” on page 78
- “Messages for the BLAST wrapper” on page 109

Chapter 8. Specifying costing nickname options

In order to produce efficient execution plans the optimizer generates a set of different plans and estimates the resources needed for each. The plan that requires least resources is used for the evaluation.

The estimates of the evaluation times for the query portions executed by external data sources are provided by the wrapper. The formulas used in this calculation use three basic parameters that can be changed to fit a particular installation.

These parameters are specified as the following nickname options:

RESET_COST
Specifies the time in milliseconds to contact the external server and get back the result.

ADVANCE_COST
Specifies the time in milliseconds to obtain each row.

BIND_COST
Specifies the time in milliseconds needed to pass a parameter from the wrapper to the external source.

All values must be given as integers. For default values, see the *DB2 Life Sciences Data Connect Release Notes Version 8*.

Procedure:

To specify a costing nickname option:

1. Analyze your installation to determine if customization of the costing options benefits your company's processing of federated queries.
2. If so, add one or more of the costing options to the wrapper's CREATE NICKNAME statement as a nickname option.
3. Submit the CREATE NICKNAME statement.

See the *DB2 SQL Reference* for more information about the CREATE NICKNAME statement.

Related tasks:

- "Registering nicknames for table-structured files" on page 19
- "Registering nicknames for Documentum data sources" on page 40
- "Registering nicknames for Excel data sources" on page 73

- “Registering nicknames for BLAST data sources” on page 98
- “Altering nicknames” on page 135
- “Registering nicknames for XML data sources” on page 118

Chapter 9. Altering nicknames

This chapter explains how to use the ALTER NICKNAME statement to alter previously registered nicknames.

Altering nicknames

You can use the ALTER NICKNAME statement to modify the federated database's representation of a data source or view.

Restrictions:

The ALTER NICKNAME statement can not be used to alter column names for any DB2 Life Sciences Data Connect wrapper.

Procedure:

To alter nickname column values, you must use the ALTER NICKNAME statement to:

- Change the local data types of these columns
- Add, change, or delete options for these columns

For more information on the ALTER NICKNAME statement, see the *DB2 SQL Reference*.

Related tasks:

- "Changing the data type" on page 135
- "Changing the nickname option" on page 136

Changing the data type

You can use the ALTER NICKNAME statement to change the data type of a column.

Procedure:

To change the data type of a column, use the ALTER NICKNAME statement.

For example, the following ALTER NICKNAME statement changes the local data type of the DRUG column to CHAR(30). The DRUG column was originally defined as a CHAR(20) using a CREATE NICKNAME statement. The nickname DRUGDATA1 refers to a local table-structured file called drugdata1.txt.

```
ALTER NICKNAME DRUGDATA1
   ALTER COLUMN DRUG
   LOCAL TYPE CHAR(30)
```

Related tasks:
- "Altering nicknames" on page 135
- "Changing the nickname option" on page 136

Changing the nickname option

You can use the ALTER NICKNAME statement to change a nickname option.

Procedure:

To change a nickname option, use the ALTER NICKNAME statement.

For example, the following ALTER NICKNAME statement changes the fully qualified path for the table-structured file, drugdata1.txt. The path was originally defined as '/user/pat/drugdata1.txt' using a CREATE NICKNAME statement. The nickname DRUGDATA1 refers to a local table-structured file called drugdata1.txt.

```
ALTER NICKNAME DRUGDATA1
   OPTIONS (SET FILE_PATH '/usr/kelly/data/drugdata1.txt')
```

Related tasks:
- "Altering nicknames" on page 135
- "Changing the data type" on page 135

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Bibliography

This bibliography contains DB2 Universal Database publications that you might find useful while working with DB2 Life Sciences Data Connect.

- *DB2 Connect User's Guide* (SC09-2954)
- *DB2 for UNIX Quick Beginnings* (GC09-2970)
- *DB2 SQL Reference* (SC09-2974)
- *DB2 Administration Guide: Planning* (SC09-2946)
- *DB2 Administration Guide: Implementation* (SC09-2944)
- *DB2 Administration Guide: Performance* (SC09-2945)
- *DB2 Message Reference* (GC09-2978)
- *IBM DB2 Universal Database Federated Systems Guide* (GC27-1224)
- *DB2 Life Sciences Data Connect Release Notes Version 8*

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Part Number: CT16FNA

Printed in U.S.A.

GC27-1235-00

(1P) P/N: CT16FNA

Spine information:

IBM® DB2® Life Sciences Data Connect

DB2 LSDC Planning, Installation, and Configuration Guide

Version 8