IBM i
Version 7.2

Database
Distributed database programming

Note

Before using this information and the product it supports, read the information in “Notices” on page 363.

This edition applies to IBM i 7.2 (product number 5770-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Distributed database programming

Distributed database programming describes the distributed relational database management portion of the IBM® i licensed program. Distributed relational database management provides applications with access to data that is external to the applications and typically located across a network of computers.

This topic also contains IBM i distributed data management (DDM) concepts, information about preparing for DDM communications, and DDM-related programming information. You can use IBM i DDM to prepare a system to access data in remote files and to control access to local files by remote systems.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

What's new for IBM i 7.2

Read about new or significantly changed information for the distributed database programming topic collection.

In IBM i 7.2, Distributed Relational Database Architecture™ (DRDA) support has been extended to include support for the following functions:

- QDDMDRDASERVER server authentication entry special value
- Simplified DDM and DRDA authentication entry management using group profiles
- DRDA explicit server name priority when using group profile server authentication entries
- QIBM_DB_DDMDRDA function usage ID alternative to DRDA/DDM user exit programs for restricting access
- Conjoined Mutual Authentication
- RDB alias can be specified explicitly in three-part SQL statements

What's new as of April 2019

- SQL Package creation will be allowed if profile of *PGM or *SRVPGM is not on the application server. The package will be owned by the connecting user profile.

How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The » image to mark where new or changed information begins.
- The « image to mark where new or changed information ends.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

To find other information about what's new or changed this release, see the Memo to users.

PDF file for Distributed database programming

You can view and print a PDF file of this information.

To view or download the PDF version of this document, select Distributed database programming.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.

2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (http://get.adobe.com/reader/).

DRDA and DDM overview

This topic describes the concepts and processing of IBM i distributed relational database support and IBM i distributed data management support.

DRDA overview

IBM i distributed relational database support consists of an implementation of IBM Distributed Relational Database Architecture (DRDA) and integration of other SQL clients through Application Requester Driver (ARD) programs.

This topic describes distributed relational database and how it is used on the IBM i operating system.

In addition, an example distributed relational database called Spiffy Corporation is described. This fictional company uses the IBM i product in a distributed relational database application program.

Distributed relational database processing

A *relational database* is a set of data stored in one or more tables in a computer.

A *table* is a two-dimensional arrangement of data consisting of horizontal rows and vertical columns as shown in the following table. Each *row* contains a sequence of values, one for each column of the table. A *column* has a name and contains a particular data type (for example, character, decimal, or integer).

Table 1. A typical relational table

Item	Name	Supplier	Quantity
78476	Baseball	ACME	650
78477	Football	Imperial	228
78478	Basketball	ACME	105
78479	Soccer ball	ACME	307

Tables can be defined and accessed in several ways on the system. One way to describe and access tables on the system is to use a language like Structured Query Language (SQL). SQL is the standard IBM database language and provides the necessary consistency to enable distributed data processing across different systems.

Another way to describe and access tables on the system is to describe physical and logical files using data description specifications (DDS) and access tables using file interfaces (for example, read and write high-level language statements).

SQL uses different terminology from that used on the IBM i operating system. For most SQL objects, there is a corresponding IBM i system object. The following table shows the relationship between SQL relational database terms and system terms.

Table 2. Relationship of SQL terms to system terms

SQL term	System term
relational database. A database that can be perceived as a set of tables and can be manipulated in accordance with the relational model of data. There are three types of relational databases a user can access from a IBM i environment, as listed under the System term column. For more information, see the Relational database topic.	**system relational database or system database**. All the database objects that exist on disk attached to the system that are not stored on independent auxiliary storage pools. **user relational database or user database**. All the database objects that exist in a single independent auxiliary storage pool group along with those database objects that are not stored in independent auxiliary storage pools. **Notes:** • The IBM i operating system can be host to multiple relational databases if independent auxiliary storage pools are configured on the system. There is always one system relational database, and there can be one or more user relational databases. Each user database includes all the objects in the system database. • The user should be aware that from a commitment control point of view, the system database is treated as a separate database, even when from an SQL point of view, it is viewed as being included within a user database. For more information, see the Troubleshooting transactions and commitment control topic. **remote relational database, or remote database**. A database that resides on IBM i or another system that can be accessed remotely.
schema. Consists of a library, a journal, a journal receiver, an SQL catalog, and an optional data dictionary. A schema groups related objects and allows you to find the objects by name. **Note:** A schema is also commonly referred to as a collection.	**library**. Groups related objects and allows you to find the objects by name.
table. A set of columns and rows.	**physical file**. A set of records.
row. The horizontal part of a table containing a serial set of columns.	**record**. A set of fields.
column. The vertical part of a table of one data type.	**field**. One or more bytes of related information of one data type.
view. A subset of columns and rows of one or more tables.	**logical file**. A subset of fields, records or both of up to 32 physical files.
index. A collection of data in the columns of a table, logically arranged in ascending or descending order.	A type of logical file.
package. An object that contains control structures for SQL statements to be used by an application server.	**SQL package**. Has the same meaning as the SQL term.

Table 2. Relationship of SQL terms to system terms (continued)

SQL term	System term
catalog. A set of tables and views that contains information about tables, packages, views, indexes, and constraints. The catalog views in QSYS2 contain information about all tables, packages, views, indexes, and constraints on the IBM i operating system. Additionally, an SQL schema contains a set of these views that only contains information about tables, packages, views, indexes, and constraints in the schema.	No similar object. However, the Display File Description (DSPFD) command and the Display File Field Description (DSPFFD) command provide some of the same information that querying an SQL catalog provides.

A distributed relational database consists of a set of tables and other objects that are spread across different but interconnected databases. The database management system may use DRDA protocols to access any of these interconnected databases. An application connection to a database in such an environment is one of two types: local or DRDA. There is, at most, only one local optimized database connection per activation group. Any additional database connections must use DRDA.

In DRDA terminology, an application requester (AR) is the code that handles the application end of a distributed connection. The AR is the application that is requesting data.

An application server (AS) is the code that handles the database end of the connection.

A simple distributed relational database is shown in the following figure where the application runs on one system, and the database is located on another system.

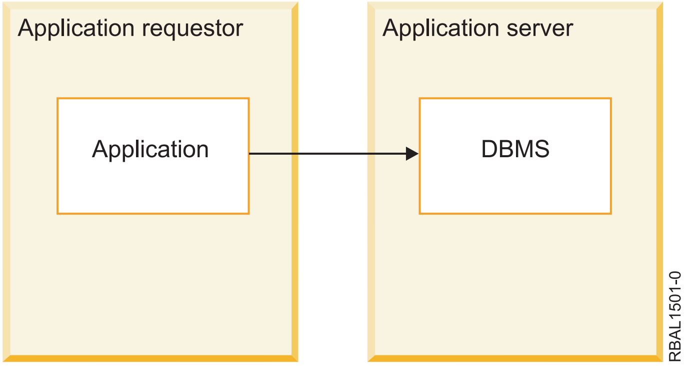


Figure 1. A simple distributed relational database

DRDA also supports multi-tier connections between an application requester and a server. In this topology, the server that an application requester connects to is an application server, but any other server further downstream is called a database server (DS) as it does not interact directly with the application requester. In addition, to highlight its role as neither the system where a database request originates nor the system that performs the database function for the request, each application server or database server between an application requester and the final database server is also called an intermediate server.

A more complex distributed relational database is shown in the following figure where the application runs on one system, and the database management system running on a second system routes a request to a database server located on a third system.

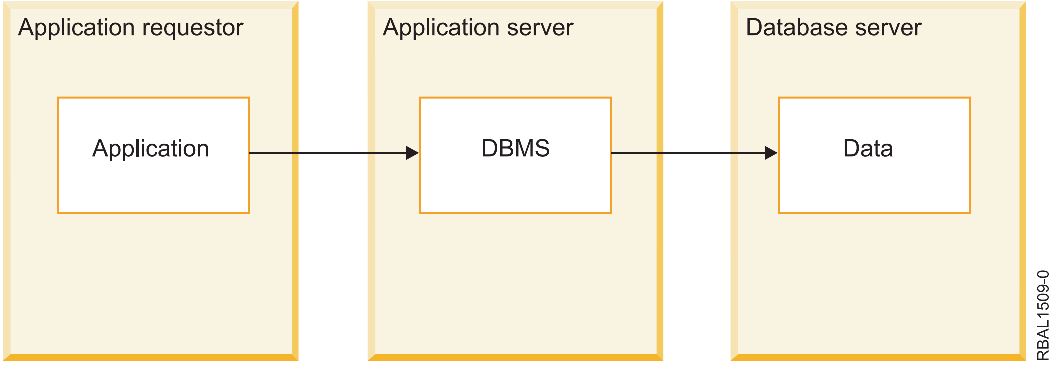


Figure 2. A more complex distributed relational database

The term *client* is often used interchangeably with AR, and *server* with AS or DS.

A *unit of work* is one or more database requests and the associated processing that make up a completed piece of work as shown in the following figure. A simple example is taking a part from stock in an inventory control application program. An inventory program can tentatively remove an item from a shop inventory account table and then add that item to a parts reorder table at the same location. The term *transaction* is another expression used to describe the unit of work concept.

In the preceding example, the unit of work is not complete until the part is both removed from the shop inventory account table and added to a reorder table. When the requests are complete, the application program can *commit* the unit of work. This means that any database changes associated with the unit of work are made permanent.

With unit of work support, the application program can also *roll back* changes to a unit of work. If a unit of work is rolled back, the changes made since the last commit or rollback operation are not applied. Thus, the application program treats the set of requests to a database as a unit.

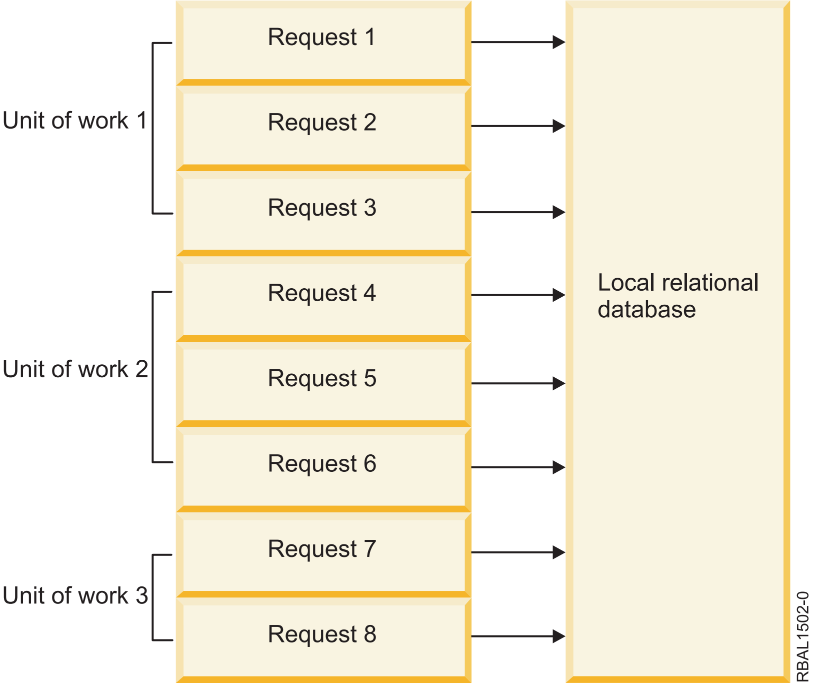


Figure 3. Unit of work in a local relational database

Related concepts
Relational database
Troubleshooting transactions and commitment control

XA transaction support for commitment control

Related reference

Display File Description (DSPFD) command

Display File Field Description (DSPFFD) command

Remote unit of work

Remote unit of work (RUW) is a form of distributed relational database processing in which an application program can access data on a remote database within a unit of work.

A remote unit of work can include more than one relational database request, but all requests must be made to the same remote database. All requests to a relational database must be completed (either committed or rolled back) before requests can be sent to another relational database. This is shown in the following figure.

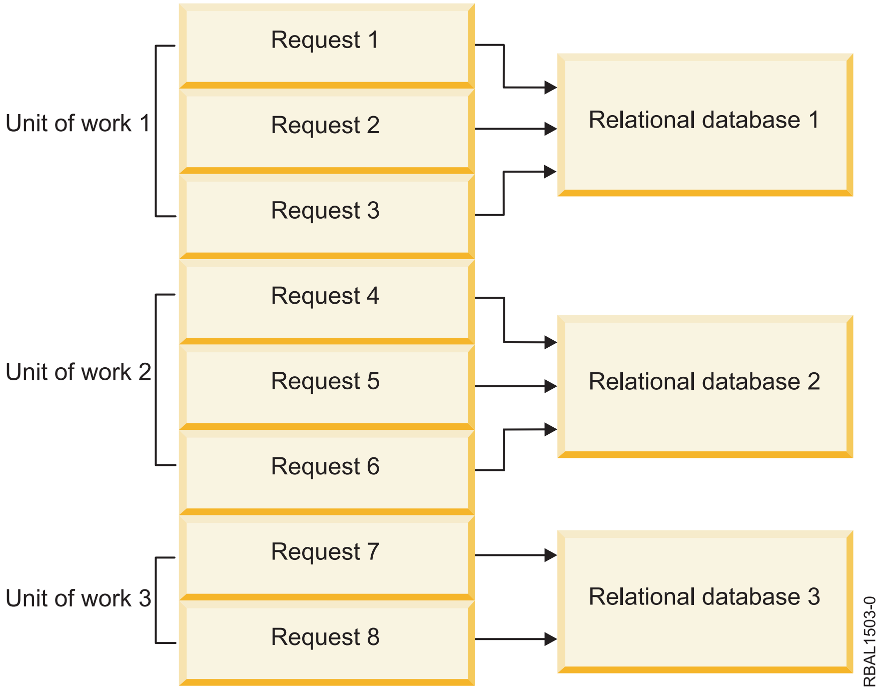


Figure 4. Remote unit of work in a distributed relational database

Remote unit of work is application-directed distribution because the application program must connect to the correct relational database system before issuing the requests. However, the application program only needs to know the name of the remote database to make the correct connection.

Remote unit of work support enables an application program to read or update data at more than one location. However, all the data that the program accesses within a unit of work must be managed by the same relational database management system. For example, the shop inventory application program must commit its inventory and accounts receivable unit of work before it can read or update tables that are in another location.

In remote unit of work processing, each computer has an associated relational database management system and an associated application requester program that help process distributed relational data requests. This allows you or your application program to request remote relational data in much the same way as you request local relational data.

Distributed unit of work

Distributed unit of work (DUW) enables a user or application program to read or update data at multiple locations within a unit of work.

Within one unit of work, an application running on one system can direct SQL requests to multiple remote database management systems using the SQL supported by those systems. For example, the shop inventory program can perform updates to the inventory table on one system and the accounts receivable table on another system within one unit of work. The following figure illustrates this idea.

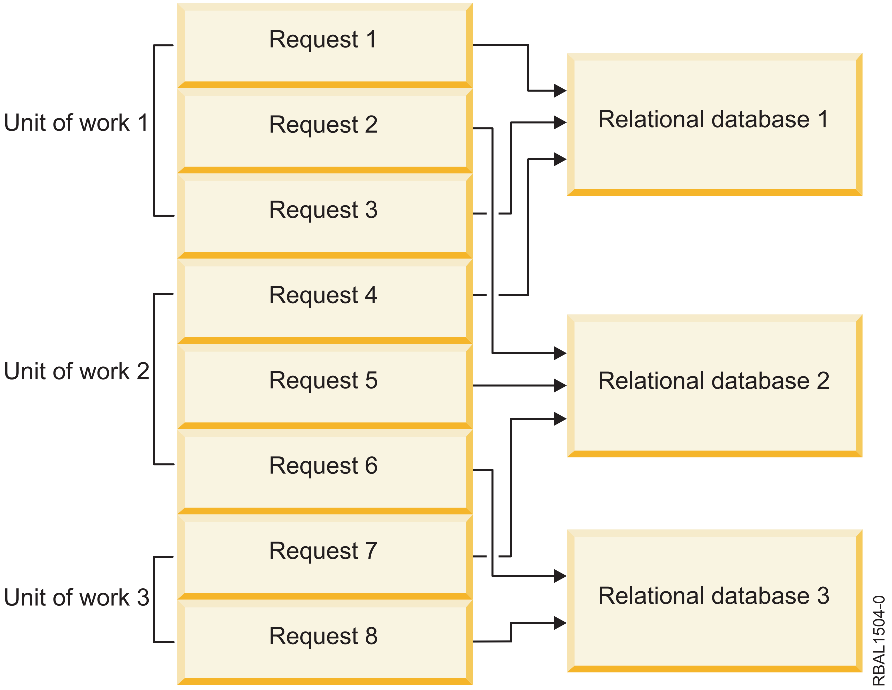


Figure 5. Distributed unit of work in a distributed relational database

The target of the requests is controlled by the user or application with SQL statements such as CONNECT TO and SET CONNECTION. Each SQL statement must refer to data at a single location.

When the application is ready to commit the work, it initiates the commit; commitment coordination is performed by a synchronization-point manager or a transaction manager.

DUW allows update access or read-only access to multiple database management systems in one unit of work.

Whether an application can update a given database management system in a unit of work is dependent on the level of DRDA (if DRDA is used to access the remote relational database) and the order in which the connections and updates are made.

Distributed request

Distributed request enables a user or application program to read or update data at one or more databases within a single SQL statement.

Within one SQL statement, an application running against a local database can direct SQL requests to one or more remote databases. For example, a program can perform updates to table A on database 1 and table B on database 2 within one SQL statement. The following figure illustrates this idea.

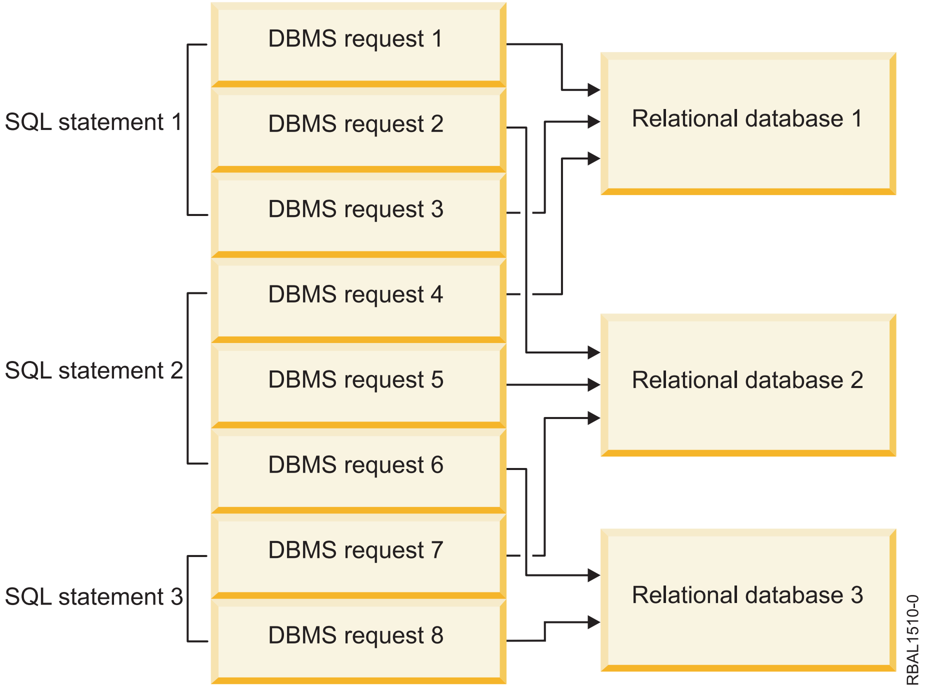


Figure 6. Distributed request in a distributed relational database

IBM i distributed relational database supports a subset of the distributed request functionality. It allows update access or read-only access to a single local or remote database in one SQL statement.

The target database of the SQL statement is controlled by the user or application by specifying database qualifiers on referenced objects. For more information on object qualification, see SQL Reference.

Remote database connections are controlled by the database management system. The user or application can not use CONNECT TO, SET CONNECTION, RELEASE or DISCONNECT against these remote database connections. Remote database connections are tied to the local database connection. They are started as needed and ended when the local database connection is ended.

Special registers and global variables are propagated over remote database connections. If special registers or global variables do not exist on the remote database, they are ignored. If global variables exist but do not have the same attributes, the operation fails. Updates to the local special registers or global variables are propagated as needed over existing remote database connections.

When the application is ready to commit the work, it initiates the commit; commitment coordination is performed by a synchronization-point manager or a transaction manager.

Whether an application can update a given database in a SQL statement is dependent on the level of DRDA (if DRDA is used to access the remote relational database) and the order in which the connections and updates are made.

Three-Part Names
A *three-part name* specifies the relational database, the schema (or library depending on naming method), and the name of an object for use in an SQL statement.

An application running against the local database can direct SQL requests to either the local database or a remote database. The database qualifiers within an SQL statement are resolved and if all refer to the same database, the SQL statement is processed at that database. If the database qualifiers within an SQL statement do not all refer to the same database, an SQL0512 is signalled.

Explicit three-part names

The following example shows an SQL statement with an explicit remote RDB qualifier (RemoteRDB).

```
CRTSQLxxx PGM(MySchema/MyPgm) RDB(LocalRDB)
```

```
  EXEC SQL SELECT * INTO :SERVICE FROM RemoteRDB/MySchema/MyTableA;
  EXEC SQL COMMIT;
```

Implicit three-part names

The following example shows two SQL statements, each with an implicit local RDB qualifier.

```
CRTSQLxxx PGM(MySchema/MyPgm) RDB(LocalRDB)
  EXEC SQL SELECT * INTO :SERVICE FROM MyTableA;
  EXEC SQL CALL MySchema/MyProcedureA;
  EXEC SQL COMMIT;
```

For more information on implicit RDB qualifiers, see SQL Reference.

Aliases

An *alias* is a substitute name for a table or view. Aliases are unique in that they can refer to a based on table or view that is on the current database or a remote database.

An application running against the local database can direct SQL requests to either the local database or a remote database. If there are aliases in the statement, the database qualifier of the based on table or view is used to determine the location to process the SQL statement.

The following example shows an SQL statement using an alias with an explicit remote database qualifier (RemoteRDB) for the based on table.

```
CRTSQLxxx PGM(MySchema/MyPgm) RDB(LocalRDB)
  EXEC SQL CREATE ALIAS MySchema/MyTableA FOR RemoteRDB/MySchema/MyTableA;
  EXEC SQL SELECT * INTO :SERVICE FROM MySchema/MyTableA;
  EXEC SQL COMMIT;
```

Other distributed relational database terms and concepts

On IBM systems, some distributed relational database support is provided by the DB2® for Linux, UNIX, and Windows, and IBM DB2 DataPropagator for iSeries, V8.1 licensed programs. In addition, you can use some of these concepts when writing IBM i application programs.

Db2® for i supports both the remote unit of work and distributed unit of work with Advanced Program-to-Program Communication (APPC) and TCP/IP communications. A *distributed request* is an SQL query directed to two or more data sources in a federated database system. This type of distributed relational database access enables a user or application program to issue a single SQL statement that can read or update data at multiple locations.

Tables in a distributed relational database do not have to differ from one another. Some tables can be exact or partial copies of one another. Extracts, snapshots, and replication are terms that describe types of copies using distributed processing.

Extracts are user-requested copies of tables. The copies are extracted from one database and loaded into another specified by the user. The unloading and loading process might be repeated periodically to obtain updated data. Extracts are most useful for one-time or infrequent occurrences, such as read-only copies of data that rarely changes.

Snapshots are read-only copies of tables that are automatically made by a system. The system refreshes these copies from the source table on a periodic basis specified by the user—perhaps daily, weekly, or monthly. Snapshots are most useful for locations that seek an automatic process for receiving updated information on a periodic basis.

Data *replication* means the system automatically updates copies of a table. It is similar to snapshots because copies of a table are stored at multiple locations. Data replication is most effective for situations that require high reliability and quick data retrieval with few updates.

Tables can also be split across computer systems in the network. Such a table is called a *distributed table*. Distributed tables are split either horizontally by rows or vertically by columns to provide easier local reference and storage. The columns of a vertically distributed table reside at various locations, as do the rows of a horizontally distributed table. At any location, the user still sees the table as if it were kept in a single location. Distributing tables is most effective when the request to access and update certain portions of the table comes from the same location as those portions of the table.

Related concepts

Distributed relational database on IBM i
Db2 for i provides all the database management functions for IBM i. Distributed relational database support on the system is an integral part of the operating system, just as is support for communications, work management, security functions, and other functions.

Distributed Relational Database Architecture support

Distributed Relational Database Architecture (DRDA) support for distributed relational database processing is used by IBM relational database products. DRDA support defines protocols for communication between an application program and a remote relational database.

DRDA support provides distributed relational database management in both IBM and non-IBM environments. In IBM environments, relational data is managed with the following programs:

- DB2 for AIX®
- DB2 for HP-UX
- Db2 for i
- DB2 for Linux
- DB2 for Sun Solaris
- DB2 for VSE/VM
- DB2 for Windows
- DB2 for z/OS®

DRDA support provides the structure for access to database information for relational database managers operating in *like* and *unlike* environments. For example, access to relational data between two or more systems with Db2 for i databases is distribution in a like environment. Access to relational data between Db2 for i and another type of system or a client different from the one embedded in IBM i is distribution in an unlike environment. One specific example of this is access to relational data between Db2 for i and IBM DB2 Universal Driver for Structured Query Language for Java™ (SQLJ) and Java Database Connectivity (JDBC).

SQL is the standard IBM database language. It provides the necessary consistency to enable distributed data processing across like and unlike operating environments. Within DRDA support, SQL allows users to define, retrieve, and manipulate data across environments that support a DRDA implementation.

The Distributed Relational Database Architecture is an extension of the distributed data management (DDM) architecture. However, DRDA and DDM methods for accessing data are different. DRDA is an extension of SQL whereas DDM is an extension of native I/O.

Using distributed relational database processing, an application can connect to a remote system using the relational database directory on the local system. The relational database directory provides the necessary links between a relational database name and the communications path to that database. An application running under the distributed relational database only has to identify the database name and run the SQL statements needed for processing.

DRDA and CDRA support

A distributed relational database might not only span different types of computers, but those computers might be in different countries or regions.

Identical systems can encode data differently depending on the language used on the system. Different systems encode data differently. For instance, a z/OS product, a IBM i product, and a Windows system that are running the DB2 for Linux, UNIX, and Windows licensed program encode numeric data in their own unique formats. In addition, a z/OS and a IBM i product use the EBCDIC encoding scheme to encode character data, while a Windows system that is running DB2 LUW uses an ASCII encoding scheme.

For numeric data, these differences do not matter. *Unlike systems* that provide Distributed Relational Database Architecture (DRDA) support automatically convert any differences between the way a number is represented in one computer system to the way it is represented in another. For example, if an IBM i application program reads numeric data from a Db2 for i database, Db2 for i sends the numeric data in the z/OS format, and the IBM i database management system converts it to IBM i numeric format.

However, the handling of character data is more complex, but this too can be handled within a distributed relational database.

Character conversion with CDRA

Not only can there be differences in encoding schemes, such as Extended Binary Coded Decimal Interchange Code (EBCDIC) versus American Standard Code for Information Interchange (ASCII), but there can also be differences related to language.

For instance, systems configured for different languages can assign different characters to the same code, or different codes to the same character. For example, a system configured for U.S. English can assign the same code to the character } that a system configured for the Danish language assigns to å. But those two systems can assign different codes to the same character such as $.

If data is to be shared across different systems, character data needs to be seen by users and applications the same way. In other words, a Windows user in New York and an IBM i user in Copenhagen both need to see a $ as a $, even though $ might be encoded differently in each system. Furthermore, the user in Copenhagen needs to see a }, if that is the character that was stored at New York, even though the code might be the same as a Danish å. In order for this to happen, the $ must be converted to the proper character encoding for a Windows system (that is, U.S. English character set, ASCII), and converted back to Danish encoding when it goes from New York to Copenhagen (that is, Danish character set, EBCDIC). This sort of character conversion is provided for by IBM i as well as the other IBM distributed relational database managers. This conversion is done in a coherent way in accordance with the Character Data Representation Architecture (CDRA).

CDRA specifies the way to identify the attributes of character data so that the data can be understood across systems, even if the systems use different character sets and encoding schemes. For conversion to happen across systems, each system must understand the attributes of the character data it is receiving from the other system. CDRA specifies that these attributes be identified through a coded character set identifier (CCSID). All character data in DB2 for z/OS, DB2 for VM, and the IBM i database management systems have a CCSID, which indicates a specific combination of encoding scheme, character set, and code page. All character data in an Extended Services environment has only a code page (but the other database managers treat that code page identification as a CCSID). A *code page* is a specific set of assignments between characters and internal codes.

For example, CCSID 37 means encoding scheme 4352 (EBCDIC), character set 697 (Latin, single-byte characters), and code page 37 (USA/Canada country extended code page). CCSID 5026 means encoding scheme 4865 (extended EBCDIC), character set 1172 with code page 290 (single-byte character set for Katakana/Kanji), and character set 370 with code page 300 (double-byte character set for Katakana/Kanji).

DRDA-enabled systems include mechanisms to convert character data between a wide range of CCSID-to-CCSID pairs and CCSID-to-code page pairs. Character conversion for many CCSIDs and code pages is already built into these products. For more information about CCSIDs supported by IBM i, see the IBM i globalization topic.

Related concepts

IBM i globalization

Related reference

Coded character set identifier

Support for the national language of any country requires the proper handling of a minimum set of characters.

Application requester driver programs

An *application requester driver* (ARD) program is a type of exit program that enables SQL applications to access data managed by a database management system other than Db2 for i.

An IBM i client calls the ARD program during the following operations:

- The package creation step of SQL precompiling, performed using the **`Create Structured Query Language Package (CRTSQLPKG)`** command or CRTSQL*xxx* commands, when the relational database (RDB) parameter matches the RDB name corresponding to the ARD program.
- Processing of SQL statements when the current connection is to an RDB name corresponding to the ARD program.

These calls allow the ARD program to pass the SQL statements and information about the statements to a remote relational database and return results back to the application requester (AR). The AR then returns the results to the application or the user. Access to relational databases accessed by ARD programs appear like access to DRDA application servers in the unlike environment.

The ARD program is registered in the system by use of the **`Add Relational Database Directory Entry (ADDRDBDIRE)`** command. One of the parameters that is specified is the library in which the program is located. For a system configured with independent auxiliary storage pools, the ARD program must reside in a library in the system database (a library that is part of the system ASP or a configured basic ASP).

Related concepts

Application programming interfaces

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Create Structured Query Language Package (CRTSQLPKG) command

Distributed relational database on IBM i

Db2 for i provides all the database management functions for IBM i. Distributed relational database support on the system is an integral part of the operating system, just as is support for communications, work management, security functions, and other functions.

The IBM i operating system can be part of a distributed relational database network with other systems that support a Distributed Relational Database Architecture (DRDA) implementation. IBM i can be an application requester (AR) or an application server (AS) in either like or unlike environments. Distributed relational database implementation on the IBM i operating system supports remote unit of work (RUW) and distributed unit of work (DUW). RUW allows you to submit multiple requests to a single database within a single unit of work, and DUW allows requests to multiple databases to be included within a single unit of work.

Using DUW support, you can decrement the inventory count of a part on one system and increment the inventory count of a part on another system within a unit of work, and then commit changes to these remote databases at the conclusion of a single unit of work using a two-phase commit process. Db2 for i does not support distributed requests, so you can only access one database with each SQL statement. The level of support provided in an application program depends on the level of support available on the application server (AS) and the order in which connections and updates are made.

In addition to DRDA access, application requester driver (ARD) programs can be used to access databases that do not support DRDA. Connections to relational databases accessed through ARD programs are treated like connections to unlike servers. Such connections can coexist with connections to DRDA application servers, connections to the local relational database, and connections which access other ARD programs.

On the IBM i operating system, the distribution functions of snapshots and replication are not automatically performed by the system. You can install and configure the DB2 DataPropagator product on IBM i to perform these functions. Also, you can use these functions in user-written application programs.

More information about how you can organize these functions in a distributed relational database is discussed in the topic Data availability and protection.

On the IBM i operating system, the distributed request function is not directly supported. However, the DataJoiner product can perform distributed queries, joining tables from a variety of data sources. DataJoiner works synergistically with DataGuide, a comprehensive information catalog in the IBM Information Warehouse family of products. DataGuide provides a graphical user interface to complete information listings about a company's data resources.

The IBM i licensed program includes runtime support for SQL. You do not need the Db2 for i Query Manager and SQL Development Kit licensed program installed on a Db2 for i application requester or application server to process distributed relational database requests or to create an SQL collection on IBM i. However, you need the Db2 for i Query Manager and SQL Development Kit program to precompile programs with SQL statements, to run interactive SQL, or to run Db2 for i Query Manager.

Communications support for the DRDA implementation on the IBM i operating system is provided under either TCP/IP or the IBM Systems Network Architecture (SNA) through the Advanced Program-to-Program Communication (APPC) protocol, with or without Advanced Peer-to-Peer Networking (APPN).

Related concepts

Other distributed relational database terms and concepts
On IBM systems, some distributed relational database support is provided by the DB2® for Linux, UNIX, and Windows, and IBM DB2 DataPropagator for iSeries, V8.1 licensed programs. In addition, you can use some of these concepts when writing IBM i application programs.

Connecting to a distributed relational database
What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another system.

Data availability and protection
In a distributed relational database environment, data availability involves not only protecting data on an individual system in the network, but also ensuring that users have access to the data across the network.

APPC, APPN and HPR

Configuring TCP/IP

OptiConnect

Example: Spiffy Corporation distributed relational database

The Spiffy Corporation is used in several IBM manuals to describe distributed relational database support. In this topic collection, this fictional company has been changed somewhat to illustrate IBM i support for DRDA in a network of IBM i products.

Examples used throughout this topic collection illustrate particular functions, connections, and processes. These might not correspond exactly to the examples used in other distributed relational database publications, but an attempt has been made to make them look familiar.

Though the Spiffy Corporation is a fictional enterprise, the business practices described here are modeled after those in use in several companies of similar construction. However, this example does not attempt to describe all that can be done using a distributed relational database, even by this example company.

Spiffy organization and system profile

Spiffy Corporation is a fictional national product distributor that sells and services automobiles, among other products, to retail customers through a network of regional offices and local dealerships.

Given the high competitiveness of today's automobile industry, the success of an operation like the Spiffy Corporation depends on high-quality servicing and timely delivery of spare parts to the customer. To meet this competition, Spiffy has established a vast service network incorporated within its dealership organization.

The dealership organization is headed by a central vehicle distributor located in Chicago, Illinois. There are several regional distribution centers across North America. Two of these are located in Minneapolis, Minnesota and Kansas City, Missouri. These centers minimize the distribution costs of vehicles and spare parts by setting up regional inventories. The Minneapolis regional center serves approximately 15 dealerships while the Kansas City center serves as many as 30 dealerships.

The figure here illustrates a system organization chart for Spiffy Corporation.

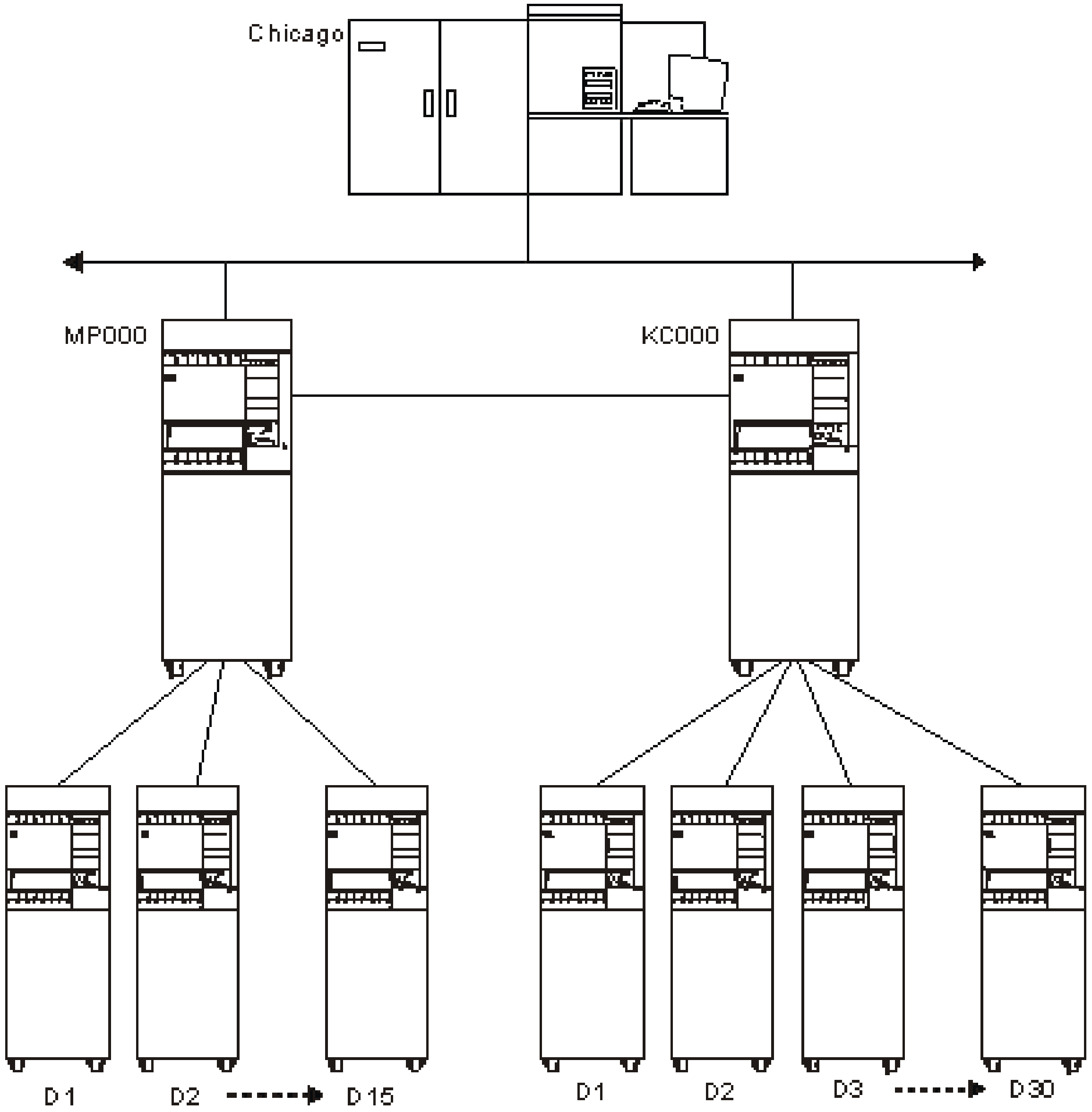


Figure 7. The Spiffy Corporation system organization

Spiffy is in the process of building up a nationwide integrated telecommunications network. For the automobile division, they are setting up a network of IBM i products for the regional distribution centers and the dealerships. These are connected to a System z® platform at the central vehicle distributor. This network is considered a vital business asset for maintaining the competitive edge.

The central distributor runs DB2 for z/OS on its System z platform with relevant decision support software. This system is used because of the large amounts of data that must be handled at any one time in a variety of application programs. The central vehicle distributor system is not dedicated to automobile division data processing. The distributor system must handle work and processes, for the corporation, that do not yet operate in a distributed database environment. The regional centers are running IBM i products. They use APPC/APPN with SNADS and 5250 display station pass-through using an SDLC protocol.

All of the dealerships use IBM i products that vary in size. These systems are connected to the regional office using SDLC protocol. The largest dealerships have a part time programmer and a system operator to tend to the data processing functioning of the enterprise. Most of the installations do not employ anyone with programming expertise, and some of the smaller locations do not employ anyone with more than a general knowledge of computers.

Business processes of the Spiffy Corporation Automobile Service

The Spiffy Corporation automobile division has business practices that are automated in this distributed relational database environment.

To keep the examples from becoming more complicated than necessary, consider just those functions in the company that pertain to vehicle servicing.

Dealerships can have a list of from 2000 to 20 000 customers. This translates to 5 service orders per day for a small dealership and up to 50 per day for a large dealership. These service orders include scheduled maintenance, warranty repairs, regular repairs, and parts ordering.

The dealers stock only frequently needed spare parts and maintain their own inventory databases. Both regional centers provide parts when requested. Dealer inventories are also stocked on a periodic basis by a forecast-model-controlled batch process.

Distributed relational database administration for the Spiffy Corporation

Spiffy Corporation requires that each dealership have one or more IBM i products and that those systems must be available to the network at certain times.

However, each dealership manages its data processing resources and procedures as a stand-alone enterprise. The size of the system and the number of business processes that are automated on it are determined by each dealership's needs and the resources available to it.

The Spiffy Corporation requires all dealerships to be active in the inventory distributed relational database. Because the corporation operates its own dealerships, it has a full complement of dealership software that might or might not access the distributed relational database environment. The Spiffy dealerships use the full set of software tools. Most of the private franchises also use them, because they are tailored specifically to the Spiffy Corporation way of doing business.

The regional distribution centers manage the inventory for their region. They also function as the database administrator for all distributed database resources used in the region. The responsibilities involved vary depending on the level of data processing competency at each dealership. The regional center is always the first contact for help for any dealership in the region.

The Minneapolis regional distribution center has a staff of IBM i programmers with a wide range of experience and knowledge about the systems and the network. The dealership load is about one half that of other regional centers to allow this center to focus on network-wide IBM i support functions. These functions include application program development, program maintenance, and problem handling.

Listed here are the database responsibilities for each level of activity in the network:

Dealerships

- Perform basic system operation and administration
- Enroll local users

Regional distribution centers

- Set up data processing for new dealerships
- Disperse database resources for discontinued dealerships
- Enroll network users in region
- Maintain inventory for region

- Develop service plans for dealerships
- Operate help desk for dealerships

Other activities

In addition to the regional distribution center activities, the Minneapolis IBM i competency center does the following activities:

- Develop applications
- Operate help desk for regional centers
- Tune database performance
- Resolve database problems

Many examples in this topic show the process of obtaining a part from inventory in order to schedule customer service or repairs. Others show distributed relational database administration tasks used to set up, secure, monitor, and resolve problems for servers in the Spiffy Corporation distributed relational database network.

DDM overview

This topic describes the purpose of distributed data management (DDM), the functions that DDM supplies, and the concepts of IBM i DDM.

DDM is part of the IBM i licensed program. IBM i DDM as a source supports Level 2.0 and below of the DDM architecture. IBM i DDM as a target supports Level 2.0 and below for *record file* (a file on disk in which the data is read and written in records) types and Level 3.0 and below of the DDM architecture for stream files (documents) and directories (folders).

IBM i DDM support allows application programs or users to access data files that reside on remote systems, and also allows remote systems to access data files on the local IBM i operating system, as shown in Figure 8 on page 17. Any system that supports the DDM architecture as a client system can access data (if authorized to do so) on any other system to which it is attached. The attached system must support DDM as a *server system* (the system that receives a request from another system to use one or more files located on the system). However, the client and server systems must support compatible subsets and levels of the DDM architecture.

The folder management services (FMS) support allows personal computer users to access folders and documents that reside on an IBM i server system. Remote systems that support Level 3.0 or Level 2.0 of the DDM architecture for the stream access method can access folders and documents on the local system.

DDM extends the file accessing capabilities of the IBM i database management support. In this topic collection, *database management* refers to the system function that controls **local** file processing; that is, it controls access to data in files stored on the local system, and it controls the transfer of that data to requesting programs on the same system.

Distributed data management controls remote file processing. DDM enables IBM i application programs to access data files stored on another system supporting DDM. Similarly, other systems that have DDM can access files in the database of the local system. DDM makes it easier to distribute file processing between two or more systems.

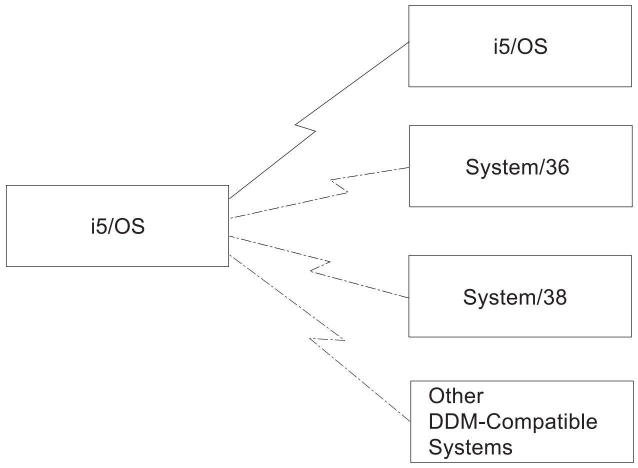


Figure 8. Client and server systems

Systems that use DDM communicate with each other using the Advanced Program-to-Program Communication (APPC) support, Advanced Peer-to-Peer Networking (APPN) support, or TCP/IP. See the *Communications Management* manual on the IBM i PDF files and manuals page and the APPC, APPN, and HPR topic for information needed to use APPC and APPN.

Folder management services (FMS) allows local access to documents or folders that are on the IBM i operating system. Personal computers can access folder management functions on the system by using DDM.

Note: Distributed data management for the IBM Personal Computer uses the IBM i portion of the IBM i Access Family licensed program.

As shown in Figure 9 on page 18, the system on which a user application issues a request involving a remote file is called a *client system*. The system that receives the request for one of its files is called the *server system*. A system can be both a client and server system for separate requests received at the same time.

Using DDM, an application program can get, add, change, and delete data records in a file that exists on a server system. It can also perform file-related operations, such as creating, deleting, renaming, or copying a file from the server system to the client system.

When DDM is in use, neither the application program nor the program user needs to know if the file that is needed exists locally or on a remote system. DDM handles remote file processing in essentially the same way as local file processing is handled on the local system, and the application program normally does not receive any indication of where the requested file is located. (However, in error conditions, messages are returned to the user that indicate, when necessary, that a remote system was accessed.) Informational messages about the use of server system files are included in the client system's job log.

When DDM is to be used, only application programmers need to know where the file is located and, using control language (CL) commands outside of the high-level language (HLL) programs, they can control which file is used. However, the programmers can also choose to use specific recovery functions to handle certain communications failures; the HLL programs might need to be changed to include handling any such failure.

Therefore, BASIC, ILE COBOL, ILE RPG, ILE C, and IBM i programs that are compiled to process database files on the local system might not need to be changed or recompiled for DDM to process those same files when they are moved to or exist on a remote system.

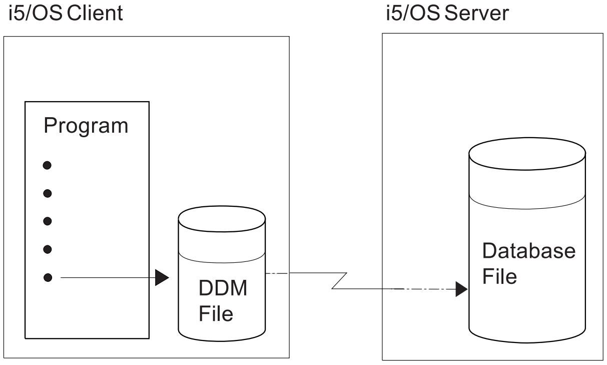


Figure 9. Moving a program from a client to a server system

Related concepts

Planning and design for DDM
There are several requirements that must be met for distributed data management (DDM) to be used properly.

System compatibility

DDM can be used to communicate between systems that are architecturally different.

Although the architectures of the IBM i operating system and System/36 are different, these systems can use DDM to access files in each other's database. To successfully communicate with each other, each system must have an implementation of DDM that is compatible with Level 2.0 or below of the IBM DDM architecture. Also, each type of system might use all or only part of the IBM DDM architecture or might have extensions to the architecture.

If you are communicating with any systems other than IBM i, you must consider the level of DDM support provided by those servers for such things as unique security considerations.

For a list of the DDM architecture manuals that supply the details about Level 3.0 or below of the IBM DDM architecture, see Architecture books.

Related reference

Architecture books
Listed here are the DDM and DRDA architecture books.

Overview of DDM functions

Here is an overview of the types of DDM functions on a server system.

The following *file* operations, normally specified in HLL programs, can be done on files at server systems:

- Allocating, opening, or closing one or more files.
- Reading, writing, changing, or deleting records in a file.

The following *file* and *nonfile* operations, normally specified in CL programs or by CL commands, can be done on files at the server systems:

- Copying the contents of a file.
- Performing operations on physical or logical file members (such as adding, clearing, or removing members), but only if the target is an IBM i or System/38.
- Accessing remote *files* for nondata purposes, such as:
  - Displaying information about one or more files, using commands such as **`Display File Description (DSPFD)`** and **`Display File Field Description (DSPFFD)`**. These commands can display the file attributes of the DDM file on the client system or the file or field attributes of the remote file on the server system.
  - Controlling the locking of files on the server system, using the **`Allocate Object (ALCOBJ)`** and **`Deallocate Object (DLCOBJ)`** commands.

- Deleting, renaming, creating, and changing files using the **Delete File (DLTF)**, **Rename Object (RNMOBJ)**, **Create Physical File (CRTPF)**, **Create Source Physical File (CRTSRCPF)**, **Create Logical File (CRTLF)**, **Change Physical File (CHGPF)**, **Change Logical File (CHGLF)**, and **Change Source Physical File (CHGSRCPF)** commands.

- Accessing remote *systems* for nondata purposes:
  - Sending a CL command to the server system (an IBM i and a System/38 only) so it can be run there, instead of on the client system (where it might not be useful to run it), using the **Submit Remote Command (SBMRMTCMD)** command. The **SBMRMTCMD** command is the method you use to move, save, or restore files on a server system. For example, a **Move Object (MOVOBJ)** command might be sent to move a database file on the server system. (For typical uses of the **SBMRMTCMD** command, refer to its description in Using CL and DDS with DDM or refer to the CL topic for a more complete description.)

Various other nonfile-related operations can be done on the server system.

Related concepts

Control language

Using CL and DDS with DDM
This topic contains DDM-related information about specific control language (CL) commands, data description specifications (DDS) considerations, DDS keywords, and DDM user profile authority.

Basic DDM concepts

Although DDM supports other functions besides opening and accessing remote files, the concepts described in this topic collection deal primarily with remote file accessing.

From a user's viewpoint, accessing data on a remote system is much the same as accessing data on the local system. The main difference is the additional time needed for the data link to pass the data between the systems whenever the remote file is accessed. Otherwise, the user or application program does not need to know whether the data being accessed came from a local or remote file. Refer to Performance considerations for DDM for additional considerations.

For DDM IBM i-to-IBM i file processing, remote file processing is done much the same as local file processing. The purpose of this topic collection is to describe the things that are different for DDM. Also, because other systems can use DDM, those considerations and concepts are covered as needed to enable the programmer to successfully prepare the system for using DDM.

The DDM concepts in this topic collection describe mainly IBM i-to-IBM i remote file processing. For purposes of illustration, concepts that relate to System/36 and System/38 are shown in some examples. If you are using DDM on both System/36 and IBM i, you should be aware that the concepts for both types are similar, except in the way they point to the remote file: IBM i and System/38 use a separate *DDM file* to refer to each remote file to be accessed; System/36 uses a network resource directory that contains one *network resource directory entry* for each remote file to be accessed.

Related concepts

Additional DDM concepts
Most users of DDM will not need the information in the remainder of these topics; it is intended primarily for experienced programmers who need to know more about DDM.

Performance considerations for DDM
These topics provide information to help you improve performance when using DDM and also provide some information about when to use something other than DDM to accomplish some functions.

Parts of DDM

DDM consists of several parts to handle remote file processing among the systems using DDM.

- Source DDM (SDDM)
- Target DDM (TDDM)
- DDM file

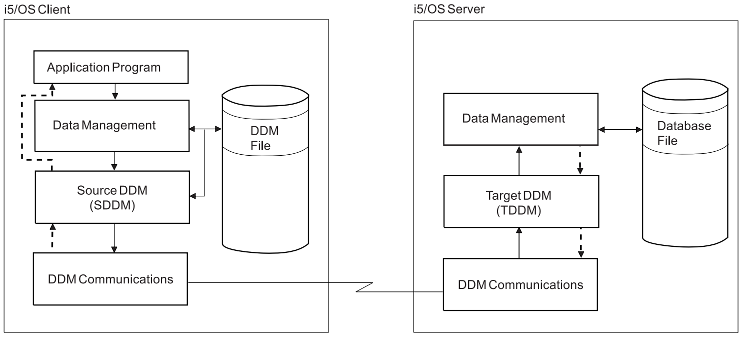


Figure 10. Communicating with DDM

The preceding figure shows how the basic parts involved in DDM communications on both systems relate to each other.

When a DDM file is accessed by a client system user or program, a DDM conversation is started between SDDM and TDDM for the job in which the program or user is operating.

Parts of DDM: Source DDM

The support on the client (or local) system is started, as needed, within a source job to do DDM functions.

The source DDM (SDDM) translates requests for remote file access from client system application programs into DDM requests that are routed to the server system for processing. The SDDM support establishes and manages a DDM conversation with the server system that has the desired remote file.

When an application program first attempts to access a remote file, a search for the requested DDM file is done on the client system. As with local file processing, if the file name is not qualified with a library name, the current library list for the job in which the program is running is searched for the specified file. When the file is found, the system accesses the file, determines that it is a DDM file and starts the SDDM.

When the SDDM is started, it checks to see if a DDM conversation is already active between the source job starting the SDDM and the server system identified by the remote location and mode values in the DDM file. If a conversation that can be used exists, it is used. If not, a program start request is issued to the appropriate server system to start a TDDM (a target job) on the server system to establish a DDM conversation between the SDDM and TDDM. Parameters that are automatically created from information in the DDM file about the remote file are passed when the remote system sends a program start request.

After the TDDM is started, the SDDM can forward each program request to the target job for processing. If, for example, input/output (I/O) operations are to be done on a remote file, the program opens the file and then issues the desired operation requests. The SDDM forwards the open request and the TDDM opens the remote file. Then the SDDM forwards each file operation request to the TDDM, and both of them handle the interchange of data between the application program and the remote file. When a DDM function is being processed, the requesting program waits for the function to be completed and the results to be received, just as it does for local file operations.

Related concepts

IBM i as the client system for DDM

All of these actions, as well as those required on the server system, must complete successfully before any operations (file or nonfile) requested by the source program can be done.

Parts of DDM: Target DDM

A server system job is started on the target (or remote) system as a result of an incoming DDM request and ends when the associated DDM conversation ends.

The target DDM (TDDM) translates DDM requests for remote file access into data management requests on the server system and then handles the return of the information that is to be sent to the client system.

The TDDM is started when the remote system sends a program start request. The TDDM is started as a batch job on the server system. After the TDDM is started and a DDM conversation is established, the

TDDM waits for a request (such as a file open or read operation, or a nonfile-related operation) to be sent by the SDDM.

When the TDDM receives a request to access an object on the server system, it searches for the requested object. If the object was not qualified with a library or path name, the current library list or current directory for the target job is searched.

When the requested object is found, the TDDM passes the first operation requested to database or folder management on the server system, which performs the operation on the object. When the operation is completed, database or folder management services return the results of the operation to the TDDM, which passes it to the SDDM. The SDDM passes the results and any accompanying data (such as records requested on a read operation) to the application program. These actions are repeated for each subsequent I/O operation request received, until the object is closed. If an operation does not complete successfully, the SDDM returns an error message to the program, providing information about the error.

The TDDM and the target job remain active until the DDM conversation is ended by the client system job that started it.

Related concepts

IBM i as the server system for DDM
The IBM i target DDM (or TDDM) is actually a job that runs a DDM-related server system program. It is started when the client system sends a program start request (an SDDM).

Parts of DDM: DDM file

A *DDM file* is a file on the client system that contains the information needed to access a data file on a server system.

The DDM file is a system object that exists on the client system to identify a remote file. It combines the characteristics of a device file and a database file. As a device file, the DDM file refers to a remote location name, local location name, device name, mode, and a remote network ID to identify a remote system as the server system. The DDM file appears to the application program as a database file and serves as the access device between a client system program and a remote file.

However, it is *not* a data file that can be accessed by a program for database operations. Instead, when a client system program specifies the name of a DDM file, the file information is used by DDM to locate the remote file whose data *is* to be accessed.

DDM file information is based on *locations*. The remote location where the remote file is located is specified using the remote location name (RMTLOCNAME) parameter on the **`Create DDM File (CRTDDMF)`** or **`Change DDM File (CHGDDMF)`** command.

The remote file name specified on the **`CRTDDMF`** or **`CHGDDMF`** command must be in the format used by the remote system.

Another use of the DDM file is to submit control language (CL) commands to the server system to run on that system. In this case, the remote file normally associated with the DDM file is ignored.

Related reference

Submit Remote Command (SBMRMTCMD) command
The **`Submit Remote Command (SBMRMTCMD)`** command submits a command using DDM to run on the server system.

DDM file creation using SNA

You can create a DDM file that uses SNA as the communication protocol for connecting with the remote system.

Each DDM file that uses SNA contains the following information.

DDM file value and description of values

DDM file name
The name of the DDM file on the client system that is used to identify a specific remote file.

Remote file name
The actual file name of the remote file; that is, the name by which it is known on the server system. (For a target System/36, this is the file *label* of the remote file.)

Remote location name
The name of the remote location where the remote file exists. This remote location name provides the data link to the server system (remote location) by using APPN/APPC, over which a DDM conversation is established when this DDM file is accessed.

Device
The name of the device on the client system used to communicate with the remote location.

Local location name
The name of the local location. This is the name by which the server system knows your system. Your system can consist of more than one local location.

Mode
The name of the mode to be used to communicate between the local location and remote location.

Remote network ID
The remote network ID to be used with the remote location. This value further qualifies the remote location name. Two locations with the same remote location name but different remote network IDs are viewed as two distinctly separate locations.

Type
The type of connection to be used to communicate with the remote location when the DDM conversation is established with the remote system. To create a DDM file that uses an SNA connection, specify *SNA. This is the default type.

DDM file creation using TCP/IP
You can create a DDM file that uses TCP/IP as the communication protocol for connecting with the remote system.

Each DDM file that uses TCP/IP contains the following information.

DDM file value and description of values

DDM file name
The name of the DDM file on the client system that is used to identify a specific remote file.

Remote file name
The actual file name of the remote file; that is, the name by which it is known on the server system.

Remote location name
The name of the remote location where the remote file exists. This remote location name provides the data link to the server system (remote location) by using TCP/IP, over which a DDM conversation is established when this DDM file is accessed.

Type
The type of connection to be used to communicate with the remote location when the DDM conversation is established with the remote system. To create a DDM file that uses TCP/IP, specify *IP.

Related concepts

Managing the TCP/IP server
The DRDA and DDM TCP/IP server does not typically require any changes to your existing system configuration. At some time, you might want to change the way the system manages the server jobs to better meet your needs, to solve a problem, to improve the system performance, or to look at the jobs on the system.

DDM file creation using RDB directory entry information
You can create a DDM file that uses the remote location information from a relational database (RDB) directory entry.

Each DDM file that uses an RDB directory entry contains the following information.

DDM file value and description of values

DDM file name
The name of the DDM file on the client system that is used to identify a specific remote file.

Remote file name
The actual file name of the remote file; that is, the name by which it is known on the server system.

Remote location name
Specify *RDB to indicate that the remote location information is taken from an RDB directory entry.

Relational database
The name of the relational database entry used for the remote location information. The remote location information in the RDB directory entry is used to establish the data link to the server system (remote location), over which a DDM conversation is established when the DDM file is accessed.

You need to specify an RDB directory entry associated with an auxiliary storage pool (ASP) group for the DDM file's remote location information to access that ASP group.

Related concepts

Disk management

Effect of job description on ASP group selection

When the target DDM server is configured to use ASP groups, and the DDM file specifies a relational database name, the relational database entry specified in the DDM file on the client is used to establish the ASP group for the target job.

When using a DDM file that does **not** specify a relational database name, the target job's ASP group is established using the initial ASP group attribute in the job description for the user profile that the target job is running under.

For DRDA, it is recommended to not rely on the initial ASP group attribute (INLASPGRP) on the connecting user's job description to establish the ASP group for an IBM i application server job. The INLASPGRP attribute can be set on Create Job Description (CRTJOBD) and Change Job Description (CHGJOBD) commands; However, the attribute may not be used for DRDA depending on the application requester's (AR) configuration, such as the security mechanism negotiated for the incoming connection. Therefore, it is recommended to configure the AR to specify the relational database name of the ASP group when connecting over DRDA.

Example: Using the basic concepts of DDM in an APPC network

This example application uses DDM to access a remote file. It can be run by a company that has warehouses located in several cities.

The following figure illustrates the relationships among the primary items included in a DDM file.

On a IBM i platform in Chicago, an Open Database File (OPNDBF) command requests that file CUST021 be opened for input. Because the file name was not qualified on the command, the library list for the source job is used to find the file, which is stored in the NYCLIB library.

Because CUST021 is a DDM file, the SDDM on the CHICAGO system is started in the source job when the file is opened. The SDDM uses the remote location and mode names (NEWYORK and MODENYC) from the DDM file to establish a DDM conversation with and start a target job (TDDM) on the appropriate server system (NEWYORK). The remote file to be accessed by the client system program is CUSTMAST in library XYZ.

The TDDM receives the remote file name from the SDDM and then allocates and opens the file named CUSTMAST, which corresponds to the DDM file named CUST021 on the client system.

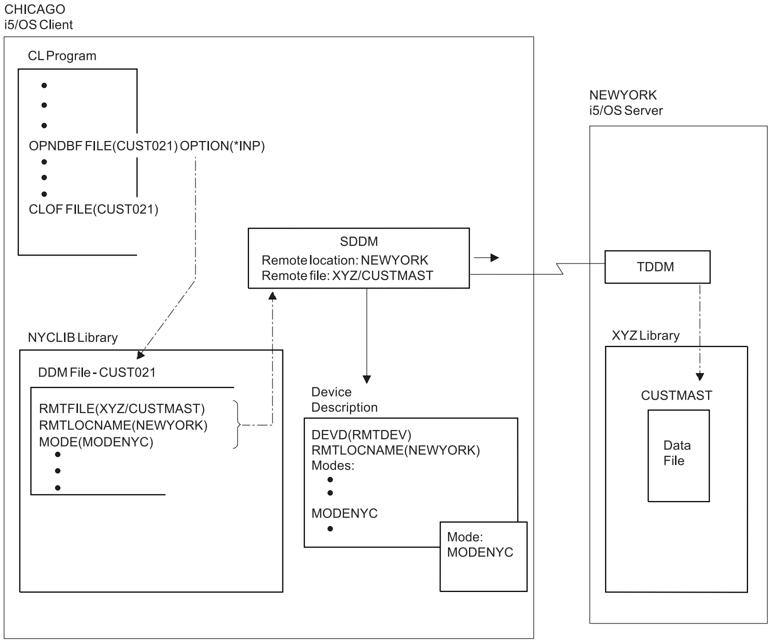


Figure 11. Relationships among DDM file parameters and the systems

The remote location name in the DDM file identifies the remote system where the file exists. The local system uses the remote location name as well as other values specified in the DDM file to select a device description. The device description can be either manually created or, if APPN is being used, automatically created and activated by the system. The SDDM establishes a DDM conversation with the server system using the values NEWYORK and MODENYC in the APPC remote location name. The APPC-related support must have been started on the server system before the request is issued by the SDDM. (No special support is required on the client system.)

Note: The APPN parameter on the Create Controller Description (APPC) (CRTCTLAPPC) and Create Controller Description (SNA Host) (CRTCTLHOST) commands determines whether the APPN support is used.

Example: Using the basic concepts of DDM in an APPN network
The IBM i Advanced Peer-to-Peer Networking (APPN) support can be used to allow DDM access to systems not directly connected to the local system.

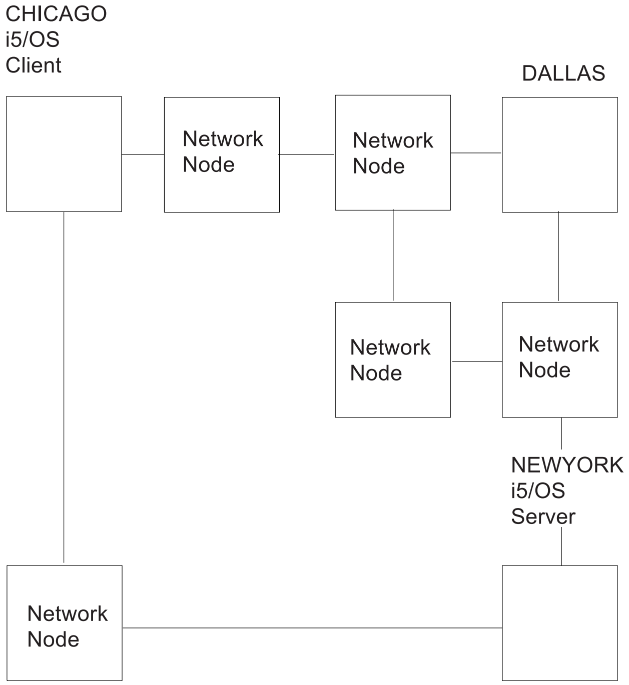


Figure 12. Using DDM in an APPN network

Figure 1 in "Example: Using the basic concepts of DDM in an APPC network" on page 23 shows a program on the Chicago system accessing a file on the New York system. Although the systems are shown as directly connected, the same DDM concepts apply if the network is configured as shown in the preceding figure. When the DDM file CUST021 in the figure is opened on the Chicago system, the APPN support finds the remote location named NEWYORK, determines the optimal path through the network, and establishes a DDM conversation with that location. Although there might be several other systems (network nodes) forwarding the data between CHICAGO and NEWYORK, the source DDM and target DDM function as if there were a direct connection between these two systems.

If the file CUSTMAST were moved from NEWYORK to some other system in the network (for example, DALLAS), then in this example, the DDM file at CHICAGO needs to be changed. The remote location name would be changed from NEWYORK to DALLAS. If a large number of systems in the network refer to the file CUSTMAST, then movement of the file results in a change to the DDM file at each of these systems. By using the IBM i capability to have multiple local location names, maintenance of these files is reduced.

In the preceding figure, the system NEWYORK can be given two local location names, NEWYORK and FILELOC. The DDM file at CHICAGO uses FILELOC as the remote location name. When access to file CUSTMAST is required, APPN finds the location FILELOC in the system named NEWYORK, and the DDM conversation is established as before.

If the file CUSTMAST is now moved from NEWYORK to DALLAS, the user at NEWYORK deletes the local location FILELOC from his system, and it is added to the system at DALLAS. This is done by using the APPN local location list. When the program in CHICAGO now attempts to access the file CUSTMAST, the APPN support finds the remote location FILELOC at the system in Dallas, and the DDM conversation is established to that system. The movement of CUSTMAST did not result in a change to the DDM file at CHICAGO.

This example shows the concept of multiple local locations and how reduced maintenance results when files are moved from one system to another. The example is not intended to suggest that a unique location name should be used for every file accessed through DDM. The decision of which files should be associated with separate local locations should be based on such factors as the movement of these files and the number of remote systems accessing these files.

Additional DDM concepts

Most users of DDM will not need the information in the remainder of these topics; it is intended primarily for experienced programmers who need to know more about DDM.

Described are conceptual details and examples about:

- Program start requests, which start the TDDMs (target jobs)
- Open data paths (ODPs), used to access the files
- Remote location information
- DDM conversations, established for source and target communications
- Source and target jobs
- I/O operations within a job

IBM i as the client system for DDM

All of these actions, as well as those required on the server system, must complete successfully before any operations (file or nonfile) requested by the source program can be done.

When the DDM file is referred to, the following things occur:

- If the request is to open a file, its information is used simultaneously to create an open data path (ODP) on the client system and to start the SDDM support, which runs within the same job as the source program. The SDDM also uses the information: to convert the client system request into a DDM request, to communicate with the appropriate server system, and to establish a DDM conversation to be used for the source job. (The ODP is partially created with the DDM file information; it is not usable until the SDDM processes the remaining information after the DDM conversation is established.)
- The communications portion of DDM establishes a communications path with the server system. The server *system* is identified by using the remote location information specified in the DDM file, and the target *file* is identified by the remote file name. Other information about the remote location, not kept in the DDM file, is stored by the SDDM. This includes the transaction program name, user ID, activation group number, and scope of the conversation. Using the remote location information, the TDDM is started on the server system and a DDM conversation is established when the remote system receives the program start request. The conversation is established the first time the remote file is accessed, but only if a conversation using the same remote location values for that server system does not already exist for the source job.
- After the DDM conversation is established, the SDDM (which can be used by multiple programs and multiple DDM files in the same source job) sends the DDM architecture command to the TDDM, for file-related requests. This command describes the file operation to be done and contains the name of the remote file (specified in the DDM file) to be accessed. For nonfile-related requests, such as when the **`Submit Remote Command (SBMRMTCMD)`** command is used, the remote file name is not sent to the TDDM; the remote file name is ignored.

The SDDM converts each program request for a file open or input/output operation (received by using the DDM file and ODP) into an equivalent DDM command request and then sends it to the server system.

The following figure shows the basic parts on the client system that are involved in accessing remote files.

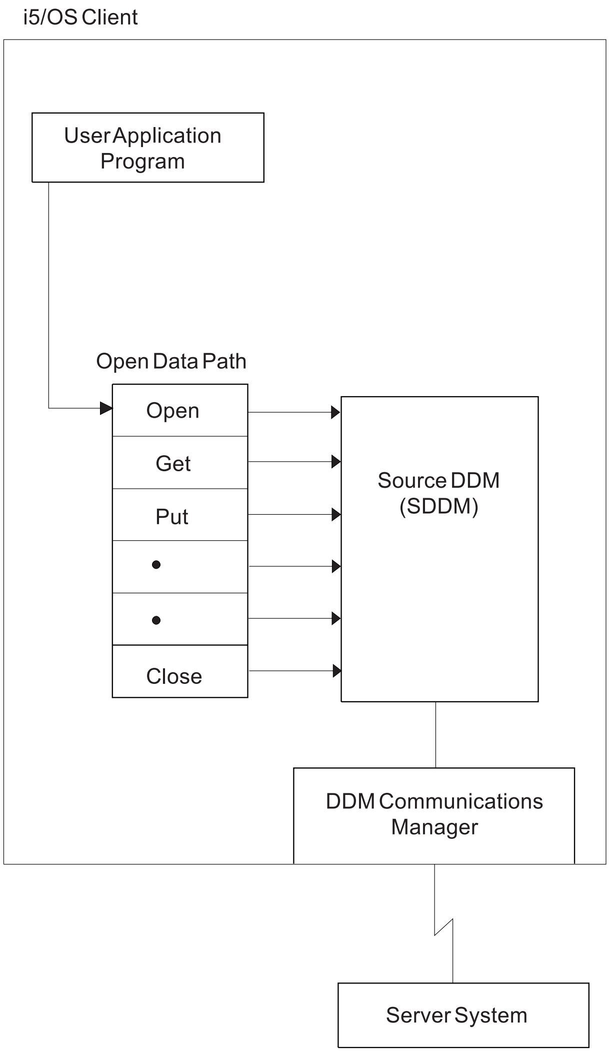


Figure 13. IBM i as the DDM client system

After each request is handled by the target job, the DDM response from the server system is returned, converted by the SDDM into the appropriate form, and passed back to the user. The response might include data (if data was requested) or an indication of status (for other types of file access). The source program waits until the function completes and the results are received.

The following figure shows a simplified example of the interchange of data between the client and server systems for a typical request to access a remote file.

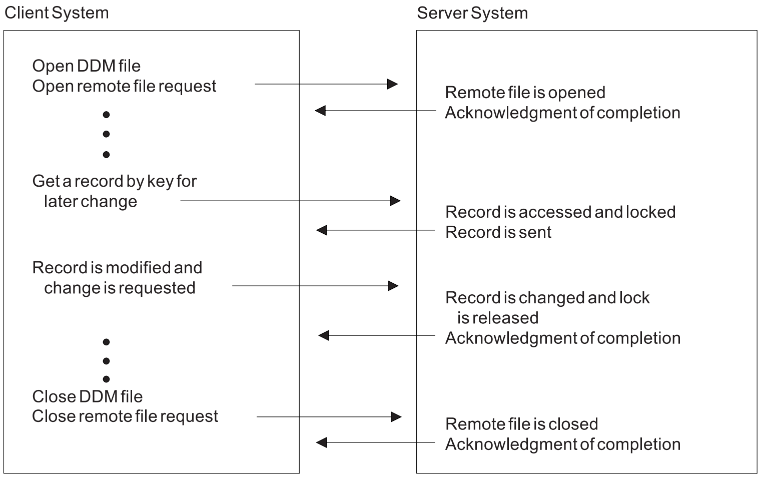

Client System
Server System
Open DDM file
Open remote file request
Remote file is opened
Acknowledgment of completion
Get a record by key for later change
Record is accessed and locked
Record is sent
Record is modified and change is requested
Record is changed and lock is released
Acknowledgment of completion
Close DDM file
Close remote file request
Remote file is closed
Acknowledgment of completion

Figure 14. Typical handling of an I/O operation request

After the first DDM file that was opened in the job is closed, the DDM conversation that it used is normally kept active. This allows the same program or another program in the job to use the same conversation when opening another DDM file, or doing other DDM-related operations. (For example, in Figure 16 on page 32, source job 3A has two DDM files using the same conversation.) This saves the time and resources required to establish a new conversation every time a new DDM file that uses the same remote location information is used in that job.

When a DDM file is closed, the DDM conversation remains active, but nothing happens in the conversation until the SDDM processes the next DDM-related request from a program. While it is not being used, however, the conversation can be dropped. This can occur if the DDMCNV job attribute's default value of *KEEP is changed to *DROP using the **Change Job (CHGJOB)** command, or if the **Reclaim DDM Conversations (RCLDDMCNV)** command or **Reclaim Resources (RCLRSC)** command is used while the job is active.

Related concepts

Parts of DDM: Source DDM
The support on the client (or local) system is started, as needed, within a source job to do DDM functions.

Integrated Language Environment and DDM
Integrated Language Environment® (ILE) introduces the concept of activation groups that run within IBM i jobs. An *activation group* is a substructure of a runtime job.

An activation group consists of system resources (storage for program or procedure variables, commitment definitions, and open files) allocated to one or more programs. An activation group is like a miniature job within a job. By default, all DDM conversations are scoped to the activation group level. To *scope* is to specify the boundary within which system resources can be used. Programs that run in different activation groups start separate DDM conversations when they use the same DDM file or the same remote location information. Sharing of existing DDM conversations takes place within the confines of the activation group. A DDM conversation can be scoped to the job level by specifying OPNSCOPE(*JOB) on the **OPNDBF** command.

Client system actions dependent on type of server system
If the client system is not IBM i or System/38, only the DDM architecture commands defined in Level 2.0 and earlier of the DDM architecture are used.

If the client system is an IBM i or System/38, then IBM i and System/38 extensions to the architecture are used to support some operations not defined by the Level 2.0 DDM architecture. Examples of System/38 and IBM i extensions to the architecture are the **Submit Remote Command (SBMRMTCMD)** and processing file members of remote files.For creating a file when the client is an IBM i and the server is also an IBM i, the IBM i extension is used.

Server systems that are not IBM i or System/38 might not be capable of handling all of the functions that an IBM i or a System/38 can handle. For example, a System/36 does not support relative record processing and keyed record processing with one open operation; therefore, programs that mix accessing records in a file by key or relative record do not work if the file is on a System/36. In addition, server systems that do not support Level 2.0 of the DDM architecture can only handle functions defined in the level they support.

Neither System/36 nor System/38 support access to folder management objects.

Note: The IBM i operating system only allows access to folder management services (FMS) objects when the client supports Level 2.0 of the DDM architecture for *stream files* (files on disks in which data is read and written in consecutive fields without record boundaries) and directories, for example, the IBM Personal Computer using DDM.

An IBM i client does not support access to stream files and directories.

IBM i as the server system for DDM

The IBM i target DDM (or TDDM) is actually a job that runs a DDM-related server system program. It is started when the client system sends a program start request (an SDDM).

For client IBM i systems, the program start request is started on the client system using information contained in the IBM-supplied intersystem communications function (ICF) file for DDM. The remote location information in the DDM file being accessed is used to send the program start request to the appropriate server system.

The attributes of the target job are determined by the values specified on the **Add Communications Entry (ADDCMNE)** command, which is used on the server system to add a communications entry to the subsystem description used for the job. This command identifies the device description, the job description (including the library list for the target job), and the default user profile to be used by the subsystem.

For a IBM i Access Family connection, the routing entry in the QIWS subsystem for DDM (CMPVAL ('DDM')), along with the device description the personal computer is connected to, is used to obtain the attributes of the target job.

After it is started, the TDDM does the following things:

- For database files:
  - Handles communications with the client system by using a DDM conversation established over an APPC, over TCP/IP, or over a IBM i Access Family data link.
  - Converts the access requests from the client system into the equivalent system functions and runs them on the server system. After the target object is located, the server system-created ODP and target database management services are used to access the object for whatever operation is requested. The TDDM can, for example, pass requests that open the object and then do requested I/O operations to the objects.
  - Includes IBM i or System/38 extensions to the DDM Level 2.0 architecture for requests received from the client system (if the client is an IBM i or a System/38), which allow most IBM i functions that operate on local systems to also work on remote IBM i systems. For example, it might receive a **SBMRMTCMD** command from the client system (an IBM i or a System/38) to do a nonfile-related operation, such as using the CL command Replace Library List (RPLLIBL) to replace the library list within the current target job.
  - Converts target IBM i responses to the equivalent DDM responses and sends them back to the client system. When the client system is an IBM i or System/38, the actual IBM i or System/38 messages are sent back to the client system.
- For folder management services objects:

  Converts the DDM stream and directory access requests into the equivalent IBM i folder management services functions and then runs them on the target server. The following commands are supported:

  - **Change Current Directory (CHGCD)**
  - **Change File Attributes (CHGFAT)**

- **Close Directory (CLSDRC)**
- **Close Document (CLOSE)**
- **Copy File (CPYFIL)**
- **Create Directory (CRTDRC)**
- **Create Stream File (CRTSTRF)**
- **Delete Directory (DELDRC)**
- **Delete File (DELFIL)**
- **Force Buffer (FRCBFF)**
- **Get Data Stream (GETSTR)**
- **Get Directory Entry (GETDRCEN)**
- **List File Attributes (LSTFAT)**
- **Load Stream File (LODSTRF)**
- **Lock Data Stream (LCKSTR)**
- **Open Directory (OPNDRC)**
- **Open Document (OPEN)**
- **Put Data Stream (PUTSTR)**
- **Query Current Directory (QRYCD)**
- **Query Space Available (QRYSPC)**
- **Rename Directory (RNMDRC)**
- **Rename File (RNMFIL)**
- **Unload Stream File (ULDSTRF)**
- **Unlock Data Stream (UNLSTR)**

The following figure shows the basic parts on the IBM i client that are involved in processing the requested destination file.

The TDDM runs as a separate batch job, just as any other user APPC, TCP/IP, or IBM i Access Family target application. A new TDDM, using additional server system resources, is started for each distinct source server program start request received by the server system. There is one target job for each DDM conversation. Each TDDM can handle access requests for multiple files in the DDM conversation.

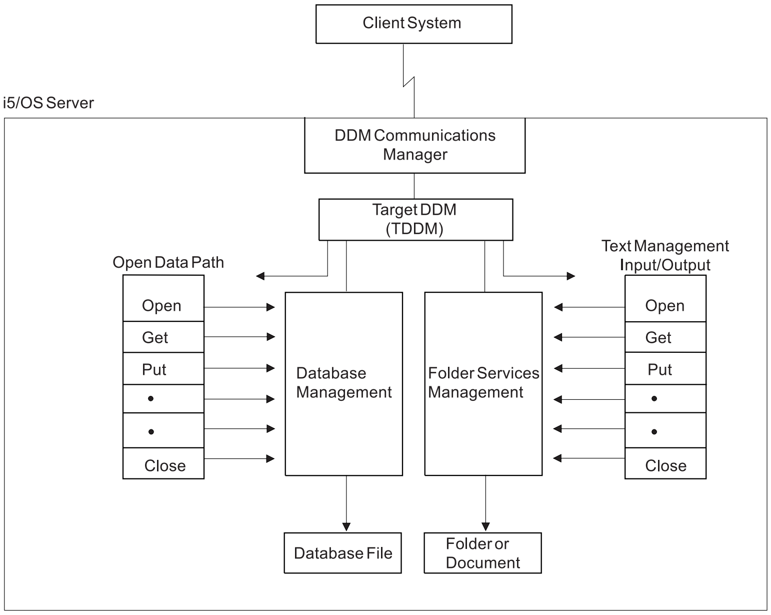


Figure 15. IBM i as the DDM server system

The subsystem, user profiles, and server resources to be used by the TDDM are defined the same as they are for other types of jobs.

Related concepts

Parts of DDM: Target DDM

A server system job is started on the target (or remote) system as a result of an incoming DDM request and ends when the associated DDM conversation ends.

DDM-related jobs and DDM conversations

This topic provides additional information about activation groups, client system jobs, server system jobs, and the DDM conversations used by those jobs.

For remote file processing, at least two separate jobs are used, one running on each system: a client system job and a server system job. (The client system job is the one in which the user application is running.) Multiple application programs can be running in different activation groups within a single client system job. Each activation group within a client system job has a separate DDM conversation and server system job for the remote location information specified in the DDM files. Multiple DDM files share a conversation when the following items are true:

- The files are accessed in the same activation group within a client job.
- The files specify the same remote location combination.

For each DDM conversation, there is one server system job, which includes the TDDM.

The SDDM runs within a client system job or activation group on the client system. It can handle multiple DDM conversations with one or more server systems at the same time. For the same client job or activation group, one SDDM handles all the remote file access requests. This is true regardless of how many server systems or remote files are involved. No separate job for the SDDM exists in the system.

If the client system DDM files involved all use the same remote location information to identify the server system, one TDDM job is created for each client system job that requests access to one or more files on the server system.

The following figure shows five programs accessing six DDM files. The numbers in the upper set of boxes representing DDM files correspond to the same numbers in the lower set of boxes representing the associated remote files. These DDM files are using four different remote location descriptions to access six different remote files, all on the same server system. Seven DDM conversations are needed to handle the processing. An explanation of the DDM conversations follows:

- PGM1 and PGM2 run in different client jobs and are using DDM files (2 and 3) that contain the same remote location information. A separate conversation is needed for each client job.
- PGM3 in client job 3 uses the two DDM files (5 and 6) that both use the same remote location information. They will share the same conversation and server job (5B).
- PGM4 and PGM5 run in different activation groups within client job 4. They are using two DDM files (5 and 6) that both use the same remote location information. A separate conversation is needed for each activation group.

In the following figure, jobs 1, 2, and 3 in System A each have a SDDM. Each activation group in job 4 has its own SDDM. Jobs 1B through 7B each have their own TDDM.

When the application program or the client job closes the DDM file on the client system, the DDM conversation and its associated server job ends, unless the following items are true:

- The value of the DDMCNV attribute of the **Change Job (CHGJOB)** command for the client job is *KEEP (the server default).
- Any locks established during the job by the **Allocate Object (ALCOBJ)** command still exist.

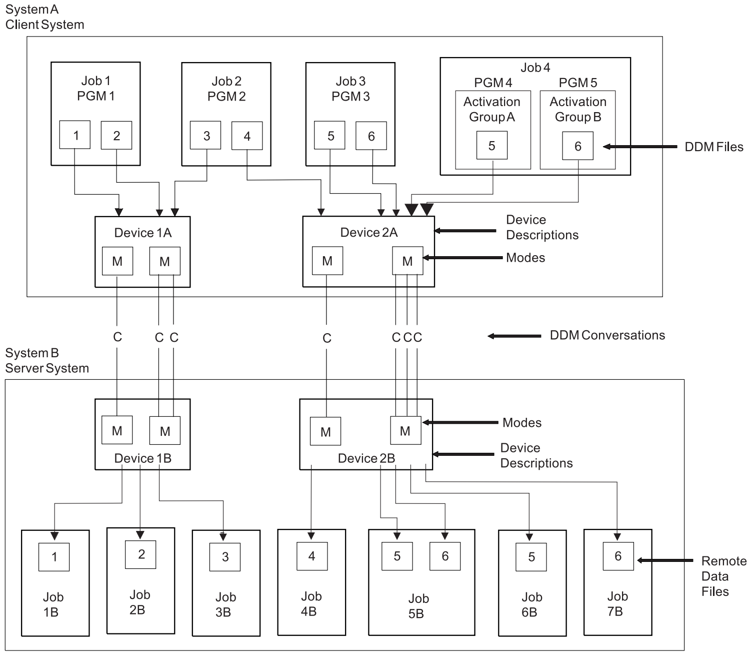


Figure 16. Relationships of DDM client and server jobs

The **CHGJOB** and **ALCOBJ** commands are described in topic Using CL and DDS with DDM. If DDMCNV(*KEEP) is specified, the DDM conversation remains active and waits for another DDM request to be started.

From a performance viewpoint, if the DDM conversation is likely to be used again, *KEEP is the value that should be used. This saves the time and resources used on the server system to start each TDDM and establish the conversation and job.

The following figure shows the relationship between the SDDM and two TDDMs on different server systems and the figure in Example: Accessing files on multiple servers with DDM topic shows the relationship between the SDDM and two TDDMs on *one* server system.

The IBM i operating system can be a client system and a server system at the same time, and two systems can be accessing files located on each other. In addition, an IBM i job can be a client job and a server job. A DDM file can refer to a remote file that is another DDM file.

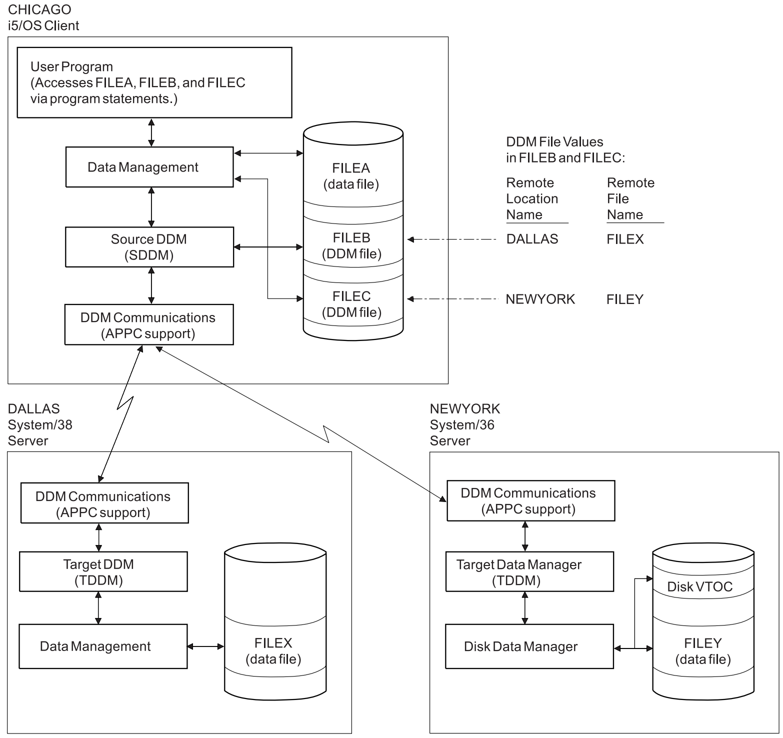


Figure 17. Example: Accessing multiple local and remote files

Examples: Accessing multiple remote files with DDM

These examples show a single application program that use DDM to access multiple remote files. The first example shows the remote files on different server systems, and the second shows them on the same server system.

Example: Accessing files on multiple systems with DDM

This topic contains a figure that shows the relationships among the client system, its DDM files, and two server systems.

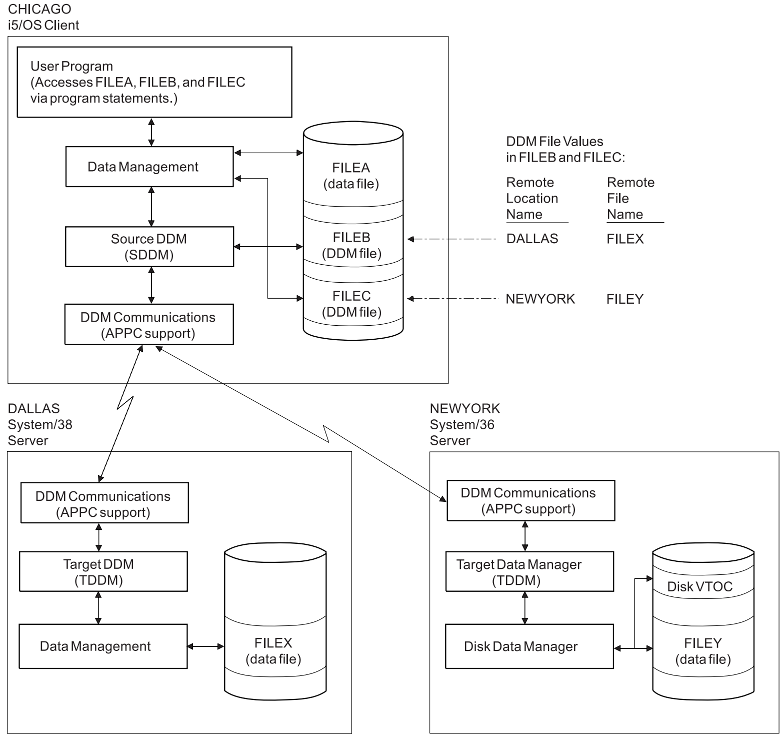


Figure 18. Example: Accessing multiple local and remote files

One server system is a System/38 and the other is a System/36. Each system has DDM installed.

The user program running on the client system is shown accessing three files: FILEA, FILEB, and FILEC. FILEA, located on the client system, is accessed using only local data management. On different server systems, DDM file FILEB corresponds to remote file FILEX and FILEC corresponds to remote file FILEY. When the program opens FILEB and FILEC, DDM allows the program to access the corresponding remote files as if they were on the client system. Only the person who defines the DDM files needs to know where each file is located or what the file name is on the remote system.

Example: Processing multiple requests for remote files with DDM

This example shows how multiple programs access multiple files on the same server system.

This example shows a System/36 server system. The SDDM is shown handling requests for two files from two programs in different jobs, and two TDDMs are handling the requests on the server system (one TDDM for each requesting program). Although program B is accessing two files on the server system, only one TDDM is created if all the associated DDM files specify the same remote location information to identify the server system.

Both programs A and B are sharing FILEA. However, because these programs are shown to be in separate jobs, they *cannot* share the same open data path (ODP) to FILEA. If they were in the same job, programs A and B can share both the ODP on the client system *and* the remote file. When multiple programs within the same job are accessing a remote file at the same time (by using one TDDM for each program), the rules for file sharing are the same for remote files as for local files. These rules are based on how the SHARE parameter is specified on the **`Create DDM File (CRTDDMF)`**, the **`Override with Database File (OVRDBF)`**, and the **`Change DDM File (CHGDDMF)`** commands.

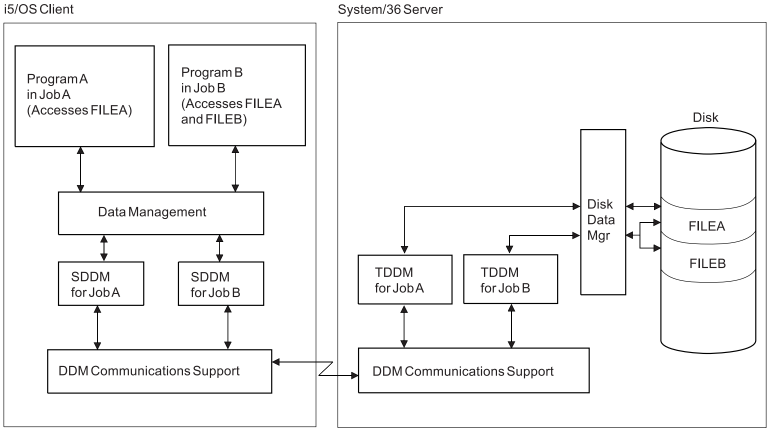

i5/OS Client
System/36 Server
Program A in Job A (Accesses FILEA)
Program B in Job B (Accesses FILEA and FILEB)
Data Management
SDDM for Job A
SDDM for Job B
DDM Communications Support
Disk
Disk Data Mgr
FILEA
FILEB
TDDM for Job A
TDDM for Job B
DDM Communications Support

Figure 19. Example: Processing multiple program and file requests

Planning and design

To prepare for a distributed relational database, you must understand both the needs of the business and relational database technology. To prepare for the use of DDM, you need to meet several requirements including communication requirements, security requirements, and modification requirements.

Related concepts

Performance considerations for DRDA

Distributed relational database performance is affected by the overall design of the database. The location of distributed data, the level of commitment control you use, and the design of your SQL indexes all affect performance.

Planning and design for DRDA

The first requirement for the successful operation of a distributed relational database is thorough planning. You must consider the needs and goals of your enterprise when making the decision to use a distributed relational database.

How you code an application program, where it resides in relation to the data, and the network design that connects application programs to data are all important design considerations.

Database design in a distributed relational database is more critical than when you deal with just one IBM i relational database. With more than one IBM i product to consider, you must develop a consistent management strategy across the network. The following operations require particular attention when forming your strategy:

- General operations
- Networking protocol
- System security
- Accounting
- Problem analysis
- Backup and recovery processes

Identifying your needs and expectations for a distributed relational database

Consider these items when analyzing your needs and expectations of a distributed relational database.

Data needs for distributed relational databases

The first step in your analysis is to determine which factors affect your data and how they affect it.

Ask yourself the following questions:

- What locations are involved?

- What kind of transactions do you envision?
- What data is needed for each transaction?
- What dependencies do items of data have on each other, especially referential limitations? For example, will information in one table need to be checked against the information in another table? (If so, both tables must be kept at the same location.)
- Does the data currently exist? If so, where is it located? Who "owns" it (that is, who is responsible for maintaining the accuracy of the data)?
- What priority do you place on the availability of the needed data? Integrity of the data across locations? Protection of the data from unauthorized access?
- What access patterns do you envision for the data? For instance, will the data be read, updated, or both? How frequently? Will a typical access return a lot of data or a little data?
- What level of performance do you expect from each transaction? What response time is acceptable?

Distributed relational database capabilities

The second step in your analysis is to decide whether your data needs lend themselves to a distributed relational database solution.

Applications where most database processing is done locally and access to remote data is needed only occasionally are typically *good* candidates for a distributed relational database.

Applications with the following requirements are usually *poor* candidates for a distributed relational database:

- The data is kept at a central site *and* most of the work that a remote user needs to do is at the central site.
- Consistently high performance, especially consistently fast response time, is needed. It takes longer to move data across a network.
- Consistently high availability, especially twenty-four hour, seven-day-a-week availability, is needed. Networks involve more systems and more in-between components, such as communications lines and communications controllers, which increases the chance of breakdowns.
- A distributed relational database function that you need is not currently available or announced.

Goals and directions for a distributed relational database

The third step in your analysis is to assess your short-term and long-term goals.

SQL is the standard IBM database language. If your goals and directions include portability or remote data access on unlike systems, you should use distributed relational databases on the IBM i operating system.

The distributed database function of distributed unit of work, as well as the additional data copying function provided by DB2 DataPropagator, broadens the range of activities you can perform on the system. However, if your distributed database application requires a function that is not currently available on IBM i, other options are available until the function is made available on the system. For example, you can do one of the following things:

- Provide the needed function yourself.
- Stage your plans for distributed relational database to allow for the new function to become available.
- Reassess your goals and requirements to see if you can satisfy them with a currently available or announced function. Some alternative solutions are listed in the following table. These alternatives can be used to supplement or replace available function.

Table 3. Alternative solutions to distributed relational database

Solution	Description	Advantages	Disadvantages
Distributed Data Management (DDM)	A function of the operating system that allows an application program or user on one system to use database files stored on a remote system. The system must be connected by a communications network, and the remote system must also use DDM.	• For simple read and update accesses, the performance is better than for SQL. • Existing applications do not need to be rewritten. • Can be used to access S/38, S/36, and CICS®.	• SQL is more efficient for complex functions. • Might not be able to access other distributed relational database platforms. • Does not perform CCSID and numeric data conversions.
Intersystem Communications Function/Common Programming Interface (ICF/CPI Communications)	ICF is a function of the operating system that allows a program to communicate interactively with another program or system. CPI Communications is a call-level interface that provides a consistent application interface for applications that use program-to-program communications. These interfaces make use of SNA's logical unit (LU) 6.2 architecture to establish a conversation with a program on a remote system, to send and receive data, to exchange control information, to end a conversation, and to notify a partner program of errors.	• Allows you to customize your application to meet your needs. • Can provide better performance.	Compared to distributed relational database and DDM, a more complicated program is needed to support communications and data conversion requirements.
Display station pass-through	A communications function that allows users to sign on to one IBM i environment from another IBM i environment and use that system's programs and data.	• Applications and data on remote systems are accessible from local systems. • Allows for quick access when data is volatile and a large amount of data on one system is needed by users on several systems.	Response time on screen updates is slower than locally attached devices.

A distributed relational database usually evolves from simple to complex as business needs change and new products are made available. Remember to consider this when analyzing your needs and expectations.

Designing the application, network, and data for a distributed relational database

Designing a distributed relational database involves making choices about applications, network considerations, and data considerations.

Tips: Designing distributed relational database applications

Distributed relational database applications have different requirements from applications developed solely for use on a local database.

To properly plan for these differences, design your applications with the following considerations in mind:

- Take advantage of the distributed unit of work (DUW) function where appropriate.
- Code programs using common interfaces.
- Consider dividing a complex application into smaller parts and placing each piece of the application in the location best suited to process it. One good way to distribute processing in an application is to make use of the SQL CALL statement to run a stored procedure at a remote location where the data to be processed resides. The stored procedure is not limited to SQL operations when it runs on a Db2 for i application server; it can use integrated database input/output or perform other types of processing.
- Investigate how the initial database applications will be prepared, tested, and used.
- Take advantage, when possible, of SQL set-processing capabilities. This will minimize communication with the application servers. For example, update multiple rows with one SQL statement whenever you can.
- Be aware that database updates within a unit of work must be done at a single site if the remote unit of work (RUW) connection method is used when the programs are prepared, or if the other nodes in the distributed application do not support DUW.
- Keep in mind that the DUW connection method restricts you from directing a single statement to more than one relational database.
- Performance is affected by the choice of connection management methods. Use of the RUW connection management method might be preferable if you do not have the need to switch back and forth among different remote relational databases. This is because more overhead is associated with the two-phase commit protocols used with DUW connection management.

  However, if you have to switch frequently among multiple remote database management systems, use DUW connection management. When running with DUW connection management, communication conversations to one database management system do not have to be ended when you switch the connection to another database management system. In the like environment, this is not as big a factor as in the unlike environment, since conversations in the like environment can be kept active by use of the default DDMCNV(*KEEP) job definition attribute. Even in the like environment, however, a performance advantage can be gained by using DUW to avoid the cost of closing cursors and sending the communication flow to establish a new connection.
- The connection management method determines the semantics of the CONNECT statement. With the RUW connection management method, the CONNECT statement ends any existing connections before establishing a new connection to the relational database. With the DUW connection management method, the CONNECT statement does not end existing connections.

Network considerations for a distributed relational database

The design of a network directly affects the performance of a distributed relational database.

To properly design a distributed relational database that works well with a particular network, do the following things:

- Because the line speed can be very important to application performance, provide sufficient capacity at the appropriate places in the network to achieve efficient performance to the main distributed relational database applications.
- Evaluate the available communication hardware and software and, if necessary, your ability to upgrade.
- For Advanced Program-to-Program Communication (APPC) connections, consider the session limits and conversation limits specified when the network is defined.
- Identify the hardware, software, and communication equipment needed (for both test and production environments), and the best configuration of the equipment for a distributed relational database network.
- Consider the skills that are necessary to support TCP/IP as opposed to those that are necessary to support APPC.
- Take into consideration the initial service level agreements with end user groups (such as what response time to expect for a given distributed relational database application), and strategies for monitoring and tuning the actual service provided.

- Understand that you cannot use an APPC-protected DUW conversation to connect to a database from an application requester (AR) which has been set to an auxiliary storage pool (ASP) group for the current thread.
- Develop a naming strategy for database objects in the distributed relational database and for each location in the distributed relational database. A *location* is a specific relational database management system in an interconnected network of relational database management systems that participate in distributed relational database. A location in this sense can also be a user database in a system configured with independent ASP groups. Consider the following items when developing this strategy:
  - The fully qualified name of an object in a distributed database has three (rather than two) parts, and the highest-level qualifier identifies the location of the object.
  - Each location in a distributed relational database should be given a unique identification; each object in the database should also have a unique identification. Duplicate identifications can cause serious problems. For example, duplicate locations and object names might cause an application to connect to an unintended remote database, and once connected, access an unintended object. Pay particular attention to naming when networks are coupled.
  - Each location in a user database should also be given a unique identification. If a user database on two different systems were to be named PAYROLL, there would be a naming conflict if an application needed to access them both from the same system. When an independent ASP device is configured, the user has an option to specify an RDB name for that device that is different from the name of the ASP device itself. It is the RDB name associated with the primary device in an ASP group by which that user database is known.

Related concepts

Communications Management PDF

Data considerations for a distributed relational database

The placement of data in respect to the applications that need it is an important consideration when designing a distributed relational database.

When making such placement decisions, consider the following items:

- The level of performance needed from the applications
- Requirements for the security, currency, consistency, and availability of the data across locations
- The amount of data needed and the predicted patterns of data access
- If the distributed relational database functions needed are available
- The skills needed to support the system and the skills that are actually available
- Who "owns" the data (that is, who is responsible for maintaining the accuracy of the data)
- Management strategy for cross-system security, accounting, monitoring and tuning, problem handling, data backup and recovery, and change control
- Distributed database design decisions, such as where to locate data in the network and whether to maintain single or multiple copies of the data

Developing a management strategy for a distributed relational database

When you are managing a distributed relational database, keep these strategies in mind.

General operations for a distributed relational database

To plan for the general operation of a distributed relational database, consider both performance and availability.

The following design considerations can help you improve both the performance and availability of a distributed relational database:

- If an application involves transactions that run frequently or that send or receive a lot of data, you should try to keep it in the same location as the data.
- For data that needs to be shared by applications in different locations, put the data in the location with the most activity.

- If the applications in one location need the data as much as the applications in another location, consider keeping copies of the data at both locations. When keeping copies at multiple locations, ask yourself the following questions about your management strategy:
  - Will users be allowed to make updates to the copies?
  - How and when will the copies be refreshed with current data?
  - Will all copies have to be backed up or will backing up one copy be sufficient?
  - How will general administration activities be performed consistently for all copies?
  - When is it permissible to delete one of the copies?
- Consider whether the distributed databases will be administered from a central location or from each database location.

You can also improve performance by doing the following things:

- If data and applications must be kept at different locations, do the following things to keep the performance within acceptable limits:
  - Keep data traffic across the network as low as possible by only retrieving the data columns that will be used by the application; that is, avoid using * in place of a list of column names as part of a SELECT statement.
  - Discourage programmers from coding statements that send large amounts of data to or receive large amounts of data from a remote location; that is, encourage the use of the WHERE clause of the SELECT statement to limit the number of rows of data.
  - Use referential integrity, triggers, and stored procedures (an SQL CALL statement after a CONNECT to a remote relational database management system); this improves performance by distributing processing to the application server (AS), which can substantially reduce line traffic.
  - Use read-only queries where appropriate by specifying the FOR FETCH ONLY clause.
  - Be aware of rules for blocking of queries. For example, in queries between IBM i operating systems, blocking of read-only data is done only for COMMIT(*NONE), or for COMMIT(*CHG) and COMMIT(*CS) when ALWBLK(*ALLREAD) is specified.
  - Keep the number of accesses to remote data low by using local data in place of remote data whenever possible.
  - Use SQL set operations to process multiple rows at the application requester with a single SQL request.
  - Try to avoid dropping of connections by using DDMCNV(*KEEP) when running with remote unit of work (RUW) connection management, or by running with distributed unit of work (DUW) connection management.
- Provide sufficient network capacity by doing the following things:
  - Increase the capacity of the network by installing high-speed, high-bandwidth lines or by adding lines at appropriate points in the network.
  - Reduce the contention or improve the contention balance on certain processors. For example, move existing applications from a host system to a departmental system, or group some distributed relational database work into batch.
- Encourage good table design. At the distributed relational database locations, encourage appropriate use of primary keys, table indexes, and normalization techniques.
- Ensure data types of host variables used in WHERE clauses are consistent with the data types of the associated key column data types. For example, a floating-point host variable has been known to disqualify the use of an index built over a column of a different data type.

You can also improve availability by doing the following things:

- In general, try to limit the amount of data traffic across the network.
- If data and applications must be kept at different locations, do the following things to keep the availability within acceptable limits:

  - Establish alternate network routes.
  - Consider the effect of time zone differences on availability:
    - Will qualified people be available to start the system?
    - Will off-hours batch work interfere with processing?
  - Ensure good backup and recovery features.
  - Ensure people are skilled in backup and recovery.

Security considerations for a distributed relational database

Part of planning for a distributed relational database involves the decisions you must make about securing distributed data.

These decisions include:

- What systems should be made accessible to users in other locations and which users in other locations should have access to those systems.
- How tightly controlled access to those systems should be. For example, should a user password be required when a conversation is started by a remote user?
- Is it required that passwords flow over the wire in encrypted form?
- Is it required that a user profile under which a client job runs be mapped to a different user identification or password based on the name of the relational database to which you are connecting?
- What data should be made accessible to users in other locations and which users in other locations should have access to that data.
- What actions those users should be allowed to take on the data.
- Whether authorization to data should be centrally controlled or locally controlled.
- If special precautions should be taken because multiple systems are being linked. For example, should name translation be used?

When making the previous decisions, consider the following items when choosing locations:

- Physical protection. For example, a location might offer a room with restricted access.
- Level of system security. The level of system security often differs between locations. The security level of the distributed database is no greater than the lowest level of security used in the network.

  All systems connected by Advanced Program-to-Program Communication (APPC) can do the following things:

  - If both systems are IBM i products, communicate passwords in encrypted form.
  - When one system receives a request to communicate with another system in the network, verify that the requesting system is actually "who it says it is" and that it is authorized to communicate with the receiving system.

  All systems can do the following things:

  - Pass a user's identification and password from the local system to the remote system for verification before any remote data access is allowed.
  - Grant and revoke privileges to access and manipulate SQL objects such as tables and views.

  The IBM i operating system includes security audit functions that you can use to track unauthorized attempts to access data, as well as to track other events pertinent to security. The system also provides a function that can prevent all distributed database access from remote systems.

  - Security-related costs. When considering the cost of security, consider both the cost of buying security-related products and the price of your information staff's time to perform the following activities:
    - Maintain identification of remote-data-accessing users at both local and remote systems.
    - Coordinate auditing functions between sites.

Related concepts

Security
The IBM i operating system has built in security elements that limit access to data resources of a server. Security options range from simple physical security to full password security coupled with authorization to commands and data objects.

Accounting for a distributed relational database

You need to be able to account and charge for the use of distributed data.

Consider the following items:

- Accounting for the use of distributed data involves the use of resources in one or more remote systems, the use of resources on the local system, and the use of network resources that connect the systems.
- Accounting information is accumulated by each system independently. Network accounting information is accumulated independent of the data accumulated by the systems.
- The time zones of various systems might have to be taken into account when trying to correlate accounting information. Each system clock might not be synchronized with the remote system clock.
- Differences might exist between each system's permitted accounting codes (numbers). For example, the IBM i operating system restricts accounting codes to a maximum of 15 characters.

The following functions are available to account for the use of distributed data:

- IBM i job accounting journal. The system writes job accounting information into the job accounting journal for each distributed relational database application. The **`Display Journal (DSPJRN)`** command can be used to write the accumulated journal entries into a database file. Then, either a user-written program or query functions can be used to analyze the accounting data.
- NetView® accounting data. The NetView licensed program can be used to record accounting data about the use of network resources.

Related reference

Display Journal (DSPJRN) command

Job accounting in a distributed relational database
The IBM i job accounting function gathers data so you can determine who is using the system and what system resources they are using.

Problem analysis for a distributed relational database

You need to manage problem analysis in a distributed database environment. Problem analysis involves both identifying and resolving problems for applications that are processed across a network of systems.

Consider the following items:

- Distributed database processing problems manifest themselves in various ways. For example, an error return code might be passed to a distributed database application by the system that detects the problem. In addition, responses might be slow, wrong, or nonexistent.
- Tools are available to diagnose distributed database processing problems. For example, each distributed relational database product provides trace functions that can help diagnose distributed data processing problems.
- When the IBM i operating system detects system failures, it logs information about program status immediately after the failure is detected.

Backup and recovery for a distributed relational database

In a single system environment, backup and recovery take place locally. But in a distributed database, backup and recovery also affect remote locations.

The IBM i operating system allows individual tables, collections, or groups of collections to be backed up and recovered. Although backup and recovery can only be done locally, you might want to have less critical data on a system that does not have adequate backup support. Backup and recovery procedures must be consistent with data that might exist on more than one application server. Because you have more than one system in the network, you might want to save such data to a second system so that it is always available to the network in some form. Strategies such as these need to be planned and laid out specifically before a database is distributed across the network.

Planning and design for DDM

There are several requirements that must be met for distributed data management (DDM) to be used properly.

Notes:

- Before determining which files should be accessed using DDM, review Performance considerations for DDM.
- Programming requirements and considerations for control language (CL) commands and data description specifications (DDS) are covered in Using CL and DDS with DDM and Operating considerations for DDM.

Related concepts

DDM overview
This topic describes the purpose of distributed data management (DDM), the functions that DDM supplies, and the concepts of IBM i DDM.

Performance considerations for DDM
These topics provide information to help you improve performance when using DDM and also provide some information about when to use something other than DDM to accomplish some functions.

Operating considerations for DDM
This topic collection tells how the IBM i operating system, both as a client or server system, communicates with another IBM i to perform remote file processing. It also describes the differences when an IBM i is communicating with a system that is not an IBM i.

Using CL and DDS with DDM
This topic contains DDM-related information about specific control language (CL) commands, data description specifications (DDS) considerations, DDS keywords, and DDM user profile authority.

Communications requirements for DDM in an APPC network

Each IBM i product in a distributed data management (DDM) network that is not using OptiConnect must meet these communications requirements.

- APPC/APPN support or the IBM i Access Family licensed program must be installed and configured on the system.
- At least one Systems Network Architecture (SNA) communications line must use synchronous data link communications (SDLC), token-ring network, Ethernet, or X.25 protocol.

The number of sessions that can be used for DDM conversations is not limited by DDM. The maximum is determined in the same manner as for any other APPC-related communications. For parallel sessions, the session maximum is specified in the mode. For single session devices, the session maximum is always one.

IBM i products in a DDM network that uses OptiConnect must have the OptiConnect software and hardware installed. OptiConnect replaces the need for SNA communications line connections.

Configuring a communications network in a TCP/IP network

This topic provides a high-level overview of the steps you take to set up a TCP/IP network.

1. Identify your IBM i platform to the local network (the network that your system is directly connected to).
   a) Determine if a line description already exists.
   b) If a line description does not already exist, create one.
   c) Define a TCP/IP interface to give your system an IP address.
2. Define a TCP/IP route. This allows your system to communicate with systems on remote TCP/IP networks (networks that your IBM i platform is not directly connected to).

3. Identify the names of the servers in your network.
    a) Build a local host table.
    b) Identify a remote name server.
4. Start TCP/IP.
5. Verify that TCP/IP works.

Security requirements for DDM

You can prevent intentional and unintentional access to the data resources of a system by the distributed data management (DDM) user.

Access to data in the DDM environment can be limited or prevented altogether, by a system-level network attribute, the DDMACC parameter on the Change Network Attributes (CHGNETA) command. This attribute allows the system (as a server system) to prevent all remote access. This attribute can also allow the system to control file access by using standard authority to files and, further, by using an optional user exit program to restrict the types of operations allowed on the files for particular users.

To provide adequate security, you might need to set up additional user profiles on the server system, one for each client system user who can have access to one or more server system files. Or, a default user profile should be provided for multiple client system users. The default user profile is determined by the communications entry used in the subserver in which the target jobs are run.

For user profiles (or their equivalent) on server systems that are not running IBM i, refer to that system's documentation.

File requirements for DDM

Before remote files can be accessed by a IBM i product, distributed data management (DDM) files must be created on the client system.

At the time a DDM file is used, the device (remote location name) and mode (APPC session characteristics) specified in the DDM file must also exist on the system if APPN is not used. If APPN is used, then the device does not need to exist on the system. However, the system identified by the remote location name must exist within the APPN network. The APPN parameter on the **`Create Controller Description (APPC) (CRTCTLAPPC)`** and the **`Create Controller Description (SNA Host) (CRTCTLHOST)`** commands controls whether APPN is used.

Program modification requirements for DDM

Remote files can be accessed by IBM i application programs written in the high-level language (HLL) and control language.

In most cases, these applications can access both local or remote files without the programs being changed. However, some considerations and restrictions might require the programs to be changed and recompiled. These are grouped in these categories:

- IBM i functions that are not supported by the DDM architecture, but for which a System/38 extension to the architecture might exist. These functions can be used only when the client and server systems are running System/38 or IBM i.
- Restrictions and considerations that apply when the *client* or *server* system is running IBM i.
- Restrictions and considerations that apply to all target systems. User programs accessing local files should program for abnormal conditions such as *No record found*, *End of file*, and *Record lock time-out on read for update*. These conditions can also occur when a remote file is being accessed using DDM. In addition, the use of DDM exposes the program to communication line failures while sending disk I/O operations.

  When a communications failure occurs, the system sends an appropriate message to the job, which is returned to the application program as a generic file error. Each high-level language provides unique user syntax capabilities for user-controlled handling or default processing of exceptional results of a disk operation. Some languages might permit the user to retrieve the job message identification (ID) that would specifically indicate a DDM communications failure. Refer to the appropriate language manual for specific capabilities.

For secondary SDLC lines, it is recommended that the INACTTMR parameter of the Create Line Description (SDLC) (CRTLINSDLC) command be set on the client and server systems to detect the stopping of polling by the primary system. This prevents the possibility of a DDM read-for-update record lock lasting indefinitely due to a communications failure on the primary system.

DDM architecture-related restrictions

The items listed in this topic are DDM architecture-related restrictions. Therefore, application programs that use these items might have to be changed and recompiled before they can access remote files.

- The DDM architecture does not support IBM i logical files that are in multiple formats. However, because multiple-format logical files are supported as a System/38 extension to the DDM architecture, they can be used with DDM, but only if the source and server systems are running IBM i or System/38.
- Externally described data (using data description specifications (DDS) on IBM i) is not supported by the DDM architecture. However, DDS can still be used, especially if both systems are running IBM i or System/38. If the server system is running IBM i or System/38, most of the DDS support can be used as though the remote file is a local file.
- To access folder management services objects, the client system must support Level 2.0 or Level 3.0 of the DDM architecture for stream files and the stream access method. The following restrictions for the byte stream model apply:
  - WAIT time is not supported by the folder management services on the **`Lock Data Stream (LCKSTR)`** command. The user must handle the waiting function on the client system.
  - The **`Copy File (CPYFIL)`** command used to copy a document on the IBM i operating system is supported with the restrictions. Only the header information is copied; no data is copied.
  - The DELDRCOP (DRCALL) parameter is not supported on the **`Delete Directory (DELDRC)`** command.
- Personal computer generic names are not allowed when performing operations on data management objects such as files, libraries, or members. However, generic names are allowed when performing operations on folder management services objects such as documents and folders. Generic names are supported where the personal computer supports the operation and in the manner that the personal computer supports the operation. For example, generic names are not supported for folders using the rename and delete commands because the personal computer does not support them.

IBM i client and server restrictions and considerations for DDM

When the client system is running IBM i, IBM i database functions can be used on remote files. However, there are some restrictions.

The restrictions are:

- An IBM i client system can create files on a system running System/38, but the DDM architecture file models are used. As a result, no multiple-format logical or join logical files can be created on a server system that is not running IBM i, including a System/38.
- Save or restore operations do not save or restore the data on a server system. Only the DDM file object can be saved or restored locally.
- Operations that delay for a time period (that is, that wait for a file or record) are determined by the time values specified on the server system. These values are specified by the WAITFILE and WAITRCD parameters on various CL commands. This can result in increased delay times when DDM is used to access files or records remotely.
- Query requests (OPNQRYF) to a System/38 cannot use group selection and join processing.
- When running System/36 applications to or from IBM i, these applications might result in timeouts while waiting for a resource to become available. When running System/36 applications to or from another System/36, the application waits indefinitely for the resource to become available.

  For both source and target DDM jobs, due to the way DDM sends APPC operations, it is possible for the DDM job on the secondary side of the APPC conversation to wait indefinitely after a line failure or other failures at the remote system.

  Consider the following suggestions to avoid indefinite waits:

  - If the remote system supports record lock timeouts, ensure that reasonable time values are specified. For example, on a target IBM i or System/38 database file, do not use the maximum value that is allowed or *NOMAX when you specify the WAITRCD parameter on the CRTPF command.

    WAITRCD addresses read-for-update operations, but does not apply to other file operations, such as read only, add, and so on.
  - When using an SDLC secondary line, use a time value for the line inactivity timer (INACTTMR). Do not use the *NOMAX value.
  - Provide the person responsible for system operation with the associated line, controller, and device names (or a list of DDM jobs that might run). If a DDM job then appears to be waiting indefinitely, this person can display the job information to determine if the job is waiting indefinitely by reviewing the job's processing unit time use (by using the Display Job (DSPJOB) command to display the active run attributes).

When the *server* system is running IBM i, IBM i database functions can be used to access remote files, with the following restrictions:

- The physical files that the logical files or join logical files are based on must exist on the same IBM i operating system.
- A logical file on an IBM i client system cannot share the access path of a remote file (on any server system).
- Query requests (OPNQRYF), which require group selection and join processing from a System/38, do not work.

Non-IBM i target restrictions and considerations for DDM

In addition to the restrictions that apply when the server system is running IBM i, the restrictions in this topic also might apply when the server system is not running IBM i or System/38.

Whether they apply depends on what the target system supports. You should refer to that system's documentation for more information.

- Only field data types that are common to the source and server systems can normally be processed by HLL applications. Floating-point data is an example of a data type that might not be common. Records can be transmitted that contain floating-point data, but the representation of floating-point data sent between systems might differ.

  The packed signs sent between systems might differ; for example, one system might use a C and another system might use an F.

  Note: It is possible for you to write your application program so that it interprets the byte string for a record processed through a DDM file in any way that you want. However, whenever you do this, it is your responsibility to ensure that the data is handled correctly.
- Any operations that request a delay period before returning, such as for record lock wait times, might be rejected or changed to a zero wait time by the server system.
- Lock requests can be changed by the server system to a more restrictive lock. This might prevent some operations from occurring at the same time that can otherwise be performed on the local IBM i operating system. See "Allocate Object (ALCOBJ) command" on page 155 for more information.
- Some IBM i parameters are ignored or cause errors if they are used during remote file processing on server systems that do not run IBM i. Examples are the FRCRATIO and FMTSLR parameters on some of the file commands. For more information, see OVRDBF (Override with Database File) command and see Copy commands with DDM.
- Member names are not supported in the DDM architecture. When the server system is not running IBM i or System/38, CL commands that have a MBR parameter, such as the **`Clear Physical File Member (CLRPFM)`** command, must be changed if the parameter specifies a member name that is different than the file name. If the member name is different, an error occurs if the command is used for a remote file that does not reside on the IBM i operating system. For some commands, MBR(*FIRST) or MBR(*LAST) is also valid. See Member-related commands with DDM for a list of all the CL commands related to file members, and for those that are not valid for accessing files on server systems that do not run IBM i.

Note: MBR(*LAST) is not supported by System/38.

- If a parameter on a CL command requires the name of a source file, then the names of the DDM files that refer to target files that are not on an IBM i operating system cannot be specified. The IBM i operating system cannot determine whether a remote file on a target that does not reside on IBM i is in fact a source file. (See Source file commands for a list of all the CL commands related to source files.)
- Certain IBM i commands that are valid for IBM i or System/38 server systems are not valid for other targets. See DDM-related CL command lists for the lists of commands that are not supported when the target is not an IBM i or a System/38.

Initial setup

The IBM i operating system provides runtime support for distributed relational databases. However, some setup work might be required to make the servers and clients ready to send and receive work, particularly in the Advanced Program-to-Program Communication (APPC) environment.

One or more subsystems can be used to control interactive, batch, spooled, and communications jobs. All the clients in the network must have their relational database directory set up with connection information. Finally, you might want to put data into the tables of the server throughout the network.

The relational database directory contains database names and values that are translated into communications network parameters. A client must have an entry for each database in the network, including the local database and any user databases that are configured on independent auxiliary storage pools (independent ASPs, also known as independent disk pools). These local entries can be added automatically by the system, or manually. Each directory entry consists of a unique relational database name and corresponding communications path information. Information about the preferred password security for outbound connections can be specified. For access provided by ARD programs, the ARD program name must be added to the relational database directory entry.

There are a number of ways to enter data into a database. You can use an SQL application program, some other high-level language application program, or one of these methods:

- Interactive SQL
- IBM i query management
- Data file utility (DFU)
- Copy File (CPYF) command

Connection and setup information for a distributed relational database network of unlike systems can be found in the *Distributed Relational Database Cross-Platform Connectivity* book, SG24-4311.

Related concepts

Independent auxiliary storage pool

Independent disk pools

Related reference

Copy File (CPYF) command

IBM i work management

All of the work on the IBM i operating system is submitted through the work management function. On the system, you can design specialized operating environments to handle different types of work to satisfy your system requirements.

However, when the operating system is installed, it includes a work management environment that supports interactive and batch processing, communications, and spool processing.

On the system, all user jobs operate in an environment called a *subsystem*, defined by a subsystem description, where the system coordinates processing and resources. Users can control a group of jobs with common characteristics independently of other jobs if the jobs are placed in the same subsystem. You can start and end subsystems as needed to support the work being done and to maintain the performance characteristics you want.

The basic types of jobs that run on the system are interactive, communications, batch, spooled, autostart, and prestart.

An interactive job starts when you sign on a workstation and ends when you sign off. An Advanced Program-to-Program Communication (APPC) batch job is a job started from a program start request from another system. A non-communications batch job is started from a job queue. Job queues are not used when starting a communications batch job. Spooling functions are available for both input and output. Autostart jobs perform repetitive work or one-time initialization work. Autostart jobs are associated with a particular subsystem, and each time the subsystem is started, the autostart jobs associated with it are started. Prestart jobs are jobs that start running before the remote program sends a program start request.

Related concepts

Managing the TCP/IP server
The DRDA and DDM TCP/IP server does not typically require any changes to your existing system configuration. At some time, you might want to change the way the system manages the server jobs to better meet your needs, to solve a problem, to improve the system performance, or to look at the jobs on the system.

Setting up your work management environment

One subsystem, called a *controlling subsystem*, starts automatically when you load the system. Two controlling subsystem configurations are supplied by IBM.

The first configuration includes the following subsystems:

- QBASE, the controlling subsystem, supports interactive, batch, and communications jobs.
- QSPL supports processing of spooling readers and writers.
- QSYSWRK supports various system functions, such as TCP/IP.
- QUSRWRK is the user work subsystem. It contains jobs that are started by systems to do work on behalf of a user.

QBASE automatically starts when the server is started. An automatically started job in QBASE starts QSPL.

The second controlling subsystem configuration supplied is more complex. This configuration includes the following subsystems:

- QCTL, the controlling subsystem, supports interactive jobs started at the console.
- QINTER supports interactive jobs started at other workstations.
- QCMN supports communications jobs.
- QBATCH supports batch jobs.
- QSPL supports processing of spooling readers and writers.
- QSYSWRK supports various system functions, such as TCP/IP.
- QUSRWRK is the user work subsystem. It contains jobs that are started by systems to do work on behalf of a user.

If you change your configuration to use the QCTL controlling subsystem, it starts automatically when the system is started. An automatically started job in QCTL starts the other subsystems.

You can change your subsystem configuration from QBASE to QCTL by changing the system value QCTLSBSD (controlling subsystem) to QCTL on the **`Change System Value (CHGSYSVAL)`** command and starting the system again.

You can change the IBM-supplied subsystem descriptions or any user-created subsystem descriptions by using the **`Change Subsystem Description (CHGSBSD)`** command. You can use this command to change the storage pool size, storage pool activity level, and the maximum number of jobs for the subsystem description of an active subsystem.

Related concepts

Communications Management PDF

Managing work

Related reference

Change Subsystem Description (CHGSBSD) command

Change System Value (CHGSYSVAL) command

APPC subsystems

In a distributed relational database using a Systems Network Architecture (SNA) network, communications jobs and interactive jobs are the main types of work an administrator must plan to manage on each system.

Systems in the network start communications jobs to handle requests from a client. A client's communications requests to other systems normally originate from interactive or batch jobs on the local system.

Setting up an efficient work management environment for the distributed relational database network systems can enhance your overall network performance by allocating system resources to the specific needs of each server and client in the network.

When the IBM i licensed program is first installed, QBASE is the default controlling subsystem. As the controlling subsystem, QBASE allocates system resources between the two subsystems QBASE and QSPL. Interactive jobs, communications jobs, batch jobs, and so on, allocate resources within the QBASE subsystem. Only spooled jobs are managed under a different subsystem, QSPL. This means you have less control of system resources for handling communications jobs versus interactive jobs than you would using the QCTL controlling subsystem.

Using the QCTL subsystem configuration, you have control of four additional subsystems for which the system has allocated storage pools and other system resources. Changing the QCTL subsystems, or creating your own subsystems gives you even more flexibility and control of your processing resources.

Different system requirements for some of the systems in the Spiffy Corporation distributed relational database network might require different work management environments for best network efficiency. The following discussions show how the distributed relational database administrator can plan a work management subsystem to meet the needs of each IBM i product in the Spiffy distributed relational database network.

In the Spiffy Corporation system organization, a small dealership might be satisfied with a QBASE level of control for the various jobs its users have on the system. For example, requests to a small dealership's relational database from the regional client (to update dealer inventory levels for a shipment) are handled as communications jobs. Requests from a dealership user to the regional server, to request a part not currently in stock locally, are handled as interactive jobs on the dealership system. Both activities are relatively small jobs because the dealership is smaller and handles fewer service orders and parts sales. The coordination of resources in the QBASE subsystem provides the level of control this enterprise requires for their interactive and communications needs.

A large dealership, on the other hand, probably manages its work through the QCTL subsystem, because of the different workloads associated with the different types of jobs.

The number of service orders booked each day can be high, requiring a query to the local relational database for parts or to the regional center server for parts not in stock at the dealership. This type of activity starts interactive jobs on their system. The dealership also starts a number of interactive jobs that are not distributed relational database related jobs, such as enterprise personnel record keeping, marketing and sales planning and reporting, and so on. Requests to this dealership from the regional center for performance information or to update inventory or work plans are communications jobs that the dealership wants to manage in a separate environment. The large dealership can also receive a request from another dealership for a part that is out of stock at the regional center.

For a large dealership, the QCTL configuration with separate subsystem management for QINTER and QCMN provides more flexibility and control for managing its work environment. In this example, interactive and communications jobs at the dealership system can be allocated more of the system resources than other types of jobs. Additionally, if communications jobs are typically fewer than interactive jobs for this system, resources can be targeted toward interactive jobs, by changing the subsystem descriptions for both QINTER and QCMN.

A work management environment tailored to a Spiffy Corporation regional center perspective is also important. In the Spiffy network, the regional center is a client to each dealership when it updates the dealership inventory table with periodic parts shipment data, or updates the service plan table with new or updated service plans for specific repair jobs. Some of these jobs can be run as interactive jobs (on the regional system) in early morning or late afternoon when system usage is typically less, or run as batch jobs (on the regional system) after regular business hours. The administrator can tailor the QINTER and QBATCH subsystems to accommodate specific processing times and resource needs.

The regional center is also a server for each dealership when a dealership needs to query the regional relational database for a part not in stock at the dealership, a service plan for a specific service job (such as rebuilding a steering rack), or for technical bulletins or recall notifications since the last update to the dealership relational database. These communications jobs can all be managed in QCMN.

However, a closer examination of some specific aspects of distributed relational database network use by the KC000 (Kansas City) regional center and the dealerships it serves suggests other alternatives to the distributed relational database administrator at Kansas City.

The KC000 system serves several large dealerships that handle hundreds of service orders daily, and a few small dealerships that handle fewer than 20 service orders each day. The remaining medium-sized dealerships each handle about 100 service orders daily. One problem that presents itself to the distributed relational database administrator is how to fairly handle all the communications requests to the KC000 system from other systems. A large dealership might control QCMN resources with its requests so that response times and costs to other systems in the network are unsatisfactory.

The distributed relational database administrator can create additional communications subsystems so each class of dealerships (small, medium, or large) can request support from the server and generally receive better response. By tailoring the subsystem attributes, prestart job entries, communications work entries, and routing entries for each subsystem description, the administrator controls how many jobs can be active on a subsystem and how jobs are processed in the subsystem.

The administrator can add a routing entry to change the class (and therefore the priority) of a DRDA/DDM job by specifying the class that controls the priority of the job and by specifying QCNTEDDM on the CMPVAL parameter, as in the following example:

```
ADDRTGE SBSD(QCMN) SEQNBR(280) CLS(QINTER) CMPVAL('QCNTEDDM' 37)
```

The administrator can also add a prestarted job for DRDA/DDM job by specifying QCNTEDDM as the prestarted job, as in the following example:

```
ADDPJE SBSD(QCMN) PGM(QCNTEDDM)
```

Related concepts

Communications Management PDF

Managing work

TCP/IP subsystems

By default, the TCP/IP server prestart jobs used for TCP/IP connections run in the QUSRWRK subsystem.

QUSRWRK is the user work subsystem. It contains jobs that are started by systems to do work on behalf of a user. The listener job that dispatches work to the prestart jobs runs in QSYSWRK.

User databases on independent auxiliary storage pools

The user can create additional IBM i relational databases by configuring independent auxiliary storage pools (independent ASPs) on the system. Each independent auxiliary storage pool group is a relational database.

In this topic collection, independent auxiliary storage pool groups are called *user databases*. They consist of all the database objects that exist on the independent auxiliary storage pool group disks. Additionally, all database objects in the system database of the IBM i operating system to which the independent auxiliary storage pool is varied on are logically included in a user database. However, from a commitment control perspective, the system database is treated differently.

There are a number of rules associated with the creation and use of user databases, besides those imposed by the commitment control considerations just mentioned. One example is that you cannot use an Advanced Program-to-Program Communication (APPC) protected distributed unit of work (DUW) conversation to connect to a database from a client which has been set to a user database (an auxiliary storage pool [ASP] group) for the current thread. Another example is that the name of any schema created in a user database must not already exist in that user database or in the associated system database. For more information about such restrictions, see the SQL reference topic.

- There are certain DRDA-related objects that cannot be contained in user databases. DDM user exit programs must reside in libraries in the system database, as must any Application Requester Driver programs.
- The process of varying on a user database causes the relational database (RDB) directory to be unavailable for a period of time, which can cause attempts by a DRDA client or server to use the directory to be delayed or to timeout. The exposure to having directory operations timeout is much greater if multiple databases are varied on at the same time. The first time a user database is varied on, an attempt is made by the system to add a directory entry for that database. If the directory is unavailable due to a concurrent vary on operation, the addition fails and the entry must be manually added.
- Other considerations in the use of user databases concern configuration of entries in the RDB directory. One of the rules for naming user databases is that user RDB names cannot match the system name specified in the network attributes (as displayed by the Display Network Attributes (DSPNETA) command).
- Local user database entries in the RDB directory are added automatically the first time that the associated databases are varied on. They are created using the *IP protocol type and with the remote location designated as LOOPBACK. LOOPBACK indicates that the database is on the same system as the directory. It is highly suggested that user databases that are intended to be switched among systems be configured to have a dedicated IP address associated with them. If the switchable database does not have a dedicated IP address, then whenever it is switched, you must manually update its directory entry on all the systems that reference that database.
- Once user databases are varied on, the system must verify RDB names on incoming requests to match either the system database or a user databases. If clients were running against invalid RDB names they must be changed.

Related concepts

Managing application CRG takeover IP addresses

Troubleshooting transactions and commitment control

Using the relational database directory
The IBM i operating system uses the relational database directory to define the relational database names that can be accessed by system applications and to associate these relational database names with their corresponding network parameters. The system also uses the directory to specify if the connection uses Systems Network Architecture (SNA) or IP.

Related reference

Display Network Attributes (DSPNETA) command

SQL reference

Using the relational database directory

The IBM i operating system uses the relational database directory to define the relational database names that can be accessed by system applications and to associate these relational database names with their corresponding network parameters. The system also uses the directory to specify if the connection uses Systems Network Architecture (SNA) or IP.

The relational database directory allows a client to accept a relational database name from the application and translate this name into the appropriate Internet Protocol (IP) address or host name and port, or the appropriate Systems Network Architecture network identifier and logical unit (LU) name values for communications processing. As of V5R2, the RDB directory is also used to specify the user's preferred outbound connection security mechanism. The relational database directory also allows associating an ARD program with a relational database name.

Each IBM i operating system in the distributed relational database network must have a relational database directory configured. There is only one relational database directory on a system. Each client in the distributed relational database network must have an entry in its relational database directory for its local relational database and one for each remote and local user relational database that the client accesses. Any system in the distributed relational database network that acts only as a server does not need to include the relational database names of other remote relational databases in its directory.

The relational database name assigned to the local relational database must be unique. That is, it should be different from any other relational database in the network. Names assigned to other relational databases in the directory identify remote relational databases, or local user databases. The names of remote RDBs must match the name an server uses to identify its local system database or one of its user databases, if configured. If the local system RDB name entry at an server does not exist when it is needed, one will be created automatically in the directory. The name used will be the current system name displayed by the Display Network Attributes (DSPNETA) command.

Related reference

Display Network Attributes (DSPNETA) command

Working with the relational database directory

Use these instructions to work with the relational database directory.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Remove Relational Database Directory Entry (RMVRDBDIRE) command

Work with Relational Database Directory Entry (WRKRDBDIRE) command

Adding an entry for SNA usage

The **Add RDB Directory Entry (ADDRDBDIRE)** display is shown here. You can use the **Add Relational Database Directory Entry (ADDRDBDIRE)** command to add an entry to the relational database directory.

```
              Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Entry:
  Relational database  . . . . .   MP311
  Relational database alias  . .   *NONE
Remote location:
  Name or address  . . . . . . .   MP311
  Type . . . . . . . . . . . . .   *SNA          *SNA, *IP
```

In this example, an entry is made to add a relational database named MP311 for a system with a remote location name of MP311 to the relational database directory on the local system. For SNA connections, the Relational database alias field must remain *NONE, the default value. The remote location name does not have to be defined before a relational database directory entry using it is created. However, the remote location name must be defined before the relational database directory entry is used in an application. The relational database name (RDB) parameter and the remote location name (RMTLOCNAME) parameter are required for the **Add Relational Database Directory Entry (ADDRDBDIRE)** command. By default, the second element of the RMTLOCNAME parameter is *SNA. The descriptive text (TEXT) parameter is optional. As shown in this example, it is a good idea to make the relational database name the same as the system name or location name specified for this system in your network configuration. This can help you identify a database name and correlate it to a particular system in your distributed relational database network, especially if your network is complex.

To see the other optional parameters on this command, press F9 on the **Add RDB Directory Entry (ADDRDBDIRE)** display. These optional parameters are shown here.

```
              Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Entry:
  Relational database  . . . . .   MP311
  Relational database alias  . .   *NONE
Remote location
  Name or address  . . . . . . .   MP311
  Type . . . . . . . . . . . . .   *SNA         *SNA, *IP

...

Device:
  APPC device description  . . .  *LOC          Name, *LOC
Local location . . . . . . . . .  *LOC          Name, *LOC, *NETATR
Remote network identifier  . . .  *LOC          Name, *LOC, *NETATR, *NONE
Mode . . . . . . . . . . . . . .  *NETATR       Name, *NETATR
Transaction program  . . . . . .  *DRDA         Character value, *DRDA
Text . . . . . . . . . . . . . .  'Oak Street Dealership'
```

The system provides *SNA as the default value for the following additional **`Add Relational Database Directory Entry (ADDRDBDIRE)`** command parameters:

- Device (DEV)
- Local location (LCLLOCNAME)
- Remote network identifier (RMTNETID)
- Mode (MODE)
- Transaction program (TNSPGM)

Notes:

1. For SNA connections, the relational database alias field must be left with its *NONE default value.
2. The transaction program name parameter in the IBM i operating system is TNSPGM. In SNA, it is TPN.
3. If you use the defaults with Advanced Program-to-Program Communication (APPC), the system determines the device, the local location, and the remote network identifier that will be used. The mode name defined in the network attributes is used and the transaction program name for Distributed Relational Database Architecture (DRDA) support is used. If you use the defaults with Advanced Peer-to-Peer Networking (APPN), the system ignores the device (DEV) parameter, and uses the local location name, remote network identifier, and mode name defined in the network attributes.

You can change any of these default values on the ADDRDBDIRE command. For example, you might have to change the TNSPGM parameter to communicate with a DB2 for VM server. By default for DB2 for VM support, the transaction program name is the name of the DB2 for VM database to which you want to connect. The default TNSPGM parameter value for DRDA (*DRDA) is X'07F6C4C2'. QCNTEDDM and DB2DRDA also map to X'07F6C4C2'.

Related tasks

Setting QCNTSRVC as a transaction program name on a DB2 for VM client
Change the UCOMDIR NAMES file to specify QCNTSRVC in the TPN tag.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Setting QCNTSRVC as a transaction program name on a Db2 for i application requester
Specify the QCNTSRVC on the TNSPGM parameter of the **`Add Relational Database Directory Entry (ADDRDBDIRE)`** or **`Change Relational Database Directory Entry (CHGRDBDIRE)`** command.

Setting QCNTSRVC as a transaction program name on a DB2 for z/OS client
Update the SYSIBM.LOCATIONS table to specify QCNTSRVC in the TPN column for the row that contains the RDB-NAME of the Db2 for i server.

Setting QCNTSRVC as a transaction program name on a DB2 for Linux, UNIX, and Windows client

If you are working with DB2 for Linux, UNIX, and Windows and would like instructions on how to set up the TPN on this family of products, there is a Web page to help you.

Adding an entry for TCP/IP usage

The **Add RDB Directory Entry (ADDRDBDIRE)** display demonstrates how the panel changes if you enter *IP as the second element of the RMTLOCNAME parameter, and what typical entries look like for an RDB that uses TCP/IP.

Although usage of the **Relational database alias** field is enabled for connections that use TCP/IP, this first TCP/IP example does not specify an alias.

```
                     Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Entry:
  Relational database  . . . . . > MP311
  Relational database alias  . .   *NONE
Remote location:
  Name or address  . . . . . . .   MP311.spiffy.com



  Type . . . . . . . . . . . . . > *IP           *SNA, *IP
Port number or service program     *DRDA
Remote authentication method:
  Preferred method . . . . . . .   *USRENCPWD    *USRENCPWD, *USRID...
  Allow lower authentication . .   *ALWLOWER     *ALWLOWER, *NOALWLOWER
Encryption algorithm . . . . . .   *DES          *DES, *AES
Secure connection  . . . . . . .   *NONE         *NONE, *SSL

                                                                    More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Specifying a relational database alias name

This example shows the addition of a directory entry that specifies an RDB alias name. This allows networks that have relational databases of the same name to uniquely identify each in a Distributed Relational Database Architecture (DRDA) environment.

When an entry using an alias has been added to the RDB directory, the entry is identified by its alias name. To display or delete the entry, you must specify the alias name.

The following display has RDBALS specified as the relational database alias name.

```
                     Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Entry:
  Relational database  . . . . . > TEST
  Relational database alias  . .   RDBALS
Remote location:
  Name or address  . . . . . . .   MP311.spiffy.com



  Type . . . . . . . . . . . . . > *IP           *SNA, *IP
Port number or service program     *DRDA
Remote authentication method:
  Preferred method . . . . . . .   *USRENCPWD    *USRENCPWD, *USRID...
  Allow lower authentication . .   *ALWLOWER     *ALWLOWER, *NOALWLOWER
Encryption algorithm . . . . . .   *DES          *DES, *AES
Secure connection  . . . . . . .   *NONE         *NONE, *SSL

                                                                    More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

When you add an entry for an alias using the Work with Relational Database Directory Entry (WRKRDBDIRE) command and option 1, you should first put the real RDB name in the **Entry** field, and press Enter. Then, after filling in the other fields including the alias name in the **Relational database alias** field, you will see the alias name replace the real RDB name in the **Entry** field of the list of RDB entries. You must change **Type** for the remote location name from *SNA to *IP. You cannot add an alias to a relational database directory entry that specifies *LOCAL as the remote location name.

When removing a relational database entry with the **Remove Relational Database Directory Entry (RMVRDBDIRE)** command, the alias name, rather than the real relational database name, is used to specify which entry to remove.

If you identify a remote database by an alias, you cannot also refer to it by its real name in the same directory.

Instead of specifying MP311.spiffy.com for the **RMTLOCNAME** parameter, you can specify the IP address (for example, 9.5.25.176). For IP connections to another IBM i environment, leave the PORT parameter value set at the default, *DRDA, unless you need to use port 447. For example, you might have port 447 configured for transmission using IP Security Architecture (IPSec). For connections to an IBM DB2 server on some other platform, for example, you might need to set the port to a number such as 50000. Refer to the product documentation for the system you are using. If you have a valid service name defined for a DRDA port at a certain location, you can also use that name instead of a number. However, on the IBM i operating system, *DRDA is preferred to the use of the DRDA service name.

Adding an entry for an application requester driver

To specify communication information and an application requester driver (ARD) program on the **Add Relational Database Directory Entry (ADDRDBDIRE)** command prompt, press F9 (All parameters) and page down.

When the ARD program will not use the communication information specified on the **ADDRDBDIRE** command (which is normally the case), use the special value *ARDPGM on the RMTLOCNAME parameter. The ARD program must reside in a library in the system database (ASP numbers 1-32).

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Using the WRKRDBDIRE command

The **Work with RDB Directory Entries** display provides you with options to add, change, display, or remove a relational database directory entry.

```
               Work with Relational Database Directory Entries

 Position to  . . . . . .

 Type options, press Enter.
   1=Add   2=Change   4=Remove   5=Display details   6=Print details

                             Remote
 Option  Entry               Location                  Text

   _      KC000               KC000     Kansas City region database
   _      MP000               *LOCAL    Minneapolis region database
   _      MP101               MP101     Dealer database MP101
   _      MP102               MP102     Dealer database MP102
   _      MP211               MP211     Dealer database MP211
   _      MP215               MP215     Dealer database MP215
   4      MP311               MP311     Dealer database MP311
   _      MP415               MP415     Dealer database MP415
   _      MP515               MP515     Dealer database MP515
   _      MP615               MP615     Dealer database MP615
   _      MP715               MP715     Dealer database MP715
                                                                More...
 F3=Exit   F5=Refresh   F6=Print list   F12=Cancel   F22=Display entire field
```

As shown on the display, option 4 can be used to remove an entry from the relational database directory on the local system. If you remove an entry, you receive another display that allows you to confirm the remove request for the specified entry or select a different relational database directory entry. If you use the **Remove Relational Database Directory Entry (RMVRDBDIRE)** command, you have the option of specifying a specific relational database name, generic names, all directory entries, or just the remote entries.

You have the option on the Work with Relational Database Directory Entries display to display the details of an entry. Output from the Work with Relational Database Entries display is sent to a display. However, if you use the **Display Relational Database Directory Entry (DSPRDBDIRE)** command, you can send the output to a printer or an output file. The relational database directory is not an IBM i object, so using an output file provides a means of backup for the relational database directory. For more information about using the **Display Relational Database Directory Entries (DSPRDBDIRE)** command with an output file for backing up the relational database directory, see Saving and restoring relational database directories.

You have the option on the Work with RDB Directory Entries display to change an entry in the relational database directory. You can also use the **Change Relational Database Directory Entry (CHGRDBDIRE)** command to make changes to an entry in the directory. You can change any of the optional command parameters and the remote location name of the system. You cannot change a relational database name for a directory entry. To change the name of a relational database in the directory, remove the entry for the relational database and add an entry for the new database name.

Note: If the remote location was changed in the relational database directory entry, then the remote journal has to be removed using the **Remove Remote Journal (RMVRMTJRN)** command or the QjoRemoveRemoteJournal API and readded using the **Add Remote Journal (ADDRMTJRN)** command or the QjoAddRemoteJournal API. If the remote location type, or authentication, or something else was changed, then remote journaling just needs to be ended using the **Change Remote Journal (CHGRMTJRN)** command or the QjoChangeJournalState API and restarted by also using the **Change Remote Journal (CHGRMTJRN)** command or the QjoChangeJournalState API. To get your change used for distributed files, you need to delete and re-create your node group, and then re-create the file.

Related tasks

Saving and restoring relational database directories
The relational database directory is not an IBM i object. Instead, it is made up of files that are opened by the system at initial program load (IPL) time.

Related reference

Add Remote Journal (ADDRMTJRN) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Change Remote Journal (CHGRMTJRN) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Remove Remote Journal (RMVRMTJRN) command

Remove Relational Database Directory Entry (RMVRDBDIRE) command

*The *LOCAL directory entry*

The directory entry containing *LOCAL is unique in that there is only one such entry in the directory, and that it specifies the name of the local system database.

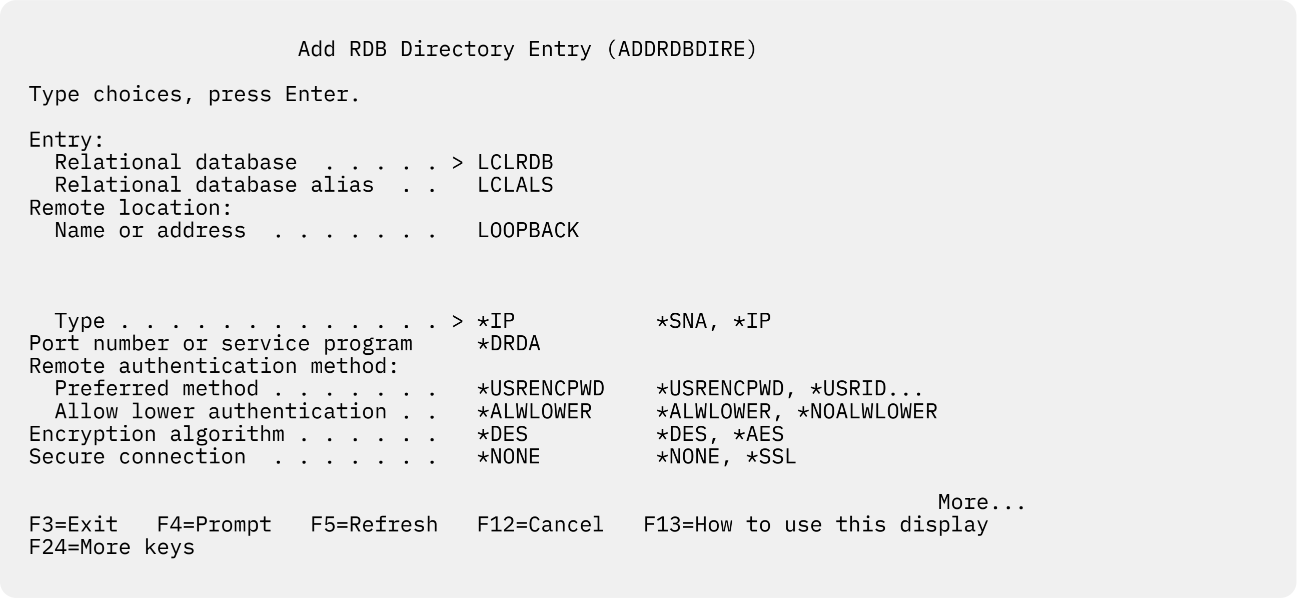
The associated RDB name can be used in the SQL statement CONNECT TO xxx (where xxx is the local system name) to connect to the local database. The effect of CONNECT TO xxx is equivalent to using the SQL statement CONNECT RESET.

If you want to make a DRDA connection to the local system database, such as for program testing, there are two special RDB names that can be used for that purpose: ME and MYSELF. For example, a programmer adds a directory entry with an RDB name of ME, with type of *IP, and with the Remote Location name of LOOPBACK. The programmer can then, in a program, run an SQL CONNECT TO ME statement and establish a sockets DRDA connection to the local system. However, general use of these RDB names is discouraged and they are documented only to warn that unexpected behavior can result from using them in some situations.

However, if you must change the name of the local RDB entry, the procedure includes doing the remove and add operation. But there are special considerations for removing the local entry, because that entry contains some system-wide DRDA attribute information. If you try to remove the entry, you will get message CPA3E01 (Removing or changing *LOCAL directory entry might cause loss of configuration data (C G)), and you will be given the opportunity to cancel (C) the operation or continue (G). The message text goes on to tell you that the entry is used to store configuration data entered with the **`Change DDM TCP/IP Attributes (CHGDDMTCPA)`** command. If the *LOCAL entry is removed, configuration data might be destroyed, and the default configuration values will be in effect. If the default values are not satisfactory, configuration data will have to be re-entered with the CHGDDMTCPA command. Before removing the entry, you might want to record the values specified in the CHGDDMTCPA command so that they can be restored after the *LOCAL entry is deleted and added with the correct local RDB name.

You cannot add an alias to a relational database directory entry that specifies *LOCAL as the remote location name. Message CPD3EC8 is displayed if you attempt to do so.

Server side RDB aliases are supported over TCP/IP. To create a server side RDB aliases, use the RDB name from the *LOCAL entry, the desired alias name, the remote location LOOPBACK and type *IP. The DRDA server jobs will then accept incoming requests against the alias name.



```
                     Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Entry:
  Relational database  . . . . . > LCLRDB
  Relational database alias  . .   LCLALS
Remote location:
  Name or address  . . . . . . .   LOOPBACK



  Type . . . . . . . . . . . . . > *IP           *SNA, *IP
Port number or service program     *DRDA
Remote authentication method:
  Preferred method . . . . . . .   *USRENCPWD    *USRENCPWD, *USRID...
  Allow lower authentication . .   *ALWLOWER     *ALWLOWER, *NOALWLOWER
Encryption algorithm . . . . . .   *DES          *DES, *AES
Secure connection  . . . . . . .   *NONE         *NONE, *SSL

                                                                    More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Directory entries for user databases on independent auxiliary storage pools

For a system with only one database (that is, without independent auxiliary storage pools (independent ASPs) configured), the *LOCAL entry refers to the single local database. For systems with multiple databases (one system database and one or more user databases), the *LOCAL entry refers to the system database.

The local user databases are represented by entries similar to remote *IP entries. The main difference is the Remote location field. In cases where the database cannot be switched to a different system, this field normally contains the word LOOPBACK. LOOPBACK represents the IP address of the host system. If the database can be switched, it is suggested that the user configure the system in such a way that a specific IP address is associated with the database regardless of the system to which it is attached. For an explanation on how dedicated IP address configuration is done, see the Managing application CRG takeover IP addresses topic. In that case, the IP address is used in the Remote location field.

If LOOPBACK is used for a switchable database, then whenever it is switched from the local system, the user will have to manually change the directory entry to replace LOOPBACK with the IP address of the new system to which it is attached, and then change it back to LOOPBACK when the database is switched back.

Related reference

Managing application CRG takeover IP addresses

Example: Setting up a relational database directory

The Spiffy Corporation network example illustrates how the relational database directory is set up and used on systems in a distributed relational database network.

The example assumes the use of Advanced Program-to-Program Communication (APPC) for communications, as opposed to TCP/IP, which would be simpler to set up. However, some elements of the example are protocol-independent. The RDB directory entries needed for APPC use are also needed in a TCP/IP network, but the parameters differ. Host names or IP addresses and port identifications would replace logical unit (LU) names, device descriptions, modes, TPNs, and so forth.

A simple relationship to consider is the one between two regional offices as shown in the following figure:

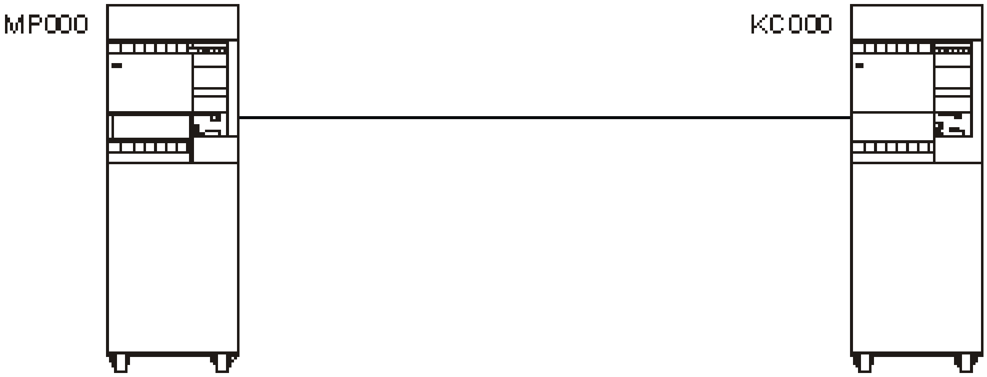


Figure 20. Relational database directory set up for two systems

The relational database directory for each regional office must contain an entry for the local relational database and an entry for the remote relational database because each system is both a client and a server. The commands to create the relational database directory for the MP000 system are:

```
ADDRDBDIRE    RDB(MP000) RMTLOCNAME(*LOCAL) TEXT('Minneapolis region database')

ADDRDBDIRE    RDB(KC000) RMTLOCNAME(KC000) TEXT('Kansas City region database')
```

In the preceding example, the MP000 system identifies itself as the local relational database by specifying *LOCAL for the RMTLOCNAME parameter. There is only one relational database on a IBM i platform. You can simplify identification of your network relational databases by making the relational database names in the RDB directory the same as the system name. The entry for the local location can have the same name as the local system name, and the entry for the remote location name can have the same name as the remote system name.

Note: The system name is specified on the SYSNAME parameter of the Change Network Attributes (CHGNETA) command. The local system is identified on the LCLLOCNAME parameter of the CHGNETA command during communications configuration. Remote locations using SNA (APPC) are identified with the RMTCPNAME parameter on the Create Controller Description (APPC) (CRTCTLAPPC) command during communications configuration. Using the same names for system names, network locations, and database names can help avoid confusion, particularly in complex networks.

The corresponding entries for the KC000 system relational database directory are:

```
ADDRDBDIRE    RDB(KC000) RMTLOCNAME(*LOCAL) TEXT('Kansas City region database')

ADDRDBDIRE    RDB(MP000) RMTLOCNAME(MP000) TEXT('Minneapolis region database')
```

A more complex example to consider is that of a regional office to its dealerships. For example, to access relational databases in the network shown in the following figure, the relational database directory for the MP000 system must be expanded to include an entry for each of its dealerships.

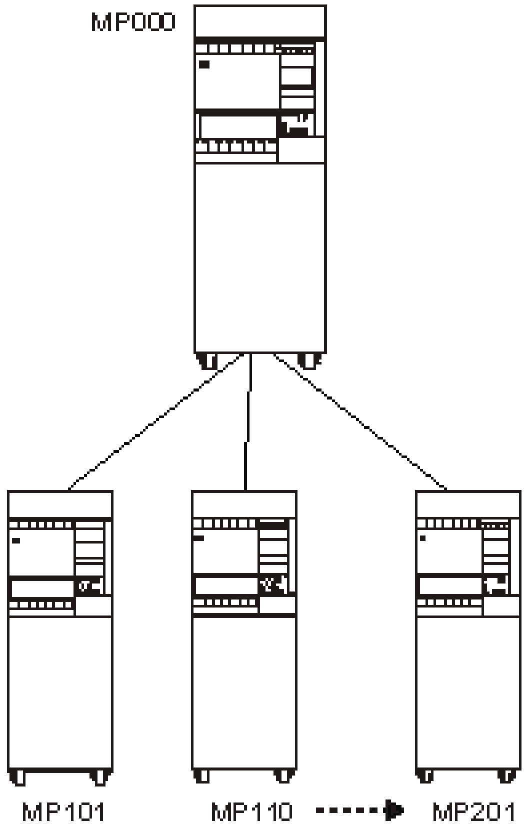


Figure 21. Relational database directory setup for multiple systems

A sample of the commands used to complete the MP000 relational database directory to include all its dealer databases is as follows:

```
PGM
ADDRDBDIRE    RDB(MP000) RMTLOCNAME(*LOCAL) +
TEXT('Minneapolis region database')
ADDRDBDIRE    RDB(KC000) RMTLOCNAME(KC000)
TEXT('Kansas City region database')
ADDRDBDIRE    RDB(MP101) RMTLOCNAME(MP101)
TEXT('Dealer database MP101')
ADDRDBDIRE    RDB(MP002) RMTLOCNAME(MP110)
TEXT('Dealer database MP110')
.
.
.
ADDRDBDIRE    RDB(MP215) RMTLOCNAME(MP201)
TEXT('Dealer database MP201')
ENDPGM
```

In the preceding example, each of the region dealerships is included in the Minneapolis relational database directory as a remote relational database.

Because each dealership can serve as a client to MP000 and to other dealership servers, each dealership must have a relational database directory that has an entry for itself as the local relational database, and have the regional office and all other dealers as remote relational databases. The database administrator has several options to create a relational database directory at each dealership system.

The most time-consuming and error-prone method is to create a relational database directory at each system by using the Add Relational Database Directory Entry (ADDRDBDIRE) command to create each directory entry on all systems that are part of the MP000 distributed relational database network.

A better alternative is to create a control language (CL) program like the one shown in the preceding example for the MP000 system. The distributed relational database administrator can copy this CL program for each of the dealership systems. To customize this program for each dealership, the database administrator changes the remote location name of the MP000 system to MP000, and changes the remote location name of the local dealership to *LOCAL. The distributed relational database administrator can distribute the customized CL program to each dealership to be run on that system to build its unique relational database directory.

A third method is to write a program that reads the relational database directory information sent to an output file as a result of using the Display Relational Database Directory Entry (DSPRDBDIRE) command. This program can be distributed to the dealerships, along with the output file containing the relational database directory entries for the MP000 system. Each system can read the MP000 output file to create a local relational database directory. The Change Relational Database Directory Entry (CHGRDBDIRE) command can then be used to customize the MP000 system directory for the local system.

Related tasks

Saving and restoring relational database directories
The relational database directory is not an IBM i object. Instead, it is made up of files that are opened by the system at initial program load (IPL) time.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Create Controller Description (APPC) (CRTCTLAPPC) command

Display Network Attributes (DSPNETA) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Setting up security

Distributed Relational Database Architecture (DRDA) security is covered in the Security topic, but for the sake of completeness, it is mentioned here as a consideration before using DRDA, or in converting your network from the use of Advanced Program-to-Program Communication (APPC) to TCP/IP.

Security setup for TCP/IP is quite different from what is required for APPC. One thing to be aware of is the lack of the secure location concept that APPC has. Because a TCP/IP server cannot fully trust that a client system is who it says it is, the use of passwords on connection requests is more important. To make it easier to send passwords on connection requests, the use of server authentication lists associated with specific user profiles has been introduced with TCP/IP support. Entries in server authentication lists can be maintained by use of the *xxx*SVRAUTE commands (where xxx represents ADD, CHG, and RMV) described in "Security" on page 71. An alternative to the use of server authentication entries is to use the USER/USING form of the SQL CONNECT statement to send passwords on connection requests.

Kerberos support provides another security option if you are using TCP/IP. Network authentication service supports Kerberos protocols and can be used to configure for Kerberos.

Setup at the server side includes deciding and specifying what level of security is required for inbound connection requests. For example, should unencrypted passwords be accepted? The default setting is that they are. The default setting can be changed by use of the Change DDM TCP/IP Attributes (CHGDDMTCPA) command.

Related concepts

Control language

Related tasks

Configuring network authentication service

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Setting up the TCP/IP server

If you own a Distributed Relational Database Architecture (DRDA) server that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Setting up the TCP/IP server can be as simple as ensuring that it is started when it is needed, which can be done by running the following command if you want it to remain active at all times:

```
CHGDDMTCPA AUTOSTART(*YES)
```

But there are other parameters that you might want to adjust to tune the server for your environment. These include the initial number of prestart jobs to start, the maximum number of jobs, threshold when to start more, and so forth.

You might want to set up a common user profile for all clients to use when connecting, or set up a set of different user profiles with different levels of security for different classes of remote users. You can then use the Add Server Authentication Entry (ADDSVRAUTE) command at the client to map each user's profile name at the client to what user profile they will run under at the server.

Related concepts

Client security in a TCP/IP network
Different connectivity scenarios call for using different levels of authentication. Therefore, an administrator can set the lowest security authentication method required by the client when connecting to a server by setting the preferred authentication method field in each RDB directory entry.

Managing the TCP/IP server
The DRDA and DDM TCP/IP server does not typically require any changes to your existing system configuration. At some time, you might want to change the way the system manages the server jobs to better meet your needs, to solve a problem, to improve the system performance, or to look at the jobs on the system.

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Setting up SQL packages for interactive SQL

This topic applies only to server systems that are not running IBM i.

If either of the following items is true, then you need to ensure that SQL packages are created at the systems:

- If you have the IBM DB2 Query Manager and SQL Development Kit for i licensed program and plan to use the interactive SQL function of that product
- If you plan to connect to DRDA servers other than IBM i that use TCP/IP from a pre-V5R1 OS/400® client, or to ones that do not have two-phase commit capability

Interactive SQL does not require SQL packages for IBM i. Normally, SQL packages are created automatically for interactive SQL users at a server system that does not run IBM i. However, a problem can occur because the initial connection for interactive SQL is to the local system, and that connection is protected by two-phase commit protocols. If a subsequent connection is made to a system that is only one-phase commit capable, or if TCP/IP is used from a pre-V5R1 OS/400 client, then that connection is read-only. When an attempt is made to automatically create a package over such a connection, it fails because the creation of a package is considered an update, and cannot be done over a read-only connection.

The solution to this is to end the connection to the local database before connecting to the remote server system. This can be done by doing a RELEASE ALL command followed by a COMMIT. Then the connection to the remote system can be made and because it is the first connection, updates can be made over it.

When you start interactive SQL, you must specify a commitment control level other than *NONE. Also, the user ID that you use to connect with must have the proper authority to create an SQL package on the server. If you receive an SQLSTATE of 42501 on the connection attempt, you might not have package creation authority.

Related reference

Connection failures specific to interactive SQL
Sometimes when you are running a CONNECT statement from interactive SQL, a general SQ30080 message is given.

Setting up DDM files

The IBM i implementation of Distributed Relational Database Architecture (DRDA) support uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

You can use DDM to submit remote commands to a server, copy tables from one system to another, and process nondistributed relational database work on another system.

With distributed relational database, information the client needs to connect to a database is provided in the relational database directory. When you use DDM, you must create a separate DDM file for each file you want to work with on the server. The DDM file is used by the application on the client to identify a remote file on the server and the communications path to the server.

As of V5R2, you can also create DDM files with a reference to an RDB directory entry. Some database administration tasks discussed in Managing a distributed relational database use DDM to access remote files. A DDM file is created using the Create Distributed Data Management File (CRTDDMF) command. You can create a DDM file before the file and communication path named in the file have been created. However, the file named in the DDM file and the communications information must be created before the DDM file is used by an application.

The following example shows one way a DDM file can be created:

```
CRTDDMF FILE (TEST/KC105TST) RMTLOCNAME(KC105)
        RMTFILE(SPIFFY/INVENT)
```

If the DDM file access in the example is to be over TCP/IP, you must specify *IP in the second element of the RMTLOCNAME parameter.

This command creates a DDM file named KC105TST and stores it in the TEST library on the client. This DDM file uses the remote location KC105 to access a remote file named INVENT, which is stored in the SPIFFY library on the server system.

You can use options on the Work with DDM Files display to change, delete, display or create DDM files.

Related concepts

DRDA and DDM administration
As an administrator for a distributed relational database, you are responsible for work that is done on several systems.

Operating remote systems
As an administrator in a distributed relational database, you might have to operate a remote IBM i product.

Related reference

Create Distributed Data Management File (CRTDDMF) command

Loading data into tables in a distributed relational database

Applications in the distributed relational database environment operate on data stored in tables. In general, applications are used to query a table for information; to insert, update, or delete rows of a table or tables; or to create a new table. Other situations occur where data on one system must be moved to another system.

Loading new data into the tables of a distributed relational database

You load data into a table by entering each data item into the table. On the IBM i operating system, you can use SQL, the Db2 for i Query Management function, or the data file utility portion of IBM i Application Development Tools to create applications that insert data into a table.

Loading data into a table using SQL

A simple method of loading data into a table is to use an SQL application and the SQL INSERT operation.

Consider a situation in which a Spiffy regional center needs to add inventory items to a dealership's inventory table on a periodic basis as regular inventory shipments are made from the regional center to the dealership.

```
INSERT INTO SPIFFY.INVENT
   (PART, DESC, QTY, PRICE)
 VALUES
  ('1234567', 'LUG NUT', 25, 1.15 )
```

The preceding statement inserts one row of data into a table called INVENT in an SQL collection named SPIFFY.

For each item on the regular shipment, an SQL INSERT statement places a row in the inventory table for the dealership. In the preceding example, if 15 different items were shipped to the dealership, the application at the regional office could include 15 SQL INSERT statements or a single SQL INSERT statement using host variables.

In this example, the regional center is using an SQL application to load data into a table at a server. Runtime support for SQL is provided in the IBM i licensed program, so the server does not need the IBM DB2 Query Manager and SQL Development Kit for i licensed program. However, IBM DB2 Query Manager and SQL Development Kit for i is required to write the application.

Related concepts

SQL programming

Related reference

SQL reference

Manipulating data in tables and files using the IBM i query management function

The IBM i licensed program provides a Db2 for i query management function that allows you to manipulate data in tables and files. A query is created using an SQL query statement.

You can run the query through CL commands or through a query callable interface in your application program. Using the query management function, you can insert a row of data into a table for the inventory updates described in “Loading data into a table using SQL” on page 63 as follows.

Create a source member INVLOAD in the source physical file INVLOAD and the SQL statement:

```
INSERT INTO SPIFFY/INVENT
   (PART, DESC, QTY, PRICE)
 VALUES
   (&PARTVALUE, &DESCVALUE, &QTYVALUE, &PRICEVALUE)
```

Use a CL command to create a query management query object:

```
CRTQMQRY QMQRY(INVLOAD) SRCFILE(INVLOAD) SRCMBR(INVLOAD)
```

The following CL command places the INSERT SQL statement results into the INVENT table in the SPIFFY collection. Use of variables in the query (&PARTVALUE, &DESCVALUE, and so on) allows you to enter the desired values as part of the STRQMQRY call, rather than requiring that you create the query management query again for each row.

```
STRQMQRY QMQRY(INVLOAD) RDB(KC000)
   SETVAR((PARTVALUE '1134567'') (DESCVALUE '''Lug Nut''')
    (QTYVALUE 25) (PRICEVALUE 1.15))
```

The query management function is dynamic, which means its access paths are built at run time instead of when a program is compiled. For this reason, the Db2 for i query management function is not as efficient for loading data into a table as an SQL application. However, you need the IBM DB2 Query Manager and SQL Development Kit for i product to write an application. Runtime support for SQL and query management is part of the IBM i licensed program.

Related concepts

Query Management Programming PDF

Entering data, update tables, and make inquiries using data file utility

The data file utility (DFU), which is part of the IBM i Applications Development Tools package available from IBM, is a program builder that helps you create programs to enter data, update tables, and make inquiries.

You do not need a programming language to use DFU. Your data entry, maintenance, or inquiry program is created when you respond to a series of displays. An advantage in using DFU is that its generic nature allows you to create a database update program to load data to a table faster than you can by using programming languages, such as SQL. You can work with data on a remote system by using DFU with DDM files, or by using display station pass-through to run DFU at the server.

Moving data from one system to another

A number of situations occur in enterprise operations that might require moving data from one IBM i operating system to another.

Here is a situation that might require moving data from one system to another: a new dealership opens in a region, and some clients from one or two other dealerships might be transferred to the new dealership as determined by client addresses. Perhaps a dealership closed or no longer represents Spiffy Corporation sales and service. That dealer's inventories and required service information must be allocated to either the regional office or other area dealerships. Perhaps a dealership has grown to the extent that it needs to upgrade its system, and the entire database must be moved to the new system.

Here are some alternatives for moving data from one system to another:

- User-written application programs
- Interactive SQL (ISQL)
- Db2 for i Query Management functions
- Copy to and from tape devices
- Copy file commands with DDM
- The network file commands
- IBM i save and restore commands

Creating a user-written application program

A program compiled with distributed unit of work (DUW) connection management can connect to a remote database and a local database and FETCH from one to INSERT into the other to move the data.

By using multirow FETCH and multirow INSERT, blocks of records can be processed at one time. Commitment control can be used to allow checkpoints to be performed at points during the movement of the data to avoid having to completely start the copy again in case of a failure.

Querying a database using interactive SQL

Using the SQL SELECT statement and interactive SQL, you can query a database on another IBM i operating system for the data that you need in order to create or update a table on the local system.

The SELECT statement allows you to specify the table name and columns containing the desired data, and selection criteria or filters that determine which rows of data are retrieved. If the SELECT statement is successful, the result is one or more rows of the specified table.

In addition to getting data from one table, SQL allows you to get information from columns contained in two or more tables in the same database by using a join operation. If the SELECT statement is successful, the result is one or more rows of the specified tables. The data values in the columns of the rows returned represent a composite of the data values contained in specified tables.

Using an interactive SQL query, the results of a query can be placed in a database file on the local system. If a commitment control level is specified for the interactive SQL process, it applies to the server; the database file on the local system is under a commitment control level of *NONE.

Interactive SQL allows you to do the following things:

- Create a new file for the results of a select.
- Replace an existing file.
- Create a new member in a file.
- Replace a member.
- Append the results to an existing member.

Consider the situation in which the KC105 dealership is transferring its entire stock of part number 1234567 to KC110. KC110 queries the KC105 database for the part they acquire from KC105. The result of this inventory query is returned to a database file that already exists on the KC110 system. This is the process you can use to complete this task:

Use the **`Start SQL (STRSQL)`** command to get the interactive SQL display. Before you enter any SQL statement (other than a CONNECT) for the new database, specify that the results of this operation are sent to a database file on the local system. To do so, follow these steps:

1. Select the Services option from the Enter SQL Statements display.
2. Select the Change Session Attributes option from the Services display.
3. Enter the Select Output Device option from the Session Attributes Display.
4. Type a 3 for a database file in the Output device field and press Enter. The following display is shown:

```
Change File

Type choices, press Enter.

File . . . . . . . . .   QSQLSELECT   Name
Library  . . . . . .   QGPL         Name
Member . . . . . . . .   *FILE        Name, *FILE, *FIRST

Option . . . . . . . .   1            1=Create new file
2=Replace file
3=Create new member
4=Replace member
5=Add to member

For a new file:
Authority  . . . . .   *LIBCRTAUT   *LIBCRTAUT, *CHANGE, *ALL
*EXCLUDE, *USE
authorization list name

Text . . . . . . . .


F3=Exit     F5=Refresh     F12=Cancel
```

5. Specify the name of the database file that is to receive the results.

When the database name is specified, you can begin your interactive SQL processing as shown in the following example.

```
Enter SQL Statements

Type SQL statement, press Enter.
Current connection is to relational database KC000.
CONNECT TO KC105__________________________________________________________
Current connection is to relational database KC105.
====> SELECT * FROM INVENTORY_____________________________________________
WHERE PART = '1234567'____________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
Bottom
F3=Exit   F4=Prompt   F6=Insert line   F9=Retrieve   F10=Copy line
F12=Cancel            F13=Services     F24=More keys
```

Related concepts

SQL programming

Related reference

SQL reference

Querying remote systems using Db2 for i query management function

The Db2 for i query management function provides almost the same support as interactive SQL for querying a remote system and returning the results in an output file to the local system.

Both interactive SQL and the query management function can perform data manipulation operations, such as INSERT, DELETE, and SELECT, for files or tables without the requirement that the table (or file) already exist in a collection (it can exist in a library). Also, query management uses SQL CREATE TABLE statements to provide a data definition when a new table is created on the system as a result of the query. Tables created from a query management function follow the same guidelines and restrictions that apply to a table created using SQL.

However, the query management function does not allow you to specify a member when you want to add the results to a file or table. The results of a query function are placed in the first file member unless you use the **`Override with Database File (OVRDBF)`** command to specify a different member before starting the query management function.

Related concepts

Query Management Programming PDF

Related reference

Override with Database File (OVRDBF) command

Copying files to and from tape

There are several different commands that you can use to copy files to and from tape.

You can copy a table or file to tape using the Copy to Tape (CPYTOTAP) command on the IBM i operating system.

Data on tape can be loaded on another system using the Copy from Tape (CPYFRMTAP) command. For more information about using the command, see the Storage solutions topic.

You can also use the Copy File (CPYF) command to load data on tape into Db2 for i. This is especially useful when you load data that was unloaded from DB2 for z/OS or DB2 Server for VM (SQL/DS). Nullable data can be unloaded from these systems in such a way that a single-byte flag can be associated with each nullable field. The CPYF command, with the *NULLFLAGS option specified for the FMTOPT parameter, can recognize the null flags and ignore the data in the adjacent field on the tape and make the field null in Db2 for i. Another useful FMTOPT parameter value for importing data from IBM mainframes is the *CVTFLOAT value. It allows floating-point data stored on tape in the z/OS format to be converted to the IEEE format used by Db2 for i.

Related concepts

Storage solutions

Related reference

Copy To Tape (CPYTOTAP) command

Copy From Tape (CPYFRMTAP) command

Copy File (CPYF) command

Moving data between systems using copy file commands

Another way to move data from one IBM i operating system to another is to copy the data using the copy file commands with DDM.

You can use the Copy File (CPYF), Copy Source File (CPYSRCF), and Copy From Query File (CPYFRMQRYF) commands to copy data between files on clients and servers. You can copy local relational database or device files from (or to) remote database files, and remote files can also be copied to remote files.

For example, if a dealership closes, the distributed relational database administrator can copy the client and inventory tables from the remote system to the local regional system. The administrator needs a properly authorized user profile on the server to access and copy the tables and must create a DDM file on the client for each table or file that is copied. The following example shows the command that the database administrator uses to copy a table called INVENT in a collection called SPIFFY from a system with a remote location name of KC105 to a regional center system called KC000. A DDM file called INCOPY in a library called TEST on the KC000 client is used for the file access. These commands are run on the KC000 system:

```
CRTDDMF   FILE(TEST/INCOPY) RMTFILE(SPIFFY/INVENT)
   RMTLOCNAME(KC105)
CPYF      FROMFILE(TEST/INCOPY)  TOFILE(TEST/INVENTDDM)
   MBROPT(*ADD)
```

In this example, the administrator runs the commands on the KC000 system. If the administrator is not on the KC000 system, then pass-through must be used to run these commands on the KC000 system. The Submit Remote Command (SBMRMTCMD) command cannot be used to run the preceding commands because the IBM i operating system cannot be a client and a server for the same job.

Consider the following items when using this command with DDM:

- A DDM file can be specified on the FROMFILE and the TOFILE parameters for the Copy File (CPYF) command and Copy Source File (CPYSRCF) commands.

  Note: For the Copy From Query File (CPYFRMQRYF) and Copy from Tape (CPYFRMTAP) commands, a DDM file name can be specified only on the TOFILE parameter; for the Copy to Tape (CPYTOTAP) command, a DDM file name can be specified only on the FROMFILE parameter.
- When a delete-capable file is copied to a non-delete capable file, you must specify COMPRESS(*YES), or an error message is sent and the job ends.
- If the remote file name on a DDM file specifies a member name, the member name specified for that file on the Copy File (CPYF) command must be the same as the member name on the remote file name on the DDM file. In addition, the Override with Database File (OVRDBF) command cannot specify a member name that is different from the member name on the remote file name on the DDM file.
- If a DDM file does not specify a member name and if the Override with Database File (OVRDBF) command specifies a member name for the file, the Copy File (CPYF) command uses the member name specified on the OVRDBF command.
- If the TOFILE parameter is a DDM file that refers to a file that does not exist, CPYF creates the file. Keep the following special considerations for remote files created with the Copy File (CPYF) command in mind:
  - The user profile for the target DDM job must be authorized to the Create Physical File (CRTPF) command on the server.

  - For an IBM i server system, the TOFILE parameter has all the attributes of the FROMFILE parameter except those described in the Database file management topic collection.
- When using TCP/IP, the second element of the RMTLOCNAME parameter of the Create Distributed Data Management File (CRTDDMF) command must be *IP.

Related concepts

Database file management

Related reference

Copy File (CPYF) command

Copy Source File (CPYSRCF) command

Copy From Query File (CPYFRMQRYF) command

Copy from Tape (CPYFRMTAP) command

Copy To Tape (CPYTOTAP) command

Create Physical File (CRTPF) command

Create Distributed Data Management File (CRTDDMF) command

Override with Database File (OVRDBF) command

Submit Remote Command (SBMRMTCMD) command

Transferring data over network using network file commands

Data can be transferred over networks protocols that support Systems Network Architecture (SNA) distribution services (SNADS). In addition to APPC and APPN protocols used with distributed relational database processing, SNADS can be used with binary synchronous equivalence link (BSCEL) and SNA Upline Facility (SNUF) protocols.

A IBM i environment supported by SNADS can send data to another system with the **Send Network File (SNDNETF)** command and receive a network file from another system with the **Receive Network File (RCVNETF)** and **Work with Network Files (WRKNETF)** commands.

Related reference

Receive Network File (RCVNETF) command

Send Network File (SNDNETF) command

Work with Network File (WRKNETF) command

Moving a table using object save and restore commands

You can move a table from another system using the **Save Object (SAVOBJ)** and **Restore Object (RSTOBJ)** commands. The save commands save database files on tape or in a save file. The save file can be distributed to another system through communications.

The save and restore commands used to save and restore tables or files include:

- **Save Library (SAVLIB)** command saves one or more collections or libraries
- **Save Object (SAVOBJ)** command saves one or more objects (including database tables and views)
- **Save Changed Object (SAVCHGOBJ)** command saves any objects that have changed since either the last time the collection or library was saved or from a specified date
- **Restore Library (RSTLIB)** command restores a collection or library
- **Restore Object (RSTOBJ)** command restores one or more objects (including database tables and views)

For example, if two dealerships are merging, the save and restore commands can be used to save collections and tables for one relational database, which are then restored on the remaining system's relational database. To accomplish this, an administrator would:

1. Use the **Save Library (SAVLIB)** command on system A to save a collection, or use the **Save Object (SAVOBJ)** command on system A to save a table.
2. Specify whether the data is saved to a save file, which can be distributed using SNADS, or saved on tape.
3. Distribute the save file to system B, or send the tape to system B.

4. Use the **Restore Library (RSTLIB)** command on system B to restore a collection, or use the **Restore Object (RSTOBJ)** command on system B to restore a table.

A consideration when you use the save and restore commands is the ownership and authorizations to the restored object. A valid user profile for the current object owner should exist on the system where the object is restored. If the current owner's profile does not exist on this system, the object is restored under the QDFTOWN default user profile. User authorizations to the object are limited by the default user profile parameters. A user with QSECOFR authority must either create the original owner's profile on this system and make changes to the restored object ownership, or specify new authorizations to this object for both local and remote users.

Related concepts

Backup and recovery

Related reference

Restore Library (RSTLIB) command

Restore Object (RSTOBJ) command

Save Changed Object (SAVCHGOBJ) command

Save Library (SAVLIB) command

Save Object (SAVOBJ) command

Moving a database to IBM i from a system other than IBM i

You might need to move a file from another IBM system to the IBM i operating system or from a non-IBM system to IBM i. This topic collection lists alternatives for moving data to the IBM i from a system other than IBM i. However, you must refer to manuals supplied with the other system or identified for the application for specific instructions on its use.

Moving data from another IBM system

There are a number of methods you can use to move data from another IBM system to the IBM i operating system.

You can use the methods listed here to move data:

- A high-level language program can be written to extract data from another system. A corresponding program for the system can be used to load data.
- For systems supporting other Distributed Relational Database Architecture (DRDA) implementations, you can use SQL functions to move data. For example, with distributed unit of work, you can open a query against the source of the data and, in the same unit of work, insert the data into a table on the system. For best performance, blocking should be used in the query and a multirow insert should be done at the system.
- Data can be extracted from tables and files on the other system and sent to the IBM i operating system on tape or over communications lines.
  - From a DB2 for z/OS database, a sample program called DSNTIAUL, supplied with the database manager, can be used to extract data from files or tables.
  - From a DB2 Server for VM (SQL/DS) database, the Database Services Utility portion of the database manager can be used to extract data.
  - From both DB2 for z/OS and DB2 Server for VM databases, Data Extract (DXT) can be used to extract data. However, DXT handling of null data is not compatible with the Copy File handling of null data described below. Therefore, DXT is not suggested for use in unloading relational data for migration to IBM i.
  - From IMS/DB hierarchical databases, DXT can be used to extract data.
- You can use standard tape management techniques to copy data to tape from DB2 for z/OS or DB2 Server for VM databases. The IBM i operating system uses the **Copy from Tape (CPYFRMTAP)** command to load data from tape. The **Copy File (CPYF)** command, however, provides special support for migrating data from IBM mainframe computers. **CPYF** can be used with tape data by the use of the **Override with Tape File (OVRTAPF)** command. The **OVRTAPF** command lets you specify

special tape-specific parameters which might be necessary when you import data from a system other than IBM i.

The special **CPYF** support lets you import nullable data and floating point data. Nullable data can be unloaded from mainframes in such a way that a single-byte flag can be associated with each nullable field. With the *NULLFLAGS option specified for the FMTOPT parameter, the **Copy File (CPYF)** command can recognize the null flags and ignore the data in the adjacent field on the tape and make the field null in Db2 for i. The other useful FMTOPT parameter value for importing data from IBM mainframes is the *CVTFLOAT value. It allows floating point data stored on tape in z/OS format to be converted to the IEEE format used by Db2 for i.

For more information about using tape devices with the IBM i operating system, see the Storage solutions topic.

- Data sent over communications lines can be handled through SNADS support on the IBM i operating system. SNADS support transfers network files for BSCEL and SNUF protocols in addition to the Advanced Program-to-Program Communication (APPC) or Advanced Peer-to-Peer Networking (APPN) protocols used for distributed relational database processing.
  - From an MVS™ system, data can be sent to the IBM i operating system using TSO XMIT functions. The system uses the **Work with Network Files (WRKNETF)** or **Receive Network File (RCVNETF)** command to receive a network file.
  - From a VM system, data can be sent to the IBM i operating system using SENDFILE functions. The system uses the **Work with Network Files (WRKNETF)** or **Receive Network File (RCVNETF)** command to receive a network file.
- From Microsoft Windows, client data can be sent to IBM i using IBM i Access, a separately orderable IBM product.
- From a variety of workstation clients, you can use the DB2 for Linux, UNIX, and Windows IMPORT and EXPORT utilities to copy data to and from IBM i. The IMPORT can import data only into existing tables.

  See the Advanced Functions and Administration on DB2 Universal Database for iSeries, SG24-4249 Redbooks® publication for examples of the IMPORT and EXPORT utilities. This publication also provides information about what file types and data formats can be used with the IMPORT and EXPORT utilities.
- Data can also be sent over communications lines that do not support SNADS, such as asynchronous communications. File transfer support (FTS), a utility that is part of the IBM i licensed program, can be used to send and receive data. For more information about working with communications and communications files, see ICF Programming.

Related concepts

Storage solutions

Related reference

Copy From Tape (CPYFRMTAP) command

Copy File (CPYF) command

Override with Database File (OVRDBF) command

Receive Network File (RCVNETF) command

Tips: Designing distributed relational database applications
Distributed relational database applications have different requirements from applications developed solely for use on a local database.

Work with Network File (WRKNETF) command

Moving data from a non-IBM system
You can copy files or tables to tape from the other system and load these files on the IBM i operating system.

Use the **Copy From Import File (CPYFRMIMPF)** command to do this.

Vendor-independent communications functions are also supported through two separately licensed IBM i programs.

Peer-to-peer connectivity functions for both local and wide area networks are provided by the Transmission Control Protocol/Internet Protocol (TCP/IP). The File Transfer Protocol (FTP) function of the TCP/IP Connectivity Utilities for IBM i licensed program allows you to receive many types of files, depending on the capabilities of the remote system.

Related concepts

TCP/IP setup

Related reference

Copy From Import File (CPYFRMIMPF) command

Security

The IBM i operating system has built in security elements that limit access to data resources of a server. Security options range from simple physical security to full password security coupled with authorization to commands and data objects.

Users must be properly authorized to have access to the database whether it is local or remote. They must also have proper authorization to collections, tables, and other relational database objects necessary to run their application programs. This typically means that distributed database users must have valid user profiles for the databases they use throughout the network. Security planning must consider user and application program needs across the network.

A distributed relational database administrator is faced with two security issues to resolve:

- System to system protection
- Identification of users at remote sites

When two or more systems are set up to access each other's databases, it is important to make sure that the other side of the communications line is the intended location and not an intruder. For DRDA access to a remote relational database, the IBM i use of Advanced Program-to-Program Communication (APPC) and Advanced Peer-to-Peer Networking (APPN) communications configuration capabilities provides options for you to do this network-level security.

The second concern for the distributed relational database administrator is that data security is maintained by the system that stores the data. In a distributed relational database, the user has to be properly authorized to have access to the database (according to the security level of the system) whether the database is local or remote. Distributed relational database network users must be properly identified with a user ID on the server for any jobs they run on the server. Distributed Relational Database Architecture (DRDA) support using both APPC/APPN and TCP/IP communications protocols provides for the sending of user IDs and passwords along with connection requests.

This topic collection discusses security topics that are related to communications and DRDA access to remote relational databases. It discusses the significant differences between conversation-level security in an APPC network connection and the corresponding level of security for a TCP/IP connection initiated by a DRDA application. In remaining security discussions, the term *user* also includes remote users starting communications jobs.

Related reference

Security considerations for a distributed relational database
Part of planning for a distributed relational database involves the decisions you must make about securing distributed data.

Elements of distributed relational database security

A distributed relational database administrator needs to protect the resources of the servers in the network without unnecessarily restricting access to data by clients in the network.

A client secures its objects and relational database to ensure only authorized users have access to distributed relational database programs. This is done using normal IBM i object authorization to identify users and to specify what each user (or group of users) is allowed to do with an object. Alternatively, authority to tables, views, and SQL packages can be granted or revoked using the SQL GRANT and

REVOKE statements. Providing levels of authority to SQL objects on the client helps ensure that only authorized users have access to an SQL application that accesses data on another system.

The level of system security in effect on the server determines whether a request from a client is accepted and whether the remote user is authorized to objects on the server.

Here are some aspects of security planning for the IBM i environment in a distributed relational database network:

- Object-related security to control user access to particular resources such as confidential tables, programs, and packages
- Location security that verifies the identity of other systems in the network
- User-related security to verify the identity and rights of users on the local system and remote systems
- Physical security such as locked doors or secured buildings that surround the systems, modems, communication lines and terminals that can be configured in the line description and used in the route selection process

Location, user-related, and object-related security are only possible if the system security level is set at level 20 or above.

For Advanced Program-to-Program Communication (APPC) conversations, when the system is using level 10 security, the IBM i operating system connects to the network as a nonsecure system. The system does not validate the identity of a remote system during session establishment and does not require conversation security on incoming program start requests. For level 10, security information configured for the APPC remote location is ignored and is not used during session or conversation establishment. If a user profile does not exist on the system, one is created.

When the system is using security level 20 or above, the IBM i operating system connects to the network as a secure system. The system can then provide conversation-level security functions and, in the case of APPC, session-level security as well.

Having system security set at the same level across the systems in your network makes the task of security administration easier. A server controls whether the session and conversation can be established by specifying what is expected from the client to establish a session. For example, if the security level on the client is set at 10 and the security level on the server is above 10, the appropriate information might not be sent and the session might not be established without changing security elements on one of the systems.

Passwords for DRDA access

The most common method of authorizing a remote user for database access is by flowing a user ID and password at connection time.

With embedded SQL one method an application programmer can use is to code the USER/USING clause on the SQL CONNECT statement. An embedded SQL example:

```
EXEC SQL CONNECT TO :rdbname USER :userid USING :pwd
```

With dynamic interfaces typically these are passed in on the connect API. A CLI example:

```
SQLConnect(rdbname, userid, pwd)
```

For Distributed Relational Database Architecture (DRDA) access to remote relational databases, once a conversation is established, you do not need to enter a password again. If you end a connection with either a RELEASE, DISCONNECT, or CONNECT statement when running with the remote unit of work (RUW) connection management method, your conversation with the first server might or might not be dropped, depending on the kind of server you are connected to and your client job attributes (for the specific rules, see Controlling DDM conversations). If the conversation to the first server is not dropped, it remains unused while you are connected to the second server. If you connect again to the first server and the conversation is unused, the conversation becomes active again without you needing to enter your user ID and password. On this second use of the conversation, your password is also not validated again.

If the local connection attempts to start a three-part naming connection, it will use the user ID and password passed in on the local connection to start the new remote connections. There are some environments, such as host server, which are not able to retrieve the password from the local connection. Therefore server authentication entries should be used for propagating passwords.

Related concepts

APPC, APPN, and HPR

Security

Related reference

Controlling DDM conversations

Normally, the DDM conversations associated with a client system job are kept active until one of the conditions described in this topic is met.

Elements of security in an APPC network

When Distributed Relational Database Architecture (DRDA) is used, the data resources of each system in the DRDA environment should be protected.

To protect data resources of each system in the DRDA environment, you can use three groups of security elements that are controlled by the following parameters:

- For system-related security or session, the LOCPWD parameter is used on each system to indicate the system validation password to be exchanged between the client and server systems when an Advanced Program-to-Program Communication (APPC) session is first established between them. Both systems must exchange the same password before the session is started. (On System/36, this password is called the location password.) In an APPC network, the LOCPWD parameter on the Create Device Description (APPC) (CRTDEVAPPC) command specifies this password. Devices are created automatically using APPN, and the location password on the remote location list specifies a password that is used by the two locations to verify identities. Use the Create Configuration List (CRTCFGL) command to create a remote location list of type (*APPNRMT).
- For user-related or location security, the SECURELOC parameter is used on each system to indicate whether it (as a server system) accepts incoming access requests that have their security already verified by the client system or whether it requires a user ID and encrypted password. In an APPC network, the SECURELOC parameter on the Create Device Description (APPC) (CRTDEVAPPC) command specifies whether the local system allows the remote system to verify security. Devices are created automatically using APPN, and the secure-location on an APPN remote Configuration List is used to determine if the local system allows the remote system to verify user security information. The SECURELOC value can be specified differently for each remote location.

  The SECURELOC parameter is used with the following security elements:

  - The user ID sent by the client system, if allowed by this parameter
  - The user ID and encrypted password, if allowed by this parameter
  - The server system user profiles, including default user profiles

  For more information, see the topic DRDA server security in an APPC network.
- For object-related security, the DDMACC parameter is used on the Change Network Attributes (CHGNETA) command to indicate whether the files on the IBM i operating system can be accessed at all by another system and, if so, at which level of security the incoming requests are to be checked.

  - If *REJECT is specified on the DDMACC parameter, all DRDA requests received by the server system are rejected.
  - If *OBJAUT is specified on the DDMACC parameter, normal object-level security is used on the server system.
  - If the name of an exit program (or access control program) is specified on the DDMACC parameter, an additional level of security is used. The user exit program can be used to control whether a given user of a specific client system can use a specific command to access (in some manner) a specific file on the server system. (See the DDM server access control exit program for additional security topic for details.)

- When a file is created on the server system using DRDA, the library name specified contains the file. If no library name is specified on the DRDA request, the current library (*CURLIB) is used. The default file authority allows only the user who created the file or the server system's security officer to access the file.

Most of the security controls for limiting remote file access are handled by the server system. Except for the user ID provided by the client system, all of these elements are specified and used on the server system. The client system, however, also limits access to server system files by controlling access to the DRDA file on the client system and by sending the user ID, when needed, to the server system.

Related concepts

DRDA server security in an APPC network
When the server system is an IBM i operating system, several elements are used together to determine whether a request to access a remote file is allowed.

Related reference

Change Network Attributes (CHGNETA) command

Create Configuration List (CRTCFGL) command

Create Device Description (APPC) (CRTDEVAPPC) command

APPN configuration lists

In an APPC network, location passwords are specified for those pairs of locations that are going to have end-to-end sessions between them.

Location passwords need not be specified for those locations that are intermediate nodes.

The remote location list is created with the **`Create Configuration List (CRTCFGL)`** command, and it contains a list of all remote locations, their location password, and whether the remote location is secure. There is one system-wide remote location configuration list on an IBM i operating system. A central site system can create location lists for remote systems by sending them a control language (CL) program.

Changes can be made to a remote configuration list using the **`Change Configuration List (CHGCFGL)`** command, however, they do not take effect until all devices for that location are all in a varied off state.

When the **`Display Configuration List (DSPCFGL)`** command is used, there is no indication that a password exists. The **`Change Configuration List (CHGCFGL)`** command indicates a password exists by placing *PASSWORD in the field if a password has been entered. There is no way to display the password. If you have problems setting up location security you might have to enter the password again on both systems to ensure that the passwords match.

Related concepts

APPC, APPN, and HPR

Related reference

Change Configuration List (CHGCFGL) command

Create Configuration List (CRTCFGL) command

Display Configuration List (DSPCFGL) command

Conversation level security

Systems Network Architecture (SNA) logical unit (LU) 6.2 architecture identifies three conversation security designations that various types of systems in an SNA network can use to provide consistent conversation security across a network of unlike systems.

The SNA security levels are:

SECURITY(NONE)
No user ID or password is sent to establish communications.

SECURITY(SAME)
Sign the user on to the remote server with the same user ID as the local server.

SECURITY(PGM)
Both a user ID and a password are sent for communications.

SECURITY(PROGRAM_STRONG)
Both a user ID and a password are sent for communications only if the password will not be sent unencrypted, otherwise an error is reported. This is not supported by DRDA on IBM i.

While the IBM i operating system supports all four SNA levels of conversation security, DRDA uses only the first three. The target controls the SNA conversation levels used for the conversation.

For the SECURITY(NONE) level, the target does not expect a user ID or password. The conversation is allowed using a default user profile on the target. Whether a default user profile can be used for the conversation depends on the value specified on the DFTUSR parameter of the Add Communications Entry (ADDCMNE) command or the Change Communications Entry (CHGCMNE) command for a given subsystem. A value of *NONE for the DFTUSR parameter means the server does not allow a conversation using a default user profile on the target. SECURITY (NONE) is sent when no password or user ID is supplied and the target has SECURELOC(*NO) specified.

For the SECURITY(SAME) level, the remote system's SECURELOC value controls what security information is sent, assuming the remote system is a IBM i product. If the SECURELOC value is *NONE, no user ID or password is sent, as if SECURITY(NONE) had been requested. If the SECURELOC value is *YES, the name of the user profile is extracted and sent along with an indication that the password has already been verified by the local system. If the SECURELOC value is *VFYENCPWD, the user profile and its associated password are sent to the remote system after the password has been encrypted to keep its value secret, so the user must have the same user profile name and password on both systems to use DRDA.

Note: SECURELOC(*VFYENCPWD) is the most secure of these three options because the most information is verified by the remote server; however, it requires that users maintain the same passwords on multiple servers, which can be a problem if users change one server but do not update their other servers at the same time.

For the SECURITY(PGM) level, the target expects both a user ID and password from the source for the conversation. The password is validated when the conversation is established and is ignored for any following uses of that conversation.

Related reference

Add Communications Entry (ADDCMNE) command

Change Communications Entry (CHGCMNE) command

DRDA server security in an APPC network

When the server system is an IBM i operating system, several elements are used together to determine whether a request to access a remote file is allowed.

User-related security elements

The user-related security elements include the SECURELOC parameter on the server system, the user ID sent by the client system (if allowed), the password for the user ID sent by the client system, and a user profile or default user profile on the server system.

Object-related security elements

The object-related security elements include the DDMACC parameter and, optionally, a user exit program supplied by the user to supplement normal object authority controls.

User-related elements of server security

A valid user profile must exist on the server to process distributed relational database work. You can specify a default user profile for a subsystem that handles communications jobs on the IBM i operating system.

The name of the default user profile is specified on the DFTUSR parameter of the Add Communications Entry (ADDCMNE) command on the server. The ADDCMNE command adds a communications entry to a subsystem description used for communications jobs.

If a default user profile is specified in a communications subsystem, whether the server is a secure location or not determines if the default user profile is used for this request. The SECURELOC parameter on the Create Device Description (APPC) (CRTDEVAPPC) command, or the secure location designation on an APPN remote location list, specifies whether the server is a secure location.

- If *YES is specified for SECURELOC or secure location on the server, the server considers the client a secure location. A user ID and an Already Verified indicator are expected from the client with its request. If a user profile exists on the server that matches the user ID sent by the client, the request is allowed. If not, the request is rejected.
- If *NO is specified for the SECURELOC parameter on the server, the server does not consider the client a secure location. Although the client still sends a user ID, the server does not use this for the request. Instead, a default user profile on the server is used for the request, if one is available. If no default user profile exists on the server, the request is rejected.
- If *VFYENCPWD is specified for SECURELOC on the server, the server considers the client a secure location, but requires that the user ID and its password be sent (in encrypted form) to verify the identity of the current user. If the user profile exists on the server that matches the user ID sent by the client, and that client has the same password on both systems, the request is allowed. Otherwise, the request is rejected.

The following table shows all of the possible combinations of the elements that control SNA SECURITY(PGM) on the IBM i operating system. A Y in any of the columns indicates that the element is present or the condition is met. An M in the PWD column indicates that the security manager retrieves the user's password and sends a protected (encrypted) password if password protection is active. If a protected password is not sent, no password is sent. A *protected password* is a character string that APPC substitutes for a user password when it starts a conversation. Protected passwords can be used only when the systems of both partners support password protection and when the password is created on a system that runs IBM i V5R3, or later, or OS/400 V2R2, or later.

Table 4. Remote access to a distributed relational database

Row	UID	PWD[1]	AVI	SEC(Y)	DFT	Valid	Access
1	Y	Y		Y	Y	Y	Use UID
2	Y	Y		Y	Y		Reject
3	Y	Y		Y		Y	Use UID
4	Y	Y		Y			Reject
5	Y	Y			Y	Y	Use UID
6	Y	Y			Y		Reject
7	Y	Y				Y	Use UID
8	Y	Y					Reject
9	Y		Y	Y	Y	Y	Use UID
10	Y		Y	Y	Y		Reject
11	Y		Y	Y		Y	Use UID
12	Y		Y	Y			Reject
13	Y	M[3]			Y	Y	Use DFT or UID[2]
14	Y	M[3]			Y		Use DFT or UID[2]

Table 4. Remote access to a distributed relational database (continued)

Row	UID	PWD[1]	AVI	SEC(Y)	DFT	Valid	Access
15	Y	M[3]				Y	Reject or UID[2]
16	Y	M[3]					Reject or UID[2]
17				Y	Y		Used DFT
18				Y			Reject
19					Y		Use DFT
20							Reject

Key:

UID
User ID sent

PWD
Password sent

AVI
Already Verified Indicator set

SEC(Y)
SECURELOC(YES) specified

DFT
Default user ID specified in communication subsystem

Valid
User ID and password are valid

Use UID
Connection made with supplied user ID

Use DFT
Connection made with default user ID

Reject
Connection not made

1. If password protection is active, a protected password is sent.
2. Use UID when password protection is active.
3. If password protection is active, the password for the user is retrieved by the security manager, and a protected password is sent; otherwise, no password is sent.

To avoid having to use default user profiles, create a user profile on the server for every client user that needs access to the distributed relational database objects. If you decide to use a default user profile, however, make sure that users are not allowed on the system without proper authorization. For example, the following command specifies the default user parameter as DFTUSER(QUSER); this allows the system to accept job start requests without a user ID or password from a communications request. The communications job is signed on using the QUSER user profile.

```
ADDCMNE   SBSD(SAMPLE) DEV(*ALL) DFTUSER(QUSER)
```

Elements of security in a TCP/IP network

DDM and DRDA over native TCP/IP does not use IBM i communications security services and concepts such as communications devices, modes, secure location attributes, and conversation security levels which are associated with Advanced Program-to-Program Communication (APPC). Therefore, security setup for TCP/IP is quite different.

Client security in a TCP/IP network

Different connectivity scenarios call for using different levels of authentication. Therefore, an administrator can set the lowest security authentication method required by the client when connecting to a server by setting the preferred authentication method field in each RDB directory entry.

The administrator might also allow the decision about the authentication method to be negotiated with the server, by choosing to allow a lower security authentication method. In this case the preferred authentication method is still attempted, but if the server cannot accept the preferred method, a lower method can be used, depending on the system security setting and other factors such as the availability of cryptographic support. For example, if two systems are in a physically unprotected environment, the administrator might choose to require Kerberos authentication without allowing lower security authentication methods.

If an encrypted server authentication method is used in the RDB directory entry, you have the option to specify *DES or *AES encryption algorithm on the **Add RDB Directory Entry (ADDRDBDIRE)** or **Change RDB Directory Entry (CHGRDBDIRE)** command. ENCALG(*DES) or ENCALG(*AES) determine what the encryption algorithm that will be requested by the client for encryption of the user ID or password. A DDM file may also take advantage of using *AES encryption by creating the DDM file with the *RDB value on RMTLOCNAME keyword of the **Create DDM File (CRTDDMF)** or **Change DDM File (CHGDDMF)** command and specifying the RDB on the RDB keyword. If you want all data to be encrypted, SSL allows all data over the network to be encrypted and supports a variety of encryption types.

These settings can be used for setting the encryption algorithm requested by the client for encrypted authentication methods:

- ENCALG(*AES)

  Advanced Encryption Standard (AES) encryption algorithm will be requested.
- ENCALG(*DES)

  Data Encryption Standard (DES) encryption algorithm will be requested. This is the default.

On the client side, you can use one of two methods to send a password along with the user ID on DRDA TCP/IP connect requests. If you do not use either of these two methods, the SQL CONNECT statement will send only a user ID for the connection attempt.

The first method to send a password is to use the USER/USING form of the SQL CONNECT statement, as in the following example from the interactive SQL environment:

```
CONNECT TO rdbname USER userid USING 'password'
```

In a program using embedded SQL, the values of the user ID and of the password can be contained in host variables in the USER/USING database.

In a program using CLI, the following example shows how the user ID and password are presented in host variables to the DRDA client:

```
SQLConnect(hdbc,sysname,SQL_NTS,                    /*do the connect to the server */
              uid,SQL_NTS,pwd,SQL_NTS);
```

A server authentication entry is a second method to define a userid & password to send via a connect request over TCP/IP. A server authentication entry list is associated with every user profile, group profile, or supplemental group profile on the system. By default, the list is empty; However, you can add entries by using the **Add Server Authentication Entry (ADDSVRAUTE)** command.

Authorization Requirement: To use the ADDSVRAUTE command, you must have the following authorizations:

- Security administrator (*SECADM) user special authority

- Object management (*OBJMGT) user special authority
- Use (*USE) authorities to the profile to which the server authentication entry is to be added or be signed on under that user profile.

To store a password using the ADDSVRAUTE command, you must set the QRETSVRSEC system value to '1'. By default, the value is '0'. Type the following command to change this value:

```
CHGSYSVAL QRETSVRSEC VALUE('1')
```

The following example shows the syntax of the ADDSVRAUTE command when using an RDB directory entry:

```
ADDSVRAUTE USRPRF(user-profile) SERVER(rdbname) USRID(userid)
PASSWORD(password)
```

The USRPRF parameter specifies the user profile under which the client job runs. What you put into the SERVER parameter is normally the name of the RDB or QDDMDRDASERVER to which you want to connect for connections made on behalf of RDB DDM files or DRDA connections. The exception is that if you are using non-RDB DDM files which were not created to use the RDB directory, you should specify QDDMDRDASERVER or QDDMSERVER in the SERVER parameter. The value given for the SERVER parameter must be in uppercase. The USRID parameter specifies the user profile under which the server job will run. The PASSWORD parameter specifies the password for the user profile.

If you omit the USRPRF parameter, it will default to the user profile under which the ADDSVRAUTE command runs. If you omit the USRID parameter, it will default to the value of the USRPRF parameter. If you omit the PASSWORD parameter, or if you have the QRETSVRSEC value set to 0, no password will be stored in the entry and when a connect attempt is made using the entry, the security mechanism attempted will be user ID only.

You can use the **Display Server Authentication Entries (DSPSVRAUTE)** command to determine what authentication entries have been added to the server authentication list. The **Retrieve Server Authentication Entries (QsyRetrieveServerEntries) (QSYRTVSE)** API in a user-written program can also be used.

You can remove a server authentication entry by using the **Remove Server Authentication Entry (RMVSVRAUTE)** command. You can change a server authentication entry by using the **Change Server Authentication Entry (CHGSVRAUTE)** command

If a server authentication entry exists for a relational database (RDB), and a user ID and password are passed in on the connection request, the user ID and password passed in take precedence over the server authentication entry.

The two special server authentication entry ADDSVRAUTE command SERVER parameter values that are supported by DRDA and DDM are:

- QDDMDRDASERVER special value on the ADDSVRAUTE command SERVER parameter for DDM & DRDA connections allows an administrator to configure a user to work with all possible DDM or DRDA connections to any system in the TCP/IP network via a common userid and password. Once configured for a specific user, no additional changes need to be made for that user as systems are added to the Relational Database Directory. As before, this does not allow a customer to connect over DRDA/DDM unless they specify a valid userid and password on the server authentication entry or CONNECT statement. This special value has all of the support that QDDMSERVER has in addition to supporting connections made by RDB DDM files and DRDA connections.
- QDDMSERVER special value on the ADDSVRAUTE command SERVER parameter for non-RDB DDM connections allows an administrator to configure a user to work with all possible non-RDB DDM connections to any system in the TCP/IP network via a common userid and password. Once configured for a specific user, no additional changes need to be made for that user as new non-RDB DDM files are created and used. The disadvantage of using QDDMSERVER is it does not support connection made with RDB DDM files or DRDA connections. It is recommended to use QDDMDRDASERVER if DDM file, RDB DDM files and DRDA connections what to take advantage of server authentication entries where a command user userid and password are needed.

When you attempt a DRDA connection over TCP/IP without specifying a userid & password, the Db2 for i client (AR) checks the server authentication list for the user profile under which the client job is running. If it finds a match between the RDB name or QDDMDRDASERVER on the CONNECT statement and the SERVER name in an authentication entry (which must be in uppercase), the associated USRID parameter in the entry is used for the connection user ID. If a PASSWORD parameter is stored in the entry, that password is also sent on the connection request.

A server authentication entry can also be used to send a password over TCP/IP for a DDM file I/O operation. When you attempt a DDM connection over TCP/IP, Db2 for i checks the server authentication list for the user profile under which the client job is running. If it finds a match between either the RDB name (if RDB DDM file is used) or QDDMSERVER (if non-RDB DDM used) and the SERVER name in an authentication entry, the associated USRID parameter in the entry is used for the connection user ID. If a PASSWORD parameter is stored in the entry, that password is also sent on the connection request.

For DRDA connection requests, if a server authentication entry specifying the system name exists, and a userid and password are not passed in on CONNECT statement, the userid and password associated with the server authentication entry will take precedence over the server authentication entry for QDDMDRDASERVER.

For DRDA connection requests, if a server authentication entry specifying the system name exists, and a user ID and password are passed in on CONNECT statement, the user ID and password associated with the CONNECT statement will take precedence over any server authentication entry.

For RDB DDM file connection requests, the server authentication entry specifying the system name will take precedence over the server authentication entry for QDDMDRDASERVER.

For non-RDB DDM file connection requests, the server authentication entry QDDMSERVER will take precedence over the server authentication entry for QDDMDRDASERVER.

Example 1: Using QDDMDRDASERVER for DRDA connections

Environment: Three systems (SYSA, SYSB, SYSC)

SYSA is the application requestor (AR). SYSB and SYSC are the application servers (AS)

On SYSA:

```
ADDSVRAUTE USRPRF(YOURPRF) SERVER(QDDMDRDASERVER) USRID(youruid) PASSWORD(yourpwd)
STRSQL
CONNECT TO SYSB
CONNECT TO SYSC
```

At this point, you have two DRDA connections made with the userid 'youruid' and password 'yourpwd' specified on the special server authentication entry value QDDMDRDASERVER.

Example 2: Using system name over QDDMDRDASERVER for DRDA connections

Environment: Three systems (SYSA, SYSB, SYSC)

SYSA is the application requestor (AR). SYSB and SYSC are the application servers (AS)

On SYSA:

```
ADDSVRAUTE USRPRF(YOURPRF) SERVER(QDDMDRDASERVER) USRID(youruid) PASSWORD(yourpwd)
ADDSVRAUTE USRPRF(YOURPRF) SERVER(SYSB) USRID(yourotheruid) PASSWORD(yourotherpwd)
ADDSVRAUTE USRPRF(YOURPRF) SERVER(SYSC) USRID(yourotheruid) PASSWORD(yourotherpwd)
STRSQL
CONNECT TO SYSB
CONNECT TO SYSC
```

At this point, you have two DRDA connections made with the userid 'yourotheruid' and password 'yourotherpwd'. This occurs because server authentication entries specifying the system name take precedence over server authentication entries specifying QDDMDRDASERVER.

Example 3: Using mix of system name and QDDMDRDASERVER for DRDA connections

Environment: Three systems (SYSA, SYSB, SYSC)

SYSA is the application requestor (AR). SYSB and SYSC are the application servers (AS)

On SYSA:

```
ADDSVRAUTE USRPRF(YOURPRF) SERVER(QDDMDRDASERVER) USRID(youruid) PASSWORD(yourpwd)
ADDSVRAUTE USRPRF(YOURPRF) SERVER(SYSB) USRID(yourotheruid) PASSWORD(yourotherpwd)
STRSQL
CONNECT TO SYSB user testuserid using 'testpassword'
CONNECT TO SYSC
```

At this point, you have two DRDA connections. The connection to SYSB would be made with userid 'testuserid' and password 'testpassword'. This occurs because specifying userid and password on a CONNECT statement takes precedence over server authentication entries. The connection to SYSC would be made with userid 'youruid' and password 'yourpwd' since it will use the QDDMDRDASERVER authentication entry when no other server authentication entry exists specifying the system name.

Example 4: Using mix of QDDMDRDASERVER and QDDMSERVER for RDB DDM and non-RDB DDM connections

Environment: Three systems (SYSA, SYSB, SYSC)

SYSA is the application requestor (AR). SYSB and SYSC are the application servers (AS)

On SYSA:

```
ADDSVRAUTE USRPRF(YOURPRF) SERVER(QDDMDRDASERVER) USRID(youruid) PASSWORD(yourpwd)
ADDSVRAUTE USRPRF(YOURPRF) SERVER(QDDMSERVER) USRID(youruid2) PASSWORD(yourpwd2)
ADDSVRAUTE USRPRF(YOURPRF) SERVER(SYSC) USRID(yourotheruid) PASSWORD(yourotherpwd)
CRTDDMF FILE(QTEMP/DDMF) RMTFILE(FILE) RMTLOCNAME(SYSB *IP)
CRTDDMF FILE(QTEMP/DDMF2) RMTFILE(FILE) RMTLOCNAME(*RDB) RDB(SYSB)
CRTDDMF FILE(QTEMP/DDMF3) RMTFILE(FILE) RMTLOCNAME(*RDB) RDB(SYSC)

SBMRMTCMD CMD('DSPLIB YOURLIB') DDMFILE(QTEMP/DDMF)
```

The connection to SYSB would be made with userid 'youruid2' and password 'yourpwd2'. This occurs because non-RDB DDM files will use QDDMSERVER for userid and password at connection time. If QDDMSERVER didn't exist, QDDMDRDASERVER would be used.

```
SBMRMTCMD CMD('DSPLIB YOURLIB') DDMFILE(QTEMP/DDMF2)
```

The connection to SYSB would be made with userid 'youruid' and password 'yourpwd'. This occurs because RDB DDM files will use QDDMDRDASERVER for userid and password at connection time if a server authentication entry doesn't exist specifying the system name.

```
SBMRMTCMD CMD('DSPLIB YOURLIB') DDMFILE(QTEMP/DDMF3)
```

The connection to SYSC would be made with userid 'yourotheruid' and password 'yourotherpwd'. This occurs because RDB DDM files will not use QDDMDRDASERVER for userid and password at connection time if a server authentication entry exists specifying the system name.

Simplified DDM and DRDA server authentication entry management using group profiles

DDM and DRDA can now take advantage of a common userid and password defined in a server authentication entry under a group profile name or supplemental group profile name. The group profile name or supplemental group profile name is specified on the USRPRF parameter of the **`Add Server Authentication Entry (ADDSVRAUTE)`** command. Prior to this enhancement, if server authentication entries were to be used to access an application server that required userid and password authentication, a server authentication entry had to be in place for every user on the system requiring access.

System administration of server authentication entries becomes much simpler when a group profile is used. A common user ID and password profile is used when establishing a group profile based DDM or DRDA connection. Remote connection capability is managed by controlling the users that belong to the group together with their authority to the group profile.

Example:

```
CRTUSRPRF USRPRF(TESTTEAM) PASSWORD(*NONE)
CRTUSRPRF USRPRF(JIM) PASSWORD(yourpasswordA) GRPPRF(TESTTEAM)
ADDSVRAUTE USRPRF(TESTTEAM) SERVER(QDDMDRDASERVER or <RDB-name of system-B>) USRID(youruseridB)
PASSWORD(yourpasswordB)

Signon to <system-A> with Jim's user profile
STRSQL
CONNECT TO <system-B>
(where <system-B> is a RDB entry defined in the Work with Relational Database Directory Entries
(WRKRDBDIRE) command.)

A connection is attempted to <system-B> with userid = youruseridB and password = yourpasswordB.
```

Note: A group member's server authentication entry takes precedence over a group profile's server authentication entry as shown in the order or progression below. The progression of DDM & DRDA server authentication entry checking occurs in the following order. The connection is attempted with the settings of the first match found.

1. Search the authentication entries where USRPRF=user profile and SERVER= application server name.
2. Search the authentication entries where USRPRF=user profile and SERVER='QDDMDRDASERVER'.
3. Search the authentication entries where USRPRF=group profile and SERVER= application server name.
4. Search the authentication entries where USRPRF=group profile and SERVER='QDDMDRDASERVER'.
5. Search the authentication entries where USRPRF=supplemental group profile and SERVER= application server name.
6. Search the authentication entries where USRPRF=supplemental group profile and SERVER='QDDMDRDASERVER'.
7. If no entry has been found in all previous steps, a USERID-only authentication attempt will be used.

DRDA explicit server name priority when using group profile server authentication entries

By default, group members do not use a group or supplemental group's server authentication entries that explicitly specify an application server name when connecting over DRDA/DDM if that group member uses QDDMDRDASERVER special server authentication entry. This occurs because, QDDMDRDASERVER at the group member is given higher priority over the explicitly named server authentication entries at the group or supplemental group. An environment variable, QIBM_DDMDRDA_SVRNAM_PRIORITY, has been added that allows group or supplemental group's server authentication entries that explicitly specify an application server name to be given higher priority over any use of QDDMDRDASERVER special server authentication entry at the group member, group or supplemental group profile when attempting a DDM/ DRDA connection. The benefit of using the environment variable is to allow group or supplemental group's server authentication entries explicitly specifying an application server name to still be used if a group member specifies QDDMDRDASERVER.

- To enable explicit server name priority at the system-level:

  ```
  ADDENVVAR ENVVAR(QIBM_DDMDRDA_SVRNAM_PRIORITY) VALUE('Y') LEVEL(*SYS)
  ```

  Restarting the job maybe necessary to pick up the change.
- To enable explicit server name priority at the job-level:

  ```
  ADDENVVAR ENVVAR(QIBM_DDMDRDA_SVRNAM_PRIORITY) VALUE('Y') LEVEL(*JOB)
  ```

Example:

```
ADDENVVAR ENVVAR(QIBM_DDMDRDA_SVRNAM_PRIORITY) VALUE('Y') LEVEL(*JOB)
CRTUSRPRF USRPRF(TESTTEAM) PASSWORD(*NONE)
CRTUSRPRF USRPRF(JIM) PASSWORD(yourpasswordA) GRPPRF(TESTTEAM)
ADDSVRAUTE USRPRF(JIM) SERVER(QDDMDRDASERVER) USRID(youruseridA) PASSWORD(yourpasswordA)
ADDSVRAUTE USRPRF(TESTTEAM) SERVER(<RDB-name for system-B>) USRID(youruseridB)
PASSWORD(yourpasswordB)

Signon to <system-A> with Jim's user profile
STRSQL
CONNECT TO <system-B>
(where <system-B> is a RDB entry defined in the Work with Relational Database Directory Entries
(WRKRDBDIRE) command.)

A connection is attempted to <system-B> with userid = youruseridB and password = yourpasswordB.
A connection made to any other system will use userid = youruseridA and password =
yourpasswordA.
```

Note: A group member's server authentication entry takes precedence over a group profile's server authentication entry as shown in the order or progression below. The progression of DDM & DRDA server authentication entry checking occurs in the following order when the environment variable is set. The connection is attempted with the settings of the first match found.

1. Search the authentication entries where USRPRF=user profile and SERVER= application server name.
2. Search the authentication entries where USRPRF=group profile and SERVER= application server name.
3. Search the authentication entries where USRPRF=supplemental group profile and SERVER= application server name.
4. Search the authentication entries where USRPRF=user profile and SERVER='QDDMDRDASERVER'.
5. Search the authentication entries where USRPRF=group profile and SERVER='QDDMDRDASERVER'.
6. Search the authentication entries where USRPRF=supplemental group profile and SERVER='QDDMDRDASERVER'.
7. If no entry has been found in all previous steps, a USERID-only authentication attempt will be used.

Enable DRDA and DDM authentication using user profile's password

This improvement has been officially called DRDA/DDM Conjoined Mutual Authentication and is only supported for IBM i to IBM i connections and enabled through a environmental variable QIBM_CONJOINED_MUT_AUTH.

Prior to this enhancement, users attempting a Distributed Relational Database Architecture (DRDA) or Distributed Data Management (DDM) connection needed to specify a password on the SQL CONNECT statement or specify a server authentication entry containing a password if the application server was configured to required a password. This required users or system administrators to spend time managing server authentication entries for all target systems or specify a password at CONNECT time when connecting remotely. If a password was not provided on the SQL CONNECT statement or not specified in a server authentication entry for the application server, the DRDA/DDM connection to the application server requiring a password would fail.

With this enhancement enabled, a IBM i to IBM i DRDA or DDM connection attempt will be made to the application server using the currently signed in user profile's userid & password. This connection attempt is only done if the environment variable is set to a value of Y, and the user has not explicitly specified a userid and password on the SQL CONNECT statement or in a server authentication entry for the application server configured to require a password.

This enhancement in beneficial in networks where users have the same userid and password combination across multiple systems. This enhancement allows users to not have to specify server authentication entries for 3-part connections.

Requirements:

Both the application requestor and application server must be anIBM i at v7r2m0 for it to be supported. This enhancement is enabled when the application requester's job adds an environment variable QIBM_CONJOINED_MUT_AUTH with value of 'Y' (see the **Add Environment Variable (ADDENVVAR)** command to add the environment variable). The application server must be set to a require a password (see the **Change DDM TCP/IP Attributes (CHGDDMTCPA)** command PWDRQD keyword). Both systems must have the same userid and password defined for it to connect. Both systems must have the same password level specified in the system value QPWDLVL. The user must not explicitly specify a userid and password on the SQL CONNECT statement or in a server authentication entry for the application server configured to require a password.

- To enable conjoined mutual authentication at the system-level:

  ```
  ADDENVVAR ENVVAR(QIBM_CONJOINED_MUT_AUTH) VALUE('Y') LEVEL(*SYS)
  ```

  Restarting the job maybe necessary to pick up the change.
- To enable conjoined mutual authentication at the job-level:

  ```
  ADDENVVAR ENVVAR(QIBM_CONJOINED_MUT_AUTH) VALUE('Y') LEVEL(*JOB)
  ```

Note: An IBM i to IBM i DRDA or DDM connection, where a password or a server authentication entry is not specified by the application requestor and the support is enabled, will result in an additional connection attempt to be made to the application server requiring a password to connect. This additional connection attempt will be made using an encrypted userid and password security mechanism where the userid is from the user profile making the connection, and the password is retrieved from that user profile. If it is determined that the passwords match, the connection is allowed as normal. If the passwords do not match, there is a security failure, or a requirement above is not met, a CPF22E2 message will be signaled, and a PW audit record is written. This invalid connection attempt is counted as one invalid signon attempt for the user profile.

Security audit records are written to the QSYS/QAUDJRN security audit journal when auditing is enabled and the auditing level is configured to authorization failures. A PW audit record appears when the userid or password fails to connect.

Kerberos source configuration

Distributed Relational Database Architecture (DRDA) and Distributed Data Management (DDM) can take advantage of Kerberos authentication if both systems are configured for Kerberos.

If a job's user profile has a valid ticket-granting ticket (TGT), the DRDA client uses this TGT to generate a service ticket and authenticate the user to the remote system. Having a valid TGT makes the need for a server authentication entry unnecessary, because no password is directly needed in that case. However, if the job's user profile does not have a valid TGT, the user ID and password can be retrieved from the server authentication entry to generate the necessary TGT and service ticket.

When using Kerberos, the remote location (RMTLOCNAME) in the RDB directory entry must be entered as the remote host name. IP addresses will not work for Kerberos authentication.

In cases where the Kerberos realm name differs from the DNS suffix name, it must be mapped to the correct realm. To do that, there must be an entry in the Kerberos configuration file (krb5.conf) to map each remote host name to its correct realm name. This host name entered must exactly match the remote location name (RMTLOCNAME). The remote location parameter displayed by the **Display Relational Database Directory Entries (DSPRDBDIRE)** or **Display DDM File (DSPDDMF)** command must match the domain name in the krb5.conf file. The following figure shows an example of the **Display Relational Database Directory Entries (DSPRDBDIRE)** command display:

```
                      Display Relational Database Detail

Relational database  . . . . . . :   RCHASXXX
Remote location:
  Remote location  . . . . . . . :   rchasxxx.rchland.ibm.com
    Type . . . . . . . . . . . . :   *IP
  Port number or service name  . :   *DRDA
  Remote authentication method:
    Preferred method . . . . . . :   *KERBEROS
    Allow lower authentication . :   *NOALWLOWER
  Secure connection  . . . . . . :   *NONE
  Encryption algorithm . . . . . :   *DES
Text . . . . . . . . . . . . . . :


Relational database type . . . . :   *REMOTE




  Press Enter to continue.
   F3=Exit   F12=Cancel
```

Here is a portion of the corresponding krb5.conf file contents showing the domain name matching the remote location name (Note: The **Display File (DSPF)** command is used to display the configuration file contents):

```
DSPF STMF('/QIBM/UserData/OS400/NetworkAuthentication/krb5.conf')

[domain_realm]
; Convert host names to realm names. Individual host names may be
; specified. Domain suffixes may be specified with a leading period
; and will apply to all host names ending in that suffix.
 rchasxxx.rchland.ibm.com = REALM.RCHLAND.IBM.COM
```

Jobs using Kerberos must be restarted when configuration changes occur to the krb5.conf file.

Related concepts

Enterprise Identity Mapping (EIM)

Related tasks

Setting up the TCP/IP server
If you own a Distributed Relational Database Architecture (DRDA) server that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Configuring network authentication service

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Display Server Authentication Entries (DSPSVRAUTE) command

Retrieve Server Authentication Entries (QsyRetrieveServerEntries) (QSYRTVSE) API

Remove Server Authentication Entry (RMVSVRAUTE) command

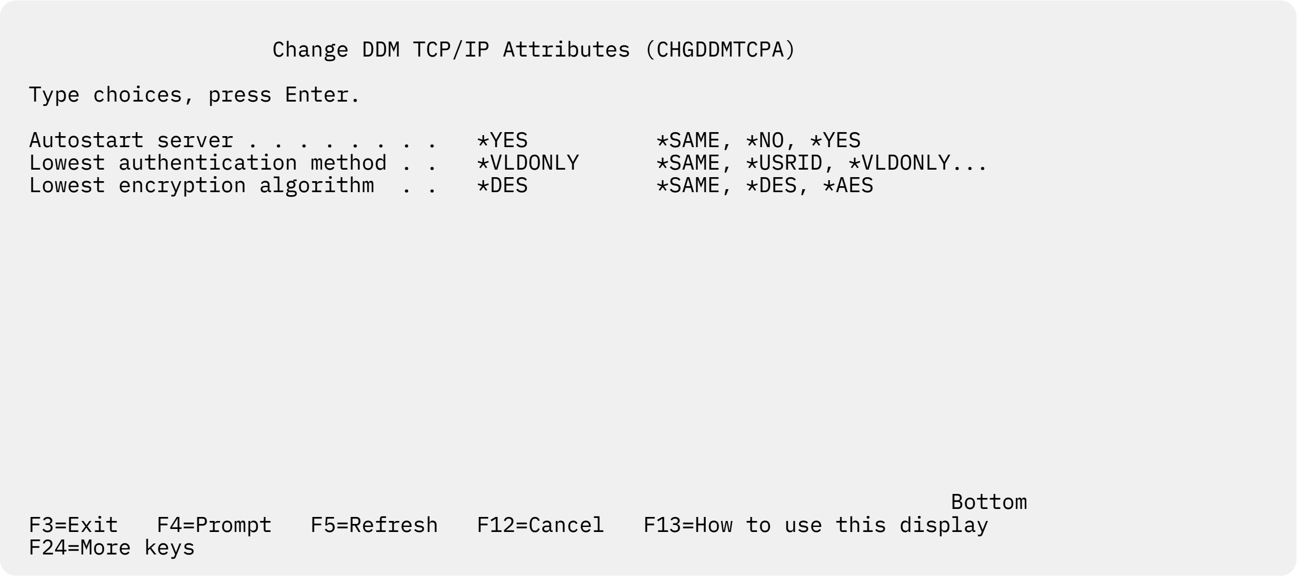
Change Server Authentication Entry (CHGSVRAUTE) command

Display File (DSPF) command

Server security in a TCP/IP network

The TCP/IP server has a default security of user ID with clear-text password. This means that, as the server is installed, inbound TCP/IP connection requests must have at least a clear-text password accompanying the user ID under which the server job is to run.

The security can either be changed with the **Change DDM TCP/IP Attributes (CHGDDMTCPA)** command or under the **Network** > **Servers** > **TCP/IP** > **DDM server properties** in System i® Navigator. You must have *IOSYSCFG special authority to change this setting.



```
                   Change DDM TCP/IP Attributes (CHGDDMTCPA)

 Type choices, press Enter.

 Autostart server . . . . . . . .   *YES          *SAME, *NO, *YES
 Lowest authentication method . .   *VLDONLY      *SAME, *USRID, *VLDONLY...
 Lowest encryption algorithm  . .   *DES          *SAME, *DES, *AES

















                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

These settings can be used for setting the lowest authentication method allowed by the server:

- PWDRQD (*USRID)

  Password is not required.
- PWDRQD(*VLDONLY)

  Password is not required, but must be valid if sent.

The difference between *USRID and *VLDONLY is that if a password is sent from a client system, it is ignored in the *USRID option. In the *VLDONLY option, however, if a password is sent, the password is validated for the accompanying user ID, and access is denied if incorrect.

Encrypted user ID and password (or PWDRQD(*ENCUSRPWD)), encrypted password required (or PWDRQD(*USRENCPWD)), and Kerberos (or PWDRQD(*KERBEROS)) can be used for higher security levels. If Kerberos is used, user profiles must be mapped to Kerberos principles using Enterprise Identity Mapping (EIM).

The following example shows the use of the **Change DDM TCP/IP Attributes (CHGDDMTCPA)** command to specify that an encrypted password must accompany the user ID. To set this option, enter:

```
CHGDDMTCPA PWDRQD(*USRENCPWD)
```

These settings can be used for setting the lowest encryption algorithm allowed by the server for encrypted authentication methods:

- ENCALG(*AES)

  Advanced Encryption Standard (AES) encryption algorithm only will be allowed.
- ENCALG(*DES)

  Data Encryption Standard (DES) encryption algorithm or higher strength encryption algorithm will be allowed.

The following example shows the use of the **Change DDM TCP/IP Attributes (CHGDDMTCPA)** command to specify that AES encryption must be used to authenticate to the server with encrypted authentication methods. To set this option, enter:

```
CHGDDMTCPA ENCALG(*AES)
```

ENCALG(*DES) or ENCALG(*AES) determine what the lowest encryption algorithm level that will be supported by the server for encryption of the user ID or password. If you want all data to be encrypted, SSL allows all data over the network to be encrypted and supports a variety of encryption types.

Note: The DDM and DRDA TCP/IP server supports the following forms of password encryption:

- Password substitute algorithm
- Diffie-Hellman public key algorithm (56 bit *DES or 256 bit *AES)
- A strong password substitute algorithm

The client and server negotiate the security mechanism that will be used, and any of the three encryption methods will satisfy the requirement of encrypted password (PWDRQD(*USRENCPWD)), as does the use of Secure Sockets Layer (SSL) data streams.

Related concepts

Enterprise Identity Mapping

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Connection security protocols

Several connection security protocols are supported by the current Db2 for i implementation of distributed data management (DDM) or Distributed Relational Database Architecture (DRDA) over TCP/IP.

- User ID only
- User ID with clear-text password
- User ID with encrypted password
- Encrypted user ID with encrypted password.
- Kerberos

With encrypted datastream support, the traditional communications trace support has little value. The **`Trace TCP/IP Application (TRCTCPAPP)`** command records outbound data streams at a point before encryption, and inbound data streams after decryption.

Related concepts

Communications trace

If you get a message in the CPF3E*xx* range or the CPF91*xx* range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Related reference

Trace TCP/IP Application (TRCTCPAPP) command

Secure Sockets Layer

Db2 for i Distributed Relational Database Architecture (DRDA) clients & servers support Secure Sockets Layer (SSL). A similar function is available with Internet Protocol Security Architecture (IPSec).

The DDM TCP/IP client and server support the SSL data encryption protocol. You can use this protocol to interoperate with other clients and servers that support SSL for record-level access, such as IBM Toolbox for Java and IBM i Access Family OLE DB Provider. Using this protocol, you can also interoperate with any DDM file I/O clients provided by independent software vendors that might support SSL.

To use SSL with the IBM i DDM TCP/IP server, you must configure the client to connect to SSL port 448 on the server.

To use SSL as a DDM TCP/IP client, you must specify SECCNN(*SSL) on the Add RDB Directory Entry (ADDRDBDIRE) command or Change RDB Directory Entry (CHGRDBDIRE) command.

If you specify PWDRQD(*USRENCPWD) on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command, you can use any valid password along with SSL. This is possible because the system recognizes that the whole data stream, including the password, is encrypted.

Related concepts

Internet Protocol Security Architecture

Internet Protocol Security Architecture (IPSec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

Secure Sockets Layer (SSL)

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Server is not started or the port ID is not valid
The error message given if the DDM TCP/IP server is not started is CPE3425.

Required programs
You will need to set up and install SSL support.

Related concepts

IBM i Access for Windows: Programming

IBM i requirements
For a IBM i product to communicate over Secure Sockets Layer (SSL) server, it must be running IBM i V5R3, or later, or OS/400 V4R4, or later, and have the following applications installed.

- TCP/IP Connectivity Utilities for IBM i, 5761-TC1 (Base TCP/IP support), see TCP/IP setup
- IBM HTTP Server for IBM i, 5761-DG1 (for access to Digital Certificate Manager), see IBM HTTP Server for IBM i
- Digital Certificate Manager, 5761-SS1 (Option 34), see Digital Certificate Manager

For a IBM i product to communicate over Secure Sockets Layer (SSL) as a client, it must be running IBM i V5R3, or later, and have the following applications and ptf's installed. IBM i 7.1 and beyond do not require a ptf for this functionality.

- TCP/IP Connectivity Utilities for IBM i, 5761-TC1 (Base TCP/IP support), see TCP/IP setup
- IBM HTTP Server for IBM i, 5761-DG1 (for access to Digital Certificate Manager), see IBM HTTP Server for IBM i
- Digital Certificate Manager, 5761-SS1 (Option 34), see Digital Certificate Manager
- V6R1M0 PTF: SI33570 (English only)
- V5R4M0 PTF: SI33515 (English only)
- V5R3M0 PTF: SI33470 (English only)

Internet Protocol Security Architecture
Internet Protocol Security Architecture (IPSec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

On the IBM i operating system, IPSec, a component of the virtual private networking (VPN) support, allows all data between two IP addresses or port combinations to be encrypted, regardless of application (such as DRDA or DDM). You can configure the addresses and ports that are used for IPSec. IBM suggests using port 447 for IPSec for either DRDA access or DDM access.

Use of any valid password along with IPSec does not in general satisfy the requirement imposed by specifying PWDRQD(*USRENCPWD) on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command, because the application (DRDA or DDM) is not able to determine if IPSec is being used. Therefore, you should avoid using PWDRQD(*USRENCPWD) with IPSec.

Related concepts

Secure Sockets Layer
Db2 for i Distributed Relational Database Architecture (DRDA) clients & servers support Secure Sockets Layer (SSL). A similar function is available with Internet Protocol Security Architecture (IPSec).

Virtual Private Networking (VPN)

Considerations for certain passwords being sent as clear text

Although the IBM i operating system supports the encryption of connection passwords, one of the connection security options you can specify in setting up an RDB directory entry is *USRIDPWD.

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Considerations for certain passwords being sent as clear text

Although the IBM i operating system supports the encryption of connection passwords, one of the connection security options you can specify in setting up an RDB directory entry is *USRIDPWD.

See the Add Relational Database Directory Entry (ADDRDBDIRE) command and the Change Relational Database Directory Entry (CHGRDBDIRE) command in Working with the relational database directory for more information.

If the system to which the connection is made allows the *USRIDPWD security option, the connection password can flow unencrypted. The SQL SET ENCRYPTION PASSWORD statement and the ENCRYPT function can also cause passwords to flow over the network unencrypted. Currently, there are two possible solutions for encrypting data streams. One is to use IPSec. As the other possibility, if you are using a client that supports SSL, you can use that protocol to encrypt data transmitted to and from an IBM i server system.

Related concepts

Internet Protocol Security Architecture
Internet Protocol Security Architecture (IPSec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

Related reference

Working with the relational database directory
Use these instructions to work with the relational database directory.

SET ENCRYPTION PASSWORD statement

Ports and port restrictions

With the advent of new choices for the security of distributed data management (DDM) communications, the system administrator can restrict certain communications modes by blocking the ports they use. This topic discusses some of these considerations.

The DDM or DRDA TCP/IP server listens on port 446 (the well-known DRDA port), port 447 (the well-known DDM port), and port 448 (the well-known SSL port). The Db2 for i implementation of DDM does not distinguish between the two ports 446 and 447, however, so both DDM and DRDA access can be done on either port.

Using the convention recommended for IPSec, the port usage for the DDM TCP/IP server follows:

- 446 for clear text data streams
- 447 for IPSec encrypted data streams (suggested)
- 448 for SSL encrypted data streams (required)

You can block usage of one or more ports at the server by using the **`Configure TCP/IP (CFGTCP)`** command. To do this, choose the `Work with TCP/IP port restrictions` option of that command. You can add a restriction so that only a specific user profile other than the one that QRWTLSTN runs under (normally QUSER) can use a certain port, such as 446. That effectively blocks 446. If 447 were configured for use only with IPSec, then blocking 446 would allow only encrypted data streams to be used for DDM and DRDA access over native TCP/IP. You could block both 446 and 447 to restrict usage only to SSL. It might be impractical to follow these examples for performance or other reasons (such as current limited availability of SSL-capable clients), but they are given to show the possible configurations.

Related reference

Configure TCP/IP (CFGTCP) command

DRDA and DDM server access control using user exit programs

A security feature of the Distributed Relational Database Architecture (DRDA) server, for use with both Advanced Program-to-Program Communication (APPC) and TCP/IP, extends the use of the DDMACC parameter of the **`Change Network Attributes (CHGNETA)`** command to DRDA.

The parameter previously applied only to DDM file I/O access. The DRDA usage of the function is limited to connection requests, however, and not to requests for data after the connection is made.

If you do not choose to use this security function, you normally do not need to do anything. The only exception is if you are currently using a DDM user exit program that is coded to reject operations if an unknown function code is received, and you are also using DRDA to access data on that system. In this case, you must change your user exit program so that a '1' is returned to allow DRDA access if the function code is 'SQLCNN '.

To use the user exit program for blocking or filtering DRDA connections, you need to create a new DRDA user exit program, or change an existing one.

Note: If your system is configured with multiple databases (ASP groups), the user exit program must reside in a library in the system database (on an auxiliary storage pool in the range 1-32).

You can find general instructions for creating a DRDA user exit program below.

This security feature adds a DRDA function code to the list of request functions that can be input to the program in the input parameter structure. The function code, named 'SQLCNN ' (SQL connect request), indicates that a DRDA connection request is being processed (see the FUNC parameter in Example: DRDA and DDM server access control using user exit program). The APP (application) input parameter is set to `'*DRDA '` instead of `'*DDM '` for DRDA connection request calls.

When you code user exit programs for DRDA, the following fields in the parameter structure might be useful:

- The USER field allows the program to allow or deny DRDA access based on the user profile ID.
- The RDBNAME field contains the name of the RDB to which the user wants to connect. This can be the system database or a user database (ASP group). This field can be useful if you want to deny access to one or more databases in an environment where multiple databases are configured.
- The SRVNAME parameter in the topic Example: DRDA and DDM server access control using user exit program might or might not be set by the caller of the user exit program. If it is set, it indicates the name of the client system. If it is not set, it has the value *N. It is always set when the DRDA application requester is a IBM i product.
- The TYPDEFN parameter gives additional information about the type of client that is connecting. For an IBM mainframe, TYPEDEFN is QTDSQL370. For a IBM i product, it is QTDSQL400. For an Intel PC, it is QTDSQLX86. For an RS/6000® client, it is QTDSQLASC.
- The PRDID (product ID) parameter identifies the product that is attempting to connect, along with the product's release level. Here is a partial list of the first three characters of these codes (You should verify the non-IBM codes before you use them in user exit programs):

  QSQ
  IBM Db2 for i

  DSN
  IBM DB2 for z/OS

  SQL
  IBM DB2 for Linux, UNIX, and Windows (formerly called DDCS)

  ARI
  IBM DB2 for VSE & VM

  GTW
  Oracle Corporation products

  GVW
  Grandview DB/DC Systems products

XDB
XDB Systems products

IFX
Informix® Software products

SIG
StarQuest products

STH
FileTek products

JCC
IBM DB2 Universal Driver for SQLJ and JDBC

The rest of the field is structured as `vvrrm`, where `vv` is version, `rr` is release, and `m` is modification level.

If the user exit program returns a RTNCODE value of 0, and the connection request came from an IBM i client, then the message indicating the connection failure to the user will be SQ30060, `User is not authorized to relational database ...` In general, the response to a denial of access by the user exit program is the DRDA RDBATHRM reply message, which indicates that the user is not authorized to the relational database. Different client platforms might report the error differently to the user.

Restrictions:

- If a function check occurs in the user exit program, the program returns the same reply message, and the connection attempt will fail. The user exit program must not do any committable updates to Db2 for i, or unpredictable results might occur.
- You should not use user exit programs to attempt to access a file that was opened in a prior call of the prestart server job.
- Prior to V5R2, a further restriction resulted when the prestart jobs used with the TCP/IP server were recycled for subsequent use. Some cleanup is done to prepare the job for its next use. Part of this processing involves using the **Reclaim Activation Group (RCLACTGRP)** command with the ACTGRP parameter with value of *ELIGIBLE. As a result, attempts to use any residual linkages in the prestart server job to activation groups destroyed by the **RCLACTGRP** can result in MCH3402 exceptions (where the program tried to refer to all or part of an object that no longer exists). One circumvention to this restriction is to set the MAXUSE value for the QRWTSRVR prestart jobs to 1 as follows: CHGPJE SBSD(QSYSWRK) PGM(QRWTSRVR) MAXUSE(1).

Related concepts

Object-related security
If the IBM i product is a server system, there are two object-related levels at which security can be enforced to control access to its relational database tables.

Related reference

Reclaim Activation Group (RCLACTGRP) command

Change Network Attributes (CHGNETA) command

Server access control exit program parameter list

The user exit program on the target server passes two parameter values: a character return code field and a character data structure containing various parameter values.

The user exit program on the target server uses the character data structure parameter values, that are passed by the TDDM, to evaluate whether to allow the request from the source server. The parameter list is created each time a file access request or command request is sent to the TDDM; when any one of the functions shown for the *Subapplication* field is requested, the parameter list is created. When file I/O operations are performed, this parameter list is created only for the file open request, not for any of the I/O operation requests that follow.

The program uses the parameter list to determine whether a source server user's file access or command request should be accepted or rejected. The list contains the following parameters and values:

- The name of the user profile or default user profile under which the source server user's request is run.
- The name of the application program on the source server being used. For DDM use, the name is *DDM. For DRDA use, the name is *DRDA.
- The name of the command or function (subapplication) being requested for use on the target server or one of its files.

  Most of the functions listed in the following table directly affect a file, including the EXTRACT function, which extracts information from the file when commands such as **`Display File Description (DSPFD)`** or **`Display File Field Description (DSPFFD)`** are specified by the source server user. Some functions are member-related functions, such as the CHGMBR function, which allows characteristics of a member to be changed. The COMMAND function indicates that a command string is submitted by the **`Submit Remote Command (SBMRMTCMD)`** command to run on the target server. The SQLCNN function specifies a DRDA connect attempt.
- The name of the file (object) to be accessed in the way specified on the previous parameter. This field does not apply if a command string (COMMAND) or stream and directory access commands are being submitted or if it is a DRDA command.
- If the stream and directory access commands are specified, then the object and directory fields have a value of *SPC. The user must go to the *Other* field to get the alternative object name and alternative path name.
- The name of the library containing the file, if a file is being accessed.
- The name of the file member, if a file member is being accessed. Stream and access commands have a value of *N.
- The format field does not apply for DDM or DRDA.
- Depending on how the next field is used, the length varies.
- The *Other* field is used for as many as three of the following six values; the first two are always specified (*N might be used for the second value if the system name cannot be determined), and either of the last four might be specified, depending on the type of function specified in the *Subapplication* field.
  - The location name of the source server. This matches the RMTLOCNAME parameter value specified in the target server's device description for the source server if APPC communications is being used.
  - The system name of the source server.
  - If a file was specified and it is to be opened, (OPEN) for I/O operations, this field indicates which type of operation is being requested. For example, if a file is being opened for read operations only, the input request value is set to a 1 and the remaining values are set to a 0.
  - The alternative object name.
  - The alternative directory name.
  - The name of the iSeries command, if a command string is being submitted, followed by all of its submitted parameters and values.

Table 5. Parameter list for user exit program on target server

Field	Type	Length	Description
User	**Character**	**10**	**User profile name of target DDM job.**
Application	**Character**	**10**	**Application name:** • `'*DDM ' for Distributed Data Management.` • `'*DRDA ' for Distributed Relational Database Architecture`

Table 5. Parameter list for user exit program on target server (continued)

Field	Type	Length	Description
Subapplication	**Character**	**10**	**Requested function:** • 'ADDMBR ' 'DELETE ' 'RGZMBR ' • 'CHANGE ' 'EXTRACT ' 'RMVMBR ' • 'CHGDTAARA ' 'INITIALIZE' 'RNMMBR ' • 'CHGMBR ' 'LOAD ' 'RTVDTAARA ' • 'CLEAR ' 'LOCK ' 'SNDDTAQ ' • 'CLRDTAQ ' 'MOVE ' 'SQLCNN ' • 'COMMAND ' 'OPEN ' • 'COPY ' 'RCVDTAQ ' • 'CREATE ' 'RENAME '
Object	**Character**	**10**	**Specified file name. *N is used when the subapplication field is 'COMMAND '. *SPC is used when the file is a document or folder.**
	Character	**10**	**Specified library name. *N is used when the subapplication field is 'COMMAND '. *SPC is used when the library is a folder.**
Member	**Character**	**10**	**Specified member name. *N is used when the member name is not applicable.**
Format	**Character**	**10**	**Not applicable for DDM.**
Length	**Decimal**	**5,0**	**Length of the next field.**
Source Remote Location	**Character**	**10**	**Remote location unit name of source system (if SNA).**
Source System Name	**Character**	**10**	**System name of remote server. If this value is not available, this field contains '*N '.**

Table 5. Parameter list for user exit program on target server (continued)

Field	Type	Length	Description
Other	**Character**	**2000**	The use of this 2000 byte area depends on the request function. If it is SQLCNN, then the DRDA mapping should be used. For other functions, use the DDM mapping. **To use DDM:** The following varies, depending on the function. If OPEN is specified to open a file: **1** Input request Char(1) 1=yes 0=no **1** Output request Char(1) 1=yes 0=no **1** Update request Char(1) 1=yes 0=no **1** Delete request Char(1) 1=yes 0=no **12** Alternative object name. **63** Alternative directory name. **1921** The command string if COMMAND is specified to submit a command. **To use DRDA:** **9** Type definition name of DRDA application requester. Product ID of DRDA application requester. **3** Product code. **2** Version ID. **2** Release ID. **1** Modification level. **18** RDB Name. **1965** Reserved.
Note: *N = Null value indicates a parameter position for which no value is being specified, allowing other parameters to follow it in positional form.			

Example: DRDA and DDM server access control using user exit program

This user exit program shows an example of a PL/I user exit program that allows all DRDA operations and all DRDA connections except when the user ID is ALIEN.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 362.

```
/**********************************************************************/
/*                                                                    */
/* PROGRAM NAME: UEPALIEN                                             */
/*                                                                    */
/* FUNCTION:     USER EXIT PROGRAM THAT IS DESIGNED TO                */
/*               RETURN AN UNSUCCESSFUL RETURN CODE WHEN              */
/*               USERID 'ALIEN' ATTEMPTS A DRDA CONNECTION.           */
/*               IT ALLOWS ALL TYPES OF DDM OPERATIONS.               */
/*                                                                    */
/* EXECUTION:    CALLED WHEN ESTABLISHED AS THE USER EXIT             */
/*               PROGRAM.                                             */
/*                                                                    */
/* ALL PARAMETER VARIABLES ARE PASSED IN EXCEPT:                      */
/*                                                                    */
/* RTNCODE -    USER EXIT RETURN CODE ON WHETHER FUNCTION IS          */
/*              ALLOWED: '1' INDICATES SUCCESS; '0' FAILURE.          */
/*                                                                    */
/**********************************************************************/

UEPALIEN: PROCEDURE (RTNCODE,CHARFLD);

DECLARE RTNCODE CHAR(1);           /* DECLARATION OF THE EXIT         */
                                   /* PROGRAM RETURN CODE.  IT        */
                                   /* INFORMS REQUEST HANDLER         */
                                   /* WHETHER REQUEST IS ALLOWED.     */
DECLARE                            /* DECLARATION OF THE CHAR         */
 1  CHARFLD,                       /* FIELD PASSED IN ON THE CALL.    */

 2  USER     CHAR(10),             /* USER PROFILE OF DDM/DRDA USER   */
 2  APP      CHAR(10),             /* APPLICATION NAME                */
 2  FUNC     CHAR(10),             /* REQUESTED FUNCTION              */
 2  OBJECT   CHAR(10),             /* FILE NAME                       */
 2  DIRECT   CHAR(10),             /* LIBRARY NAME                    */
 2  MEMBER   CHAR(10),             /* MEMBER NAME                     */
 2  RESERVED CHAR(10),             /* RESERVED FIELD                  */
 2  LNGTH    PIC '99999',          /* LENGTH OF USED SPACE IN REST    */
 2  REST,                          /* REST OF SPACE = CHAR(2000)      */

 3  LUNAME  CHAR(10),              /* REMOTE LU NAME (IF SNA)         */
 3  SRVNAME CHAR(10),              /* REMOTE SERVER NAME              */
 3  TYPDEFN CHAR(9),               /* TYPE DEF NAME OF DRDA AR        */
 3  PRDID,                         /* PRODUCT ID OF DRDA AR           */

 5 PRODUCT  CHAR(3),               /* PRODUCT CODE                    */
 5 VERSION  CHAR(2),               /* VERSION ID                      */
 5 RELEASE  CHAR(2),               /* RELEASE ID                      */
 5 MOD      CHAR(1),               /* MODIFICATION LEVEL              */
 5 RDBNAME CHAR(18),               /* RDB NAME                        */
 5 REMAING CHAR(1965),             /* REMAINING VARIABLE SPACE        */

START:
IF (USER = 'ALIEN' &         /* IF USER IS 'ALIEN' AND                */
      FUNC = 'SQLCNN') THEN        /*   FUNCTION IS DRDA CONNECT      */
  RTNCODE = '0';                   /* SET RETURN CODE TO UNSUCCESSFUL */
ELSE                               /* IF ANY OTHER USER, OR DDM       */
  RTNCODE = '1';                   /* SET RETURN CODE TO SUCCESSFUL   */

END UEPALIEN;
```

Figure 22. Example PL/I user exit program

DRDA and DDM server access control using function usage ID

A security feature of the Distributed Data Management (DDM) and Distributed Relational Database Architecture (DRDA) server, for use with both Advanced Program-to-Program Communication (APPC) and TCP/IP, allows the use of QIBM_DB_DDMDRDA function usage ID to block database server access to users and groups over DDM and DRDA with out having to write a user exit program.

The function usage ID QIBM_DB_DDMDRDA is used to block DDM and DRDA application server inbound connection requests to users and groups and is not based on the communication protocol. It can be configured using the Change Function Usage (CHGFCNUSG) command, Work with Function Usage (WRKFNCUSG) command. The function usage ID is shipped with default authority set to *ALLOWED. The security officer can easily deny access to specific users and groups. This function usage ID is meant to be alternative to a user exit program approach. No coding is required. It is easy to change and auditable.

Example of denying a user DDM or DRDA access:

```
CHGFCNUSG FCNID(QIBM_DB_DDMDRDA) USER(user1) USAGE(*DENIED)
```

The authorization checking for this function usage ID occurs prior to any user exit programs getting control. If the user fails the function usage authorization check, the user exit program will not be called.

Many applications and some IBM products rely upon QRWTSRVR jobs. Disallowing access too broadly can cause these applications to fail in a manner that does not obviously point to function usage administration. Most clients will report this connection failure as connection was dropping when a user is denied access.

Users denied access with see a CPF9898 message "QIBM_DB_DDMDRDA FUNCTION USAGE IS DENIED FOR USER PROFILE XXXXXXXX" at connect time for IBM i releases 6.1, 6.1.1, and 7.1 that will state that the function usage id QIBM_DB_DDMDRDA is set to deny their user profile access. Users that already have a connection made before a system administrator sets their user profile to denied will continue to be able to use their existing connection and be revoked on future connects.

Special cases:

- First, a special case exists when user profiles appear under the function usage more than once. Since a user can belong to one or more group profiles, the function usage specification could refer to the user in multiple ways for a single function usage. First, if multiple group profiles, for which a user belongs to, are referenced under the function usage and the user profile is NOT referenced under the function usage, the user will be granted function usage if ANY of the group profiles listed have USAGE(*ALLOWED). If one group profile was configured with usage *DENIED while the other group profile has *ALLOWED, The user would be granted function usage. All a user needs is for one of their group profiles to have permission to use the function.
- Second, a special case exists when one or more group profiles, for which a user belongs to, are referenced under the function usage and the user profile IS explicitly referenced under the function usage. The explicit user reference will determine whether the user will be granted function usage. Again, a user belongs to two group profiles, GRPA and GRPB. If both groups were configured with usage *ALLOWED and the user profile was had function usage *DENIED, the user would not be granted function usage. Individual function usage references take precedence over group profile references.

Audit records:

Security audit records are written to the QSYS/QAUDJRN security audit journal when auditing is enabled and the auditing level is configured to record authorization failures. A GR-F audit record appears with *USAGEFAILURE will be sent if the function usage check fails.

Example:

```
CHGSYSVAL SYSVAL(QAUDCTL) VALUE('*AUDLVL')
CHGSYSVAL SYSVAL(QAUDLVL) VALUE('*AUTFAIL')
DSPJRN JRN(QSYS/QAUDJRN) FROMTIME('01/25/2012' 080000)

Display Journal Entries

Journal . . . . . . : QAUDJRN Library . . . . . . : QSYS
Largest sequence number on this screen . . . . . . : 00000000000000003036
Type options, press Enter.
5=Display entire entry


Opt Sequence Code Type Object Library Job Time
3035 T GR QRWTSRVR 8:24:29
3036 T GR QRWTSRVR 8:24:30

Display Journal Entry

Object . . . . . . . : Library . . . . . . :
Member . . . . . . . :
Incomplete data . . : No Minimized entry data : *NONE
Sequence . . . . . . : 3036
Code . . . . . . . . : T - Audit trail entry
```

```
Type . . . . . . . . . : GR - General purpose audit record

Entry specific data
Column *...+....1....+....2....+....3....+....4....+....5
00001 'FZRSLFUSER *USAGEFAILURE '
00051 ' '
00101 ' QIBM_DB_DDMDRDA '
00151 ' '
00201 ' '
00251 ' '
00301 ' '
```

Lastly, this function usage ID is shipped with the function usage configuration "*ALLOBJ special authority" set to *USER. This setting allows users with *ALLOBJ user special authority to use function as though they had been granted explicit function usage authority. If you do not want to allow this *ALLOBJ authorization, use the Change Function Usage (CHGFCNUSG) command to change the configuration to *NOTUSED.

Example:

```
CHGFCNUSG FCNID(QIBM_DB_DDMDRDA) ALLOBJAUT(*NOTUSED)
```

Object-related security

If the IBM i product is a server system, there are two object-related levels at which security can be enforced to control access to its relational database tables.

The DDMACC parameter is used on the Change Network Attributes (CHGNETA) command to indicate whether the tables on this system can be accessed at all by another system and, if so, at which level of security the incoming DRDA requests are to be checked.

- If *REJECT is specified on the DDMACC parameter, all distributed relational database requests received by the server are rejected. However, this system (as a client) can still use SQL requests to access tables on other systems that allow it. No remote system can access a database on any IBM i environment that specifies *REJECT.

  If *REJECT is specified while an SQL request is already in use, all *new* jobs from any system requesting access to this system's database are rejected and an error message is returned to those jobs; existing jobs are not affected.

- If *OBJAUT is specified on the DDMACC parameter, normal object-level security is used on the server.

  The DDMACC parameter is initially set to *OBJAUT. A value of *OBJAUT allows all remote requests, but they are controlled by the object authorizations on this server. If the DDMACC value is *OBJAUT, the user profile used for the job must have appropriate object authorizations through private, public, group, or adopted authorities, or the profile must be on an authorization list for objects needed by the client job. For each SQL object on the system, all users, no users, or only specific users (by user ID) can be authorized to access the object.

  The user ID that must be authorized to objects is the user ID of the server job. See the “Elements of security in an APPC network” on page 73 topic for information about what user profile the server job runs under.

  In the case of a TCP/IP connection, the server job initially starts running under QUSER. After the user ID is validated, an exchange occurs so that the job then runs under the user profile specified on the connection request. The job inherits the attributes (for example, the library list) of that user profile.

  When the value *OBJAUT is specified, it indicates that no further verification (beyond IBM i object-level security) is needed.

- For DDM jobs, if the name of an exit program (or access control program) is specified on the DDMACC parameter, an additional level of security is used. The exit program can be used to control whether a user of a DDM client can use a specific command to access a specific file on the IBM i operating system.

  For DRDA jobs, if the name of an exit program (access control program) is specified on the DDMACC parameter, the system treats the entry as though *OBJAUT were specified, with one exception. The only effect that an exit program can have on a DRDA job is to reject a connection request.

The DDMACC parameter, initially set to *OBJAUT, can be changed to one of the previously described values by using the Change Network Attributes (CHGNETA) command, and its current value can be displayed by the Display Network Attributes (DSPNETA) command. You can also get the value in a CL program by using the Retrieve Network Attributes (RTVNETA) command.

If the DDMACC parameter value is changed, although it takes effect immediately, it affects only *new* distributed relational database jobs started on this system (as the server). Jobs running on this server before the change was made continue to use the old value.

Related concepts

Communications Management PDF

DRDA and DDM server access control using user exit programs
A security feature of the Distributed Relational Database Architecture (DRDA) server, for use with both Advanced Program-to-Program Communication (APPC) and TCP/IP, extends the use of the DDMACC parameter of the **Change Network Attributes (CHGNETA)** command to DRDA.

Related reference

Change Network Attributes (CHGNETA) command

Display Network Attributes (DSPNETA) command

Retrieve Network Attributes (RTVNETA) command

DRDA: Authority to distributed relational database objects

You can use either the SQL GRANT and REVOKE statements or the control language (CL) Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT) commands to grant and revoke a user's authority to relational database objects.

The SQL GRANT and REVOKE statements only operate on packages, tables, and views. In some cases, it is necessary to use GRTOBJAUT and RVKOBJAUT to authorize users to other objects, such as commands and programs.

The authority checked for SQL statements depends on whether the statement is static, dynamic, or being run interactively.

For interactive SQL statements, authority is checked against the authority of the person processing the statement. Adopted authority is not used for interactive SQL statements.

Users running a distributed relational database application need authority to run the SQL package on the server. The GRANT EXECUTE ON PACKAGE statement allows the owner of an SQL package, or any user with administrative privileges to it, to grant specified users the privilege to run the statements in an SQL package. You can use this statement to give all users authorized to the server, or a list of one or more user profiles on the server, the privilege to run statements in an SQL package.

Normally, users have processing privileges on a package if they are authorized to the distributed application program created using the CRTSQL*xxx* command. If the package is created using the Create Structured Query Language Package (CRTSQLPKG) command, you might have to grant processing privileges on the package to users. You can issue this statement in an SQL program or using interactive SQL. A sample statement is as follows:

```
GRANT EXECUTE
ON PACKAGE  SPIFFY.PARTS1
TO PUBLIC
```

The REVOKE EXECUTE ON PACKAGE statement allows the owner of an SQL package, or any user with administrative privileges to it, to remove the privilege to run statements in an SQL package from specified users. You can remove the EXECUTE privilege to all users authorized to the server or to a list of one or more user profiles on the server.

If you granted the same privilege to the same user more than once, revoking that privilege from that user nullifies all those grants. If you revoke an EXECUTE privilege on an SQL package you previously granted to a user, it nullifies any grant of the EXECUTE privilege on that SQL package, regardless of who granted it. A sample statement is as follows:

```
REVOKE EXECUTE
ON PACKAGE  SPIFFY.PARTS1
FROM PUBLIC
```

You can also grant authority to an SQL package using the Grant Object Authority (GRTOBJAUT) command or revoke authority to an SQL package using the Revoke Object Authority (RVKOBJAUT) command.

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Grant Object Authority (GRTOBJAUT) command

Revoke Object Authority (RVKOBJAUT) command

Security for SQL objects

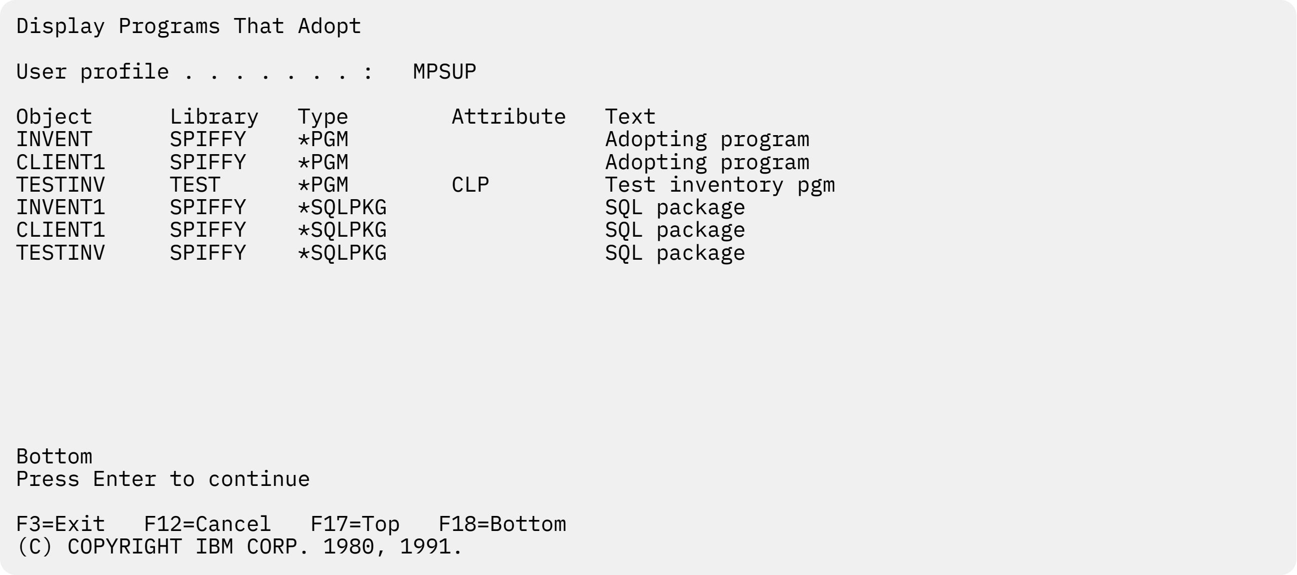
Distributed relational database statements
The statements included with the SQL language specifically support a distributed relational database.

DRDA: Programs that run under adopted authority for a distributed relational database

A distributed relational database program can run under adopted authority, which means the user adopts the program owner's authority to objects used by the program while running the program. When a program is created using the *SQL precompiler option for naming, the program runs under the program owner's user profile.

An SQL package from an unlike system always adopts the package owner's authority for all static SQL statements in the package. An SQL package created on the IBM i operating system using the CRTSQL*xxx* command with OPTION(*SQL) specified also adopts the package owner's authority for all static SQL statements in the package.

A distributed relational database administrator can check security exposure on servers by using the Display Programs that Adopt (DSPPGMADP) command. The DSPPGMADP command displays the programs and SQL packages that use a specified user profile, as shown here. You can also send the results of the command to a printer or to an output file.



```
Display Programs That Adopt

User profile . . . . . . . :   MPSUP

Object      Library   Type        Attribute   Text
INVENT      SPIFFY    *PGM                    Adopting program
CLIENT1     SPIFFY    *PGM                    Adopting program
TESTINV     TEST      *PGM        CLP         Test inventory pgm
INVENT1     SPIFFY    *SQLPKG                 SQL package
CLIENT1     SPIFFY    *SQLPKG                 SQL package
TESTINV     SPIFFY    *SQLPKG                 SQL package









Bottom
Press Enter to continue

F3=Exit   F12=Cancel   F17=Top   F18=Bottom
(C) COPYRIGHT IBM CORP. 1980, 1991.
```

Protection strategies in a distributed relational database

Network security in an IBM i distributed relational database must be planned to protect critical data on any server from unauthorized access. But because of the distributed nature of the relational database, security planning must ensure that availability of data in the network is not unnecessarily restricted.

One of the decisions that a distributed relational database administrator needs to make is which system security level to put in place for each system in the network. A system security level of 10 provides no security for servers other than physical security at the system site. A system security level of 20 provides some protection to servers because network security checking is done to ensure that the local and remote system are correctly identified. However, this level does not provide the object authorization necessary to protect critical database elements from unauthorized access. A system security level of 30 and above is the suggested choice for systems in a network that want to protect specific system objects.

The distributed relational database administrator must also consider how communications are established between clients on the network and the servers. Some questions that need to be resolved might include:

- Should a default user profile exist on a server?

  Maintaining many user profiles throughout a network can be difficult. However, creating a default user profile in a communications subsystem entry opens the server to incoming communications requests if the server is not a secure location. In some cases this might be an acceptable situation, in other cases a default user profile might reduce the system protection capabilities too far to satisfy security requirements.

  For example, systems that serve many clients need a high level of security. If their databases were lost or damaged, the entire network could be affected. Because it is possible to create user profiles or group profiles on a server that identifies all potential users needing access, it is unnecessary for the database administrator to consider creating a default user profile for the communications subsystem or subsystems managing distributed relational database work.

  In contrast, a system that rarely acts as a server to other systems in the network and that does not contain sensitive or critical data might use a default user profile for the communications subsystem managing distributed relational database work. This might prove particularly effective if the same application is used by all the other systems in the network to process work on this database.

  Strictly speaking, the concept of a default user applies only to the use of APPC. However, a similar technique can be used with systems that are using TCP/IP. A single user ID can be established under which the server jobs can run. The Add Server Authentication Entry (ADDSVRAUTE) command can be used on all clients to specify that a user ID should be used for all users to connect with. The server authentication entries can have a password specified on them, or they can specify *NONE for the password, depending on the setting of the PWDRQD parameter on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command at the server. The default value of this attribute is that passwords are required.

- How should access to database objects be handled?

  Authority to objects can be granted through private authority, group authority, public authority, adopted authority, and authentication lists. While a user profile (or default profile) has to exist on the server for the communications request to be accepted, how the user is authorized to objects can affect performance.

  Whenever possible, use group authority or authentication lists to grant access to a distributed relational database object. It takes less time and system resources to check these than to review all private authorities.

  For TCP/IP connections, you do not need a private user ID for each user that can connect to a server, because you can map user IDs.

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

DRDA and DDM application development

Programmers can write high-level language programs that use SQL statements for IBM i distributed application programs. This topic also indicates which languages, utilities, and application programs support DDM, and provides the DDM-specific information to properly access remote files.

Related concepts

SQL programming

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Application development for DRDA

Programmers can write high-level language programs that use SQL statements for IBM i distributed application programs.

The main differences from programs written for local processing only are the ability to connect to remote databases and to create SQL packages. The CONNECT SQL statement can be used to explicitly connect a client to a server, or the name of the relational database can be specified when the program is created to allow an implicit connection to occur. Also, the SET CONNECTION, RELEASE, and DISCONNECT statements can be used to manage connections for applications that use distributed unit of work.

An *SQL package* is an IBM i object used only for distributed relational databases. It can be created as a result of the precompile process of SQL or can be created from a compiled program object. An SQL package resides on the server. It contains SQL statements, host variable attributes, and access plans that the server uses to process a client's request.

Because application programs can connect to many different systems, programmers might need to pay more attention to data conversion between systems. The IBM i operating system provides for conversion of various types of data, including coded character set identifier (CCSID) support for the management of character information.

You can create and maintain programs for a distributed relational database on the system using the SQL language the same way you use it for local-processing applications. You can embed static and dynamic Structured Query Language (SQL) statements with any one or more of the following high-level languages:

- PL/I PRPQ
- ILE C
- COBOL/400
- ILE COBOL
- FORTRAN/400
- RPG/400®
- ILE RPG

The process of developing distributed applications is similar to that of developing SQL applications for local processing. The difference is that the application for distributed processing must specify the name of the relational database to which it connects. This might be done when you precompile the program or within the application.

The same SQL objects are used for both local and distributed applications, except that one object, the SQL package, is used exclusively for distributed relational database support. You create the program using the Create SQL program (CRTSQL*xxx*) command. The *xxx* in this command refers to the host language CI, CBL, CBLI, FTN, PLI, RPG, or RPGI. The SQL package might be a product of the precompile in this process. The Create Structured Query Language Package (CRTSQLPKG) command creates SQL packages for existing distributed SQL programs.

You must have the IBM DB2 Query Manager and SQL Development Kit for i licensed program installed to precompile programs with SQL statements. However, you can create SQL packages from existing distributed SQL programs with only the compiled program installed on your system. The IBM DB2 Query Manager and SQL Development Kit for i licensed program also allows you to use interactive SQL to access a distributed relational database. This is helpful when you are debugging programs because it allows you to test SQL statements without having to precompile and compile a program.

Programming considerations for a distributed relational database application

Programming considerations for a distributed relational database application on the IBM i operating system fall into two main categories: those that deal with a function that is supported on the local system and those that are a result of having to connect to other systems.

Related reference

Tips: Designing distributed relational database applications
Distributed relational database applications have different requirements from applications developed solely for use on a local database.

Naming of distributed relational database objects

SQL objects are created and maintained as IBM i objects. You can use either of the two naming conventions in Db2 for i programming: system (*SYS) and SQL (*SQL).

The naming convention that you use affects the method for qualifying file and table names. It also affects security and the terms used on the interactive SQL displays. Distributed relational database applications can access objects on another IBM i platform using either naming convention. However, if your program accesses a system that is not Db2 for i, only SQL names can be used. Names can be specified by using the NAMING parameter on the Start SQL (STRSQL) command, by using the OPTION parameter on one of the CRTSQLxxx commands, or by using the naming connection property for call level interface (CLI) and Java Database Connectivity (JDBC).

Related reference

Naming conventions

*System (*SYS) naming convention*

When you use the system naming convention, files are qualified by library name in the form *library/file*.

Tables created using this naming convention assume the public authority of the library in which they are created. If the table name is not explicitly qualified and a default collection name is used in the DFTRDBCOL parameter of the CRTSQL*xxx* or CRTSQLPKG command, the default collection name is used for static SQL statements. If the file name is not explicitly qualified and the default collection name is not specified, the following rules apply:

- All SQL statements except certain CREATE statements cause SQL to search the library list (*LIBL) for the unqualified file.
- The CREATE statements resolve to unqualified objects as follows:
  - CREATE TABLE: The table name must be explicitly qualified.
  - CREATE VIEW: The view is created in the first library referred to in the subselect.
  - CREATE INDEX: The index is created in the collection or library that contains the table on which the index is being built.

*SQL (*SQL) naming convention*

When you use the SQL naming convention, tables are qualified by the collection name in the form *collection.table*.

If the table name is not explicitly qualified and the default collection name is specified in the default relational database collection (DFTRDBCOL) parameter of the CRTSQLxxx or **`Create Structured Query Language Package (CRTSQLPKG)`** command, the default collection name is used. If the table name is not explicitly qualified and the default collection name is not specified, the following rules apply:

- For static SQL, the default qualifier is the user profile of the program owner.
- For dynamic SQL or interactive SQL, the default qualifier is the user profile of the job running the statement.

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Default collection name
You can specify a default collection name to be used by an SQL program by supplying this name for the DFTRDBCOL parameter on the CRTSQL*xxx* command when you precompile the program.

The DFTRDBCOL parameter provides the program with the collection name as the library for an unqualified file if the *SYS naming convention is used, or as the collection for an unqualified table if the *SQL naming convention is used. If you do not specify a default collection name when you precompile the program, the rules for unqualified names apply for each naming convention. The default relational database collection name only applies to static SQL statements.

You can also use the DFTRDBCOL parameter on the **`Create Structured Query Language Package (CRTSQLPKG)`** command to change the default collection of a package. After an SQL program is compiled, you can create a new SQL package to change the default collection.

Related tasks

Using the Create SQL Package (CRTSQLPKG) command
You can enter the Create SQL Package (CRTSQLPKG) command to create an SQL package from a compiled distributed relational database program. You can also use this command to replace an SQL package that was created previously.

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Connecting to a distributed relational database
What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another system.

There are two types of CONNECT statements with the same syntax but different semantics:

- CONNECT (Type 1) is used for remote unit of work.
- CONNECT (Type 2) is used for distributed unit of work.

For embedded SQL applications, the RDBCNNMTH parameter on the CRTSQLxxx commands determines the type of CONNECT. CLI and Java applications always use distributed unit of work because of the nature of the language.

Related reference

Distributed relational database statements
The statements included with the SQL language specifically support a distributed relational database.

Remote unit of work
The *remote unit of work* facility provides for the remote preparation and processing of SQL statements.

An activation group at system A can connect to a server at system B. Then, within one or more units of work, that activation group can process any number of static or dynamic SQL statements that reference objects at B. After ending a unit of work at B, the activation group can connect to a server at system C, and so on.

Most SQL statements can be remotely prepared and processed with the following restrictions:

- All objects referenced in a single SQL statement must be managed by the same server.
- All of the SQL statements in a unit of work must be run by the same server.

Activation group states
An activation group is in one of three states at any time.

The three states are:

- Connectable and connected
- Unconnectable and connected
- Connectable and unconnected

The following figure shows the state transitions:

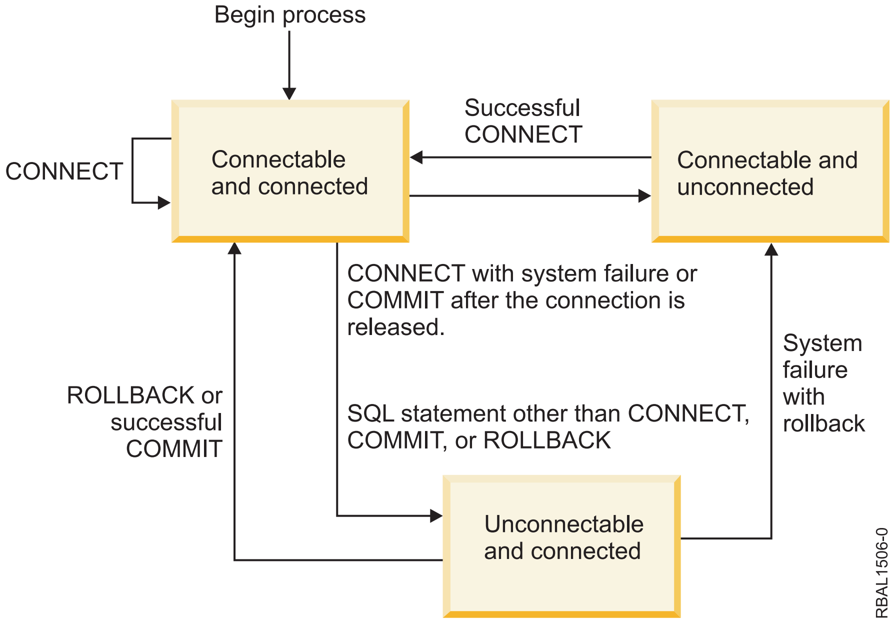

Begin process
CONNECT
Connectable and connected
Successful CONNECT
Connectable and unconnected
CONNECT with system failure or COMMIT after the connection is released.
System failure with rollback
ROLLBACK or successful COMMIT
SQL statement other than CONNECT, COMMIT, or ROLLBACK
Unconnectable and connected
RBAL1506-0

Figure 23. Remote unit of work activation group connection state transition

The initial state of an activation group is *connectable* and *connected*. The server to which the activation group is connected is determined by the RDB parameter on the CRTSQLxxx and STRSQL commands and might involve an implicit CONNECT operation. An implicit CONNECT operation cannot occur if an implicit or explicit CONNECT operation has already successfully or unsuccessfully occurred. Thus, an activation group cannot be implicitly connected to a server more than once.

Connectable and connected state
An activation group is connected to a server and CONNECT statements can be executed. The activation group enters this state when it completes a rollback or successful commit from the unconnectable and connected state, or a CONNECT statement is successfully executed from the connectable and unconnected state.

Unconnectable and connected state
An activation group is connected to a server, but a CONNECT statement cannot be successfully executed to change servers. The activation group enters this state from the connectable and connected state when it executes any SQL statement other than CONNECT, COMMIT, or ROLLBACK.

Connectable and unconnected state
An activation group is not connected to a server. The only SQL statement that can be executed is CONNECT.

The activation group enters this state when:

- The connection was previously released and a successful COMMIT is executed.
- The connection is disconnected using the SQL DISCONNECT statement.
- The connection was in a connectable state, but the CONNECT statement was unsuccessful.

Consecutive CONNECT statements can be executed successfully because CONNECT does not remove the activation group from the connectable state. A CONNECT to the server to which the activation group is currently connected is executed like any other CONNECT statement.

CONNECT cannot run successfully when it is preceded by any SQL statement other than CONNECT, COMMIT, DISCONNECT, SET CONNECTION, RELEASE, or ROLLBACK (unless running with COMMIT(*NONE)). To avoid an error, execute a commit or rollback operation before a CONNECT statement is executed.

Distributed unit of work
The application-directed distributed unit of work facility also provides for the remote preparation and execution of SQL statements in the same fashion as remote unit of work.

Like remote unit of work, an activation group at system A can connect to a server at system B and run any number of static or dynamic SQL statements that reference objects at B before ending the unit of work. All objects referenced in a single SQL statement must be managed by the same server. However, unlike remote unit of work, any number of servers can participate in the same unit of work. A commit or rollback operation ends the unit of work.

Activation group states
An activation group is always in the *connected* or *unconnected* state and has a set of zero or more connections. Each connection of an activation group is uniquely identified by the name of the server of the connection.

An SQL connection is always in one of the following states:

- Current and held
- Current and released
- Dormant and held
- Dormant and released

Initial state of an activation group
An activation group is initially in the connected state and has exactly one connection. The initial state of a connection is *current and held*.

The following figure shows the state transitions:

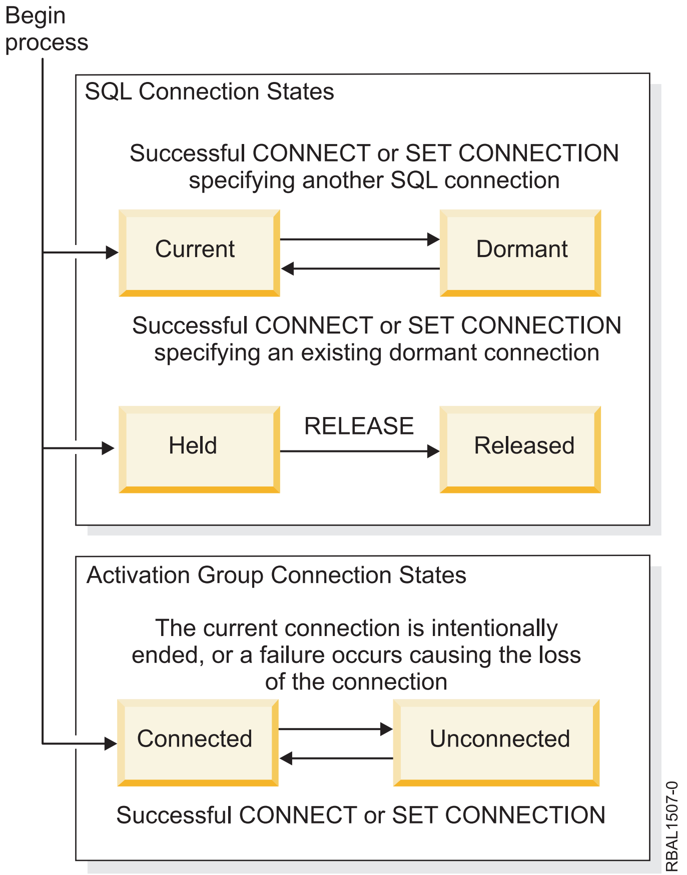

Begin process
SQL Connection States
Successful CONNECT or SET CONNECTION specifying another SQL connection
Current
Dormant
Successful CONNECT or SET CONNECTION specifying an existing dormant connection
Held
RELEASE
Released
Activation Group Connection States
The current connection is intentionally ended, or a failure occurs causing the loss of the connection
Connected
Unconnected
Successful CONNECT or SET CONNECTION
RBAL1507-0

Figure 24. Application-directed distributed unit of work connection and activation group connection state transitions

Connection states

This topic discusses the different connection states and ways to change them.

If an application processes a CONNECT statement and the server name is known to the application requester and is not in the set of existing connections of the activation group, then:

- The current connection is placed in the dormant state and held state.
- The server name is added to the set of connections and the new connection is placed in the current and held state.

If the server name is already in the set of existing connections of the activation group, an error occurs.

A connection in the dormant state is placed in the current state using the SET CONNECTION statement. When a connection is placed in the current state, the previous current connection, if any, is placed in the dormant state. No more than one connection in the set of existing connections of an activation group can be current at any time. Changing the state of a connection from current to dormant or from dormant to current has no effect on its held or released state.

A connection is placed in the released state by the RELEASE statement. When an activation group executes a commit operation, every released connection of the activation group is ended. Changing the state of a connection from held to released has no effect on its current or dormant state. Thus, a connection in the released state can still be used until the next commit operation. There is no way to change the state of a connection from released to held.

Activation group connection states

A different server can be established by the explicit or implicit execution of a CONNECT statement.

The following rules apply:

- An activation group cannot have more than one connection to the same server at the same time.
- When an activation group executes a SET CONNECTION statement, the specified location name must be an existing connection in the set of connections of the activation group.
- When an activation group executes a CONNECT statement, the specified server name must not be an existing connection in the set of connections of the activation group.

If an activation group has a current connection, the activation group is in the *connected* state.

The CURRENT SERVER special register contains the name of the server of the current connection. The activation group can execute SQL statements that refer to objects managed by that server.

An activation group in the unconnected state enters the connected state when it successfully executes a CONNECT or SET CONNECTION statement.

If an activation group does not have a current connection, the activation group is in the *unconnected* state. The CURRENT SERVER special register contents are equal to blanks. The only SQL statements that can be executed are CONNECT, DISCONNECT, SET CONNECTION, RELEASE, COMMIT, and ROLLBACK.

An activation group in the connected state enters the unconnected state when its current connection is intentionally ended or the execution of an SQL statement is unsuccessful because of a failure that causes a rollback operation at the server and loss of the connection. Connections are intentionally ended when an activation group successfully executes a commit operation and the connection is in the released state, or when an application process successfully executes the DISCONNECT statement.

When a connection is ended

When a connection is ended, all resources that were acquired by the activation group through the connection and all resources that were used to create and maintain the connection are no longer allocated.

For example, when the activation group executes a RELEASE statement, any open cursors will be closed when the connection is ended during the next commit operation.

A connection can also be ended as a result of a communications failure in which case the activation group is placed in the unconnected state. All connections of an activation group are ended when the activation group ends.

Run with both RUW and DUW connection management

Programs compiled with remote unit of work (RUW) connection management can be called by programs compiled with distributed unit of work (DUW) connection management. SET CONNECTION, RELEASE, and DISCONNECT statements can be used by the program compiled with RUW connection management to work with any of the active connections.

However, when a program compiled with DUW connection management calls a program compiled with RUW connection management, CONNECTs that are performed in the program compiled with RUW connection management will attempt to end all active connections for the activation group as part of the CONNECT.

Such CONNECTs will fail if the conversation used by active connections uses protected conversations. Furthermore, when protected conversations were used for inactive connections and the DDMCNV job attribute is *KEEP, these unused DDM conversations will also cause the connections in programs compiled with RUW connection management to fail. To avoid this situation, run with DDMCNV(*DROP) and perform a RELEASE and COMMIT prior to calling any programs compiled with RUW connection management that perform CONNECTs.

Likewise, when creating packages for programs compiled with DUW connection management after creating a package for a program compiled with RUW connection management, either run with DDMCNV(*DROP) or perform a RCLDDMCNV after creating the package for the programs compiled with DUW connection management.

Programs compiled with DUW connection management can also be called by programs compiled with RUW connection management. When the program compiled with DUW connection management performs a CONNECT, the connection performed by the program compiled with RUW connection management is not disconnected. This connection can be used by the program compiled with DUW connection management.

Implicit connection management for the default activation group

Implicit connection occurs when the client detects the first SQL statement is being issued by the first active SQL program for the default activation group and these items are true.

- The SQL statement being issued is not a CONNECT statement with parameters.
- SQL is not active in the default activation group.

For a distributed program, the implicit connection is to the relational database specified on the RDB parameter. For a nondistributed program, the implicit connection is to the local relational database.

SQL ends any active connections in the default activation group when SQL becomes not active. SQL becomes not active when the client detects that the first active SQL program for the process has ended and the following conditions are all met:

- There are no pending SQL changes
- There are no connections using protected conversations
- A SET TRANSACTION statement is not active
- No programs that were precompiled with CLOSQLCSR(*ENDJOB) were run

If there are pending changes, protected conversations, or an active SET TRANSACTION statement, then SQL is placed in the exited state. If programs precompiled with CLOSQLCSR(*ENDJOB) were run, then SQL will remain active for the default activation group until the job ends.

- At the end of a unit of work if SQL is in the exited state. This occurs when you issue a COMMIT or ROLLBACK command outside of an SQL program.
- At the end of a job.

Implicit connection management for nondefault activation groups

Implicit connection occurs when the client detects the first SQL statement issued for the activation group and it is not a CONNECT statement with parameters.

For a distributed program, the implicit connection is made to the relational database specified on the RDB parameter. For a nondistributed program, the implicit connection is made to the local relational database.

Implicit disconnection can occur at the following parts of a process:

- When the activation group ends, if commitment control is not active, activation group level commitment control is active, or the job level commitment definition is at a unit of work boundary.

  If the job level commitment definition is active and not at a unit of work boundary then SQL is placed in the exited state.
- If SQL is in the exited state, when the job level commitment definition is committed or rolled back.
- At the end of a job.

The following example program is not distributed (no connection is required). It is a program run at a Spiffy Corporation regional office to gather local repair information into a report.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
CRTSQLxxx PGM(SPIFFY/FIXTOTAL) COMMIT(*CHG) RDB(*NONE)

    PROC: FIXTOTAL;
       .
       .
       .
      SELECT * INTO :SERVICE      A
               FROM REPAIRTOT;
    EXEC SQL
    COMMIT;
       .
       .
       .
    END FIXTOTAL;
```

A

Statement run on the local relational database

Another program, such as the following example, can gather the same information from Spiffy dealerships in the Kansas City region. This is an example of a distributed program that is implicitly connected and disconnected:

```
CRTSQLxxx PGM(SPIFFY/FIXES) COMMIT(*CHG) RDB(KC101) RDBCNNMTH(*RUW)

    PROC: FIXES;
       .
       .
       .
    EXEC SQL
      SELECT * INTO :SERVICE              B
               FROM SPIFFY.REPAIR1;

    EXEC SQL  C
     COMMIT;
       .
       .
       .
    END FIXES;  D
```

B

Implicit connection to server. The statement runs on the server.

C

End of unit of work. The client is placed in a connectable and connected state if the COMMIT is successful.

D

Implicit disconnection at the end of the SQL program.

Explicit connection management

The CONNECT statement is used to explicitly connect a client to an identified server. This SQL statement can be embedded within an application program or you can issue it using interactive SQL.

The CONNECT statement is used with a TO or RESET clause. A CONNECT statement with a TO clause allows you to specify connection to a particular AS relational database. The CONNECT statement with a RESET clause specifies connection to the local relational database.

When you issue (or the program issues) a CONNECT statement with a TO or RESET clause, the AS identified must be described in the relational database directory. The AR must also be in a connectable state for the CONNECT statement to be successful.

The CONNECT statement has different effects depending on the connection management method you use. For RUW connection management, the CONNECT statement has the following effects:

- When a CONNECT statement with a TO or RESET clause is successful, the following events take place:
  - Any open cursors are closed, any prepared statements are discarded, and any held resources are released from the previous AS if the application process was placed in the connectable state through the use of COMMIT HOLD or ROLLBACK HOLD SQL statement, or if the application process is running COMMIT(*NONE).
  - The application process is disconnected from its previous AS, if any, and connected to the identified AS.
  - The name of the AS is placed in the Current Server special register.
  - Information that identifies the system module that returned the error is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
- If the CONNECT statement is unsuccessful for any reason, the application remains in the connectable but unconnected state. An application in the connectable but unconnected state can only run the CONNECT statement.
- Consecutive CONNECT statements can be run successfully because CONNECT does not remove the AR from the connectable state. A CONNECT to the AS to which the AR is currently connected is run like any other CONNECT statement.
- If running with commitment control, the CONNECT statement cannot run successfully when it is preceded by any SQL statement other than CONNECT, SET CONNECTION, COMMIT, ROLLBACK, DISCONNECT, or RELEASE. To avoid an error, perform a COMMIT or ROLLBACK operation before a CONNECT statement is run. If running without commitment control, the CONNECT statement is always allowed.

For DUW connection management, the CONNECT statement has the following effects:

- When a CONNECT statement with a TO or RESET clause is successful, the following events take place:
  - The name of the AS is placed in the Current Server special register.
  - Information that identifies the system module that returned the error is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
  - Information on the type of connection is also placed into the SQLCA and SQL diagnostic area. Encoded in these is the following information:
    - Whether the application is in a connected or unconnected state can be found in SQLERRD(5) in the SQLCA or in DB2_CONNECTION_STATE in the SQL diagnostic area.
    - Whether a remote connection uses a protected or unprotected conversation is found in SQLERRD(4) in the SQLCA or in DB2_CONNECTION_TYPE in the SQL diagnostic area.
    - Whether the connection is always read-only, always capable of updates, or whether the ability to update can change between each unit of work is found in SQLERRD(4) in the SQLCA or in DB2_CONNECTION_STATUS in the SQL diagnostic area.

See the SQL programming topic for more information about SQLERRD fields in the SQLCA and about connection information in the SQL diagnostic area.

- If the CONNECT statement with a TO or RESET clause is unsuccessful because the AR is not in the connectable state or the *server-name* is not listed in the local relational database directory, the connection state of the AR is unchanged.
- A connection to a currently connected AS results in an error.
- A connection without a TO or RESET clause can be used to obtain information about the current connection. This includes the following information:
  - Information that identifies the system module that returned the status is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
  - Other status information is described in the previous paragraphs discussing the contents of SQLERRD(4) and SQLERRD(5) and the corresponding information in the SQL diagnostic area.

It is a good practice for the first SQL statement run by an application process to be the CONNECT statement. However, when you have CONNECT statements embedded in your program, you might want to dynamically change the AS name if the program connects to more than one AS. If you are going to run the application at multiple systems, you can specify the CONNECT statement with a host variable as shown here so that the program can be passed the relational database name.

```
CONNECT TO  : host-variable
```

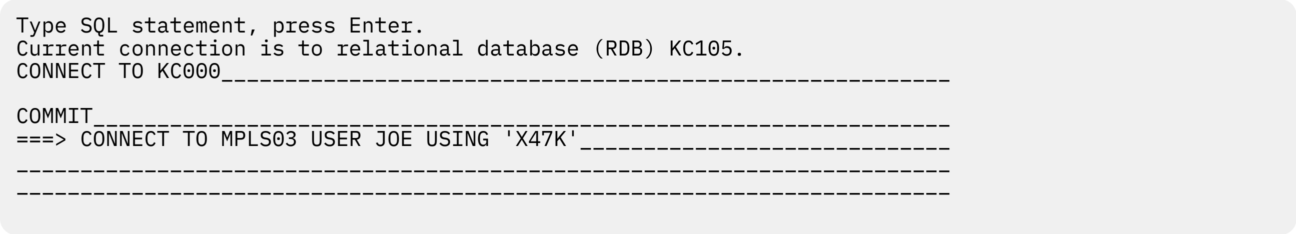
Without CONNECT statements, all you need to do when you change the AS is to recompile the program with the new relational database name.

The following example shows two forms of the CONNECT statement (**1** and **2**) in an application program:

```
CRTSQLxxx PGM(SPIFFY/FIXTOTAL) COMMIT(*CHG) RDB(KC105)

  PROC: FIXTOTAL;
  EXEC SQL   CONNECT TO KC105; 1

  EXEC SQL
    SELECT * INTO :SERVICE
             FROM REPAIRTOT;

  EXEC SQL COMMIT;

  EXEC SQL CONNECT TO MPLS03 USER :USERID USING :PW;  2

  EXEC SQL SELECT …

  EXEC SQL COMMIT;

  END FIXTOTAL;
```

The example (**2**) shows the use of the USER/USING form of the CONNECT statement. You must specify the user ID and password with host variables when this form of the CONNECT statement is embedded in a program. If you are using TCP/IP, a user ID and password can be extracted from a security object at connect time if you have used the **Add Server Authentication Entry (ADDSVRAUTE)** command with the appropriate parameters to store them.

The following example shows both CONNECT statement forms in interactive SQL. Note that the password must be enclosed in single quotation marks.



```
Type SQL statement, press Enter.
Current connection is to relational database (RDB) KC105.
CONNECT TO KC000__________________________________________________________

COMMIT____________________________________________________________________
===> CONNECT TO MPLS03 USER JOE USING 'X47K'______________________________
__________________________________________________________________________
__________________________________________________________________________
```

Related concepts

Using the relational database directory
The IBM i operating system uses the relational database directory to define the relational database names that can be accessed by system applications and to associate these relational database names with their corresponding network parameters. The system also uses the directory to specify if the connection uses Systems Network Architecture (SNA) or IP.

SQL specific to distributed relational database and SQL CALL

During the precompile process of a distributed Db2 for i application, the IBM i licensed program might build SQL packages to be run on a server.

After the application is compiled, a distributed SQL program and package must be compatible with the systems that are being used as clients and servers. The Preparing distributed relational database programs topic gives you more information about the changes to the precompile process and the addition of SQL packages.

These topics give an overview of the SQL statements that are used with distributed relational database support and some things for you to consider about coexistence with other systems.

Related concepts

Preparing distributed relational database programs
When you write a program using SQL, you can embed the SQL statements in a host program.

SQL programming

Related reference

SQL reference

Distributed relational database statements

The statements included with the SQL language specifically support a distributed relational database.

These statements include:

- CONNECT
- SET CONNECTION
- RELEASE
- DISCONNECT
- DROP PACKAGE
- GRANT EXECUTE ON PACKAGE
- REVOKE EXECUTE ON PACKAGE

The SQL CALL statement can be used locally, but its primary purpose is to allow a procedure to be called on a remote system.

Related concepts

Connecting to a distributed relational database
What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another system.

DRDA: Authority to distributed relational database objects
You can use either the SQL GRANT and REVOKE statements or the control language (CL) Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT) commands to grant and revoke a user's authority to relational database objects.

Working with SQL packages

An *SQL package* is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on a server.

SQL CALL statement (stored procedures)
The SQL CALL statement is not actually specific to distributed relational databases, but a discussion of it is included here because its main value is in distributing application logic and processing.

Result sets can be generated in the stored procedure by opening one or more SQL cursors associated with SQL SELECT statements. In addition, a maximum of one array result set can also be returned. For more information about writing stored procedures that return result sets, see the descriptions of the SET RESULT SETS and CREATE PROCEDURE statements in the SQL reference topic.

The CALL statement provides a capability in a DRDA environment much like the Remote Procedure Call (RPC) mechanism does in the Open Software Foundation (OSF) Distributed Computing Environment (DCE). In fact, an SQL CALL to a program on a remote relational database actually is a remote procedure call. This type of RPC has certain advantages; for instance, it does not require the compilation of interface definitions, nor does it require the creation of stub programs.

You might want to use SQL CALL, or *stored procedures*, as the technique is sometimes called, for the following reasons:

- To reduce the number of message flows between the client and server to perform a given function. If a set of SQL operations are to be run, it is more efficient for a program at the server to contain the statements and interconnecting logic.
- To allow local database operations to be performed at the remote location.
- To perform nondatabase operations (for example, sending messages or performing data queue operations) using SQL.

  Note: Unlike database operations, these operations are not protected by commitment control by the system.
- To access system APIs on a remote system.

A stored procedure and application program can run in the same or different activation groups. It is recommended that the stored procedure be compiled with ACTGRP(*CALLER) specified to achieve consistency between the application program at the client and the stored procedure at the server. If the stored procedure is designed to return result sets, then you should not create it to run in a *NEW activation group. If you do, the cursors associated with the result sets might be prematurely closed when the procedure returns to the caller and the activation group is destroyed.

When a stored procedure is called that issues an inquiry message, the message is sent to the QSYSOPR message queue. The stored procedure waits for a response to the inquiry message. To have the stored procedure respond to the inquiry message, use the **`Add Reply List Entry (ADDRPYLE)`** command and specify *SYSRPYL on the INQMSGRPY parameter of the **`Change Job (CHGJOB)`** command in the stored procedure.

When a stored procedure and an application program run under different commitment definitions, the COMMIT and ROLLBACK statements in the application program only affect its own commitment definition. You must commit the changes in the stored procedure by other means.

Related reference
SQL reference
Add Reply List Entry (ADDRPYLE) command
Change Job (CHGJOB) command
Testing and debugging

Testing and debugging distributed SQL programs is similar to testing and debugging local SQL programs, but certain aspects of the process are different.

Db2 for i CALL considerations
Stored procedures written in C that are called on some platforms cannot use argc and argv as parameters (that is, they cannot be of type main()). This differs from IBM i stored procedures, which must use argc and argv.

For examples of stored procedures for DB2 platforms, see the \SQLLIB\SAMPLES (or /sqllib/samples) subdirectory. Look for outsrv.sqc and outcli.sqc in the C subdirectory.

For DB2 stored procedures called by the IBM i operating system, make sure that the procedure name is in uppercase. The system currently converts procedure names to uppercase. This means that a procedure on the DB2 server, having the same procedure name but in lowercase, will not be found. For IBM i stored procedures, the procedure names are in uppercase.

Stored procedures on the IBM i operating system cannot contain a COMMIT statement when they are created to run in the same activation group as the calling program (the proper way to create them). In DB2, a stored procedure is allowed to contain a COMMIT statement, but the application designer should be aware that there is no knowledge on the part of Db2 for i that the commit occurred.

Db2 for i coexistence
When you write and maintain programs for a distributed relational database using the SQL language, you need to consider the other systems in the distributed relational database network.

The program you are writing or maintaining might have to be compatible with the following items:

- Other IBM i products
- Previous IBM i operating system releases
- Systems other than IBM i

Remember that the SQL statements in a distributed SQL program run on the server. Even though the program runs on the client, the SQL statements are in the SQL package to be run on the server. Those statements must be supported by the server and be compatible with the collections, tables, and views that exist on the server. Also, the users who run the program on the client must be authorized to the SQL package and other SQL objects on the server.

You can convert a non-distributed embedded SQL program to a distributed embedded SQL program by creating the program again using the CRTSQL*xxx* command and specifying the relational database name (RDB parameter) for a server. This compiles the program again using the distributed relational database support in Db2 for i and creates the SQL package needed on the server.

You can write Db2 for i programs that run on server systems that do not run IBM i, and these other platforms might support more or fewer SQL functions. Statements that are not supported on the Db2 for i client can be used and compiled on the server when the server supports the function. SQL programs written to run on an IBM i server system only provide the level of support described in this topic collection. See the support documentation for the other systems to determine the level of function they provide.

Ending DRDA units of work
You should be careful about ending SQL programs with uncommitted work. When a program ends with uncommitted work, the connection to the relational database remains active.

However, in some cases involving programs running in system-named activation groups, the system performs an automatic commit operation when the program ends.

This behavior differs from that of other systems because in the IBM i operating system, COMMITs and ROLLBACKs can be used as commands from the command line or in a CL program. However, the preceding scenario can lead to unexpected results in the next SQL program run, unless you plan for the situation. For example, if you run interactive SQL next (STRSQL command), the interactive session starts up in the state of being connected to the previous server with uncommitted work. As another example, if following the preceding scenario, you start a second SQL program that does an implicit connection, an attempt is made to find and run a package for it on the AS that was last used. This might not be the server that you intended. To avoid these surprises, always commit or roll back the last unit of work before ending any application program.

Stored procedures, user-defined functions, and commitment control

When an application, such as interactive SQL, is running without commitment control active (COMMIT(*NONE)) over a DRDA connection, it is possible for a called stored procedure or user-defined function (UDF) to start commitment control on the IBM i operating system.

This results in a mismatch in commitment control between the client and the server, causing the possibility of uncommitted updates when the application terminates.

You should avoid this situation. If, however, you choose to implement it, one solution is for the stored procedure or UDF running under commitment control to explicitly commit all of its database updates. If that is not done, the server will detect the pending updates during the disconnection process and automatically commit the pending work.

Coded character set identifier

Support for the national language of any country requires the proper handling of a minimum set of characters.

A cross-system support for the management of character information is provided with the IBM Character Data Representation Architecture (CDRA). CDRA defines the coded character set identifier (CCSID) values to identify the code points used to represent characters, and to convert these codes (character data), as needed to preserve their meanings.

The use of an architecture such as CDRA and associated conversion protocols is important in the following situations:

- More than one national language version is installed on the IBM i operating system.
- Multiple IBM i operating systems are sharing data between systems in different countries with different primary national language versions.
- IBM i operating systems and systems other than IBM i are sharing data between systems in different countries with different primary national language versions.

Tagging is the primary means to assign meaning to coded graphic characters. The tag might be in a data structure that is associated with the data object (explicit tagging), or it might be inherited from objects such as the job or the system itself (implicit tagging).

Db2 for i tags character columns with CCSIDs. A *CCSID* is a 16-bit number identifying a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and additional coding-related information that uniquely identifies the coded graphic character representation used. When running applications, data is not converted when it is sent to another system; it is sent as tagged along with its CCSID. The receiving job automatically converts the data to its own CCSID if it is different from the way the data is tagged.

The CDRA has defined the following range of values for CCSIDs.

00000
Use next hierarchical CCSID

00001 through 28671
IBM-registered CCSIDs

28672 through 65533
Reserved

65534
Refer to lower hierarchical CCSID

65535
No conversion done

See *Character Data Representation Architecture - Level 1, Registry* for a complete list of the CDRA CCSIDs.

The following illustration shows the parts of a CCSID.

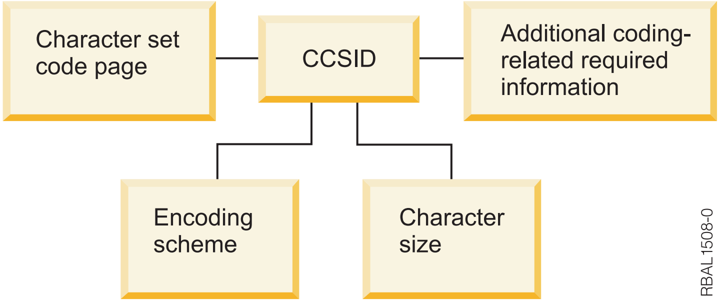


Figure 25. Coded character set identifier (CCSID)

Related concepts

IBM i globalization

SQL programming

DRDA and CDRA support

A distributed relational database might not only span different types of computers, but those computers might be in different countries or regions.

Related reference

SQL reference

IBM i support

You can change the CCSID for an IBM i job by using the Change Job (CHGJOB) command.

If a CCSID is not specified in this way, the job CCSID is obtained from the CCSID attribute of the user profile. If a CCSID is not specified on the user profile, the system gets it from the QCCSID system value. This QCCSID value is initially set to 65535. If your system is in a distributed relational database with unlike systems, it might not be able to use CCSID 65535.

All control information that flows between the client and server is in CCSID 500 (a DRDA standard). This is information such as collection names, table names, and some descriptive text. Using variant characters for control information causes these names to be converted, which can affect performance. Package names are also sent in CCSID 500. Using variant characters in a package name causes the package name to be converted. This means the package is not found at run time.

After a job has been initiated, you can change the job CCSID by using the Change Job (CHGJOB) command. To do this:

1. Enter the Work with Job (WRKJOB) command to get the Work with Jobs display.
2. Select option 2 (Display job definition attributes). This locates the current CCSID value so you can reset the job to its original CCSID value later.
3. Enter the Change Job (CHGJOB) command with the new CCSID value.

The new CCSID value is reflected in the job immediately. However, if the job CCSID you change is an AR job, the new CCSID does not affect the work being done until the next CONNECT.

⚠ **Attention:** If you change the CCSID of a server job, the results cannot be predicted.

Source files are tagged with the job CCSID if a CCSID is not explicitly specified on the Create Source Physical File (CRTSRCPF) or Create Physical File (CRTPF) command for source files. Externally described database files and tables are tagged with the job CCSID if a CCSID is not explicitly specified in data description specification (DDS), in interactive data definition utility (IDDU), or in the CREATE TABLE SQL statement.

For source and externally described files, if the job CCSID is 65535, the default CCSID based on the language of the operating system is used. Program described files are tagged with CCSID 65535. Views are tagged with the CCSID of its corresponding table tag or column-level tags. If a view is defined over several tables, it is tagged at the column level and assumes the tags of the underlying columns. Views cannot be explicitly tagged with a CCSID. The system automatically converts data between the job and the table if the CCSIDs are not equal and neither of the CCSIDs is equal to 65535.

When you change the CCSID of a tagged table, it cannot be tagged at the column level or have views defined on it. To change the CCSID of a tagged table, use the Change Physical File (CHGPF) command. To change a table with column-level tagging, you must create it again and copy the data to a new table using FMT(*MAP) on the Copy File (CPYF) command. When a table has one or more views defined, you must follow these steps to change the table:

1. Save the view and table along with their access paths.
2. Delete the views.
3. Change the table.
4. Restore the views and their access paths over the created table.

Source files and externally described files migrated to Db2 for i that are not tagged or are implicitly tagged with CCSID 65535 will be tagged with the default CCSID based on the language of the operating system installed. This includes files that are on the system when you install a new release and files that are restored to Db2 for i.

All data that is sent between an AR and an AS is sent not converted. In addition, the CCSID is also sent. The receiving job automatically converts the data to its own CCSID if it is different from the way the data is tagged. For example, consider the following application that is run on a dealership system, KC105.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
CRTSQLxxx PGM(PARTS1) COMMIT(*CHG)  RDB(KC000)

    PROC: PARTS1;
       .
       .
    EXEC SQL
      SELECT * INTO :PARTAVAIL
               FROM INVENTORY
               WHERE ITEM = :PARTNO;
       .
       .
    END PARTS1;
```

In the preceding example, the local system (KC105) has the QCCSID system value set at CCSID 37. The remote regional center (KC000) uses CCSID 937 and all its tables are tagged with CCSID 937. CCSID processing takes place as follows:

- The KC105 system sends an input host variable (:PARTNO) in CCSID 37. (The DECLARE VARIABLE SQL statement can be used if the CCSID of the job is not appropriate for the host variable.)
- The KC000 system converts :PARTNO to CCSID 937, selects the required data, and sends the data back to KC105 in CCSID 937.
- When KC105 gets the data, it converts the data to CCSID 37 and places it in :PARTAVAIL for local use.

Related concepts

User FAQs
You need to consider these conditions when working with another specific IBM product.

Related reference

Change Job (CHGJOB) command

Change Physical File (CHGPF) command

Copy File (CPYF) command

Create Physical File (CRTPF) command

Create Source Physical File (CRTSRCPF) command

Work with Job (WRKJOB) command

Other DRDA data conversion

Sometimes, when you are doing processing on a remote system, your program might need to convert the data from one system so that it can be used on the other. Distributed Relational Database Architecture (DRDA) support on the IBM i operating system converts the data automatically between other systems that use DRDA support.

When a Db2 for i client connects to a server, it sends information that identifies its type. Likewise, the server sends back information to the system that identifies its processor type (for example, z/OS host or IBM i). The two systems then automatically convert the data between them as defined for this connection. This means that you do not need to program for architectural differences between systems.

Data conversion between IBM systems with DRDA support includes data types, such as:

- Floating point representations
- Zoned decimal representations
- Byte reversal
- Mixed data types
- Data types specific to IBM i:
  - DBCS-only
  - DBCS-either
  - Integer with precision and scale

The following is a list of data types and their mapped data types if required.

Table 6. Mapped data types

Source Type	**Mapped Type**
XML string internal encoding	BLOB
XML string external encoding	CLOB or DBCLOB
Fixed binary	char(n) for BIT
Variable binary	varchar(n) for BIT
Decimal floating point	8-byte floating point

Preparing distributed relational database programs

When you write a program using SQL, you can embed the SQL statements in a host program.

The *host program* is the program that contains the SQL statements, written in one of the *host languages*: PL/I PRPQ, ILE C, COBOL/400, ILE COBOL, FORTRAN/400, RPG/400, or ILE RPG programming language. In a host program, you use variables referred to as *host variables*. These are variables used in SQL statements that are identifiable to the host program. In RPG, this is called a *field name*; in FORTRAN, PL/I, and C, this is known as a *variable*; in COBOL, this is called a *data item*.

You can code your distributed Db2 for i programs in a way similar to the coding for a Db2 for i program that is not distributed. You use the host language to embed the SQL statements with the host variables. Also, like a Db2 for i program that is not distributed, a distributed Db2 for i program is prepared using the certain processes.

However, a distributed Db2 for i program also requires that an SQL package is created on the server to access data.

This topic collection discusses these steps in the process, outlining the differences for a distributed Db2 for i program.

Precompiling programs with SQL statements

You must precompile and compile an application program containing embedded SQL statements before you can run it. Precompiling such programs is done by an SQL precompiler.

The SQL precompiler scans each statement of the application program source and does the following things:

- Looks for SQL statements and for the definition of host variable names
- Verifies that each SQL statement is valid and free of syntax errors
- Validates the SQL statements using the description in the database
- Prepares each SQL statement for compilation in the host language
- Produces information about each precompiled SQL statement

Application programming statements and embedded SQL statements are the primary input to the SQL precompiler. The SQL precompiler assumes that the host language statements are syntactically correct. If the host language statements are not syntactically correct, the precompiler might not correctly identify SQL statements and host variable declarations.

The SQL precompile process produces a listing and a temporary source file member. It can also produce the SQL package depending on what is specified for the OPTION and RDB parameters of the precompiler command.

Related reference

Compiling an application program

The Db2 for i precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

Listing

The output listing is sent to the printer file specified by the PRTFILE parameter of the CRTSQL*xxx* command.

The following items are written to the printer file:

- Precompiler options

  This is a list of all the options specified with the CRTSQL*xxx* command and the date the source member was last changed.
- Precompiler source

  This output is produced if the *SOURCE option is used for non-ILE precompiles or if the OUTPUT(*PRINT) parameter is specified for ILE precompiles. It shows each precompiler source statement with its record number assigned by the precompiler, the sequence number (SEQNBR) you see when using the source entry utility (SEU), and the date the record was last changed.
- Precompiler cross-reference

  This output is produced if *XREF was specified in the OPTION parameter. It shows the name of the host variable or SQL entity (such as tables and columns), the record number where the name is defined, what the name is defined, and the record numbers where the name occurs.
- Precompiler diagnostic list

  This output supplies diagnostic messages, showing the precompiler record numbers of statements in error.

Temporary source file member

Source statements processed by the precompiler are written to QSQLTEMP in the QTEMP library (QSQLTEMP1 in the QTEMP library for programs created using **`CRTSQLRPGI`**).

In your precompiler-changed source code, SQL statements have been converted to comments and calls to the SQL interface modules: QSQROUTE, QSQLOPEN, QSQLCLSE, and QSQLCMIT. The name of the temporary source file member is the same as the name specified in the PGM parameter of CRTSQL*xxx*. This member cannot be changed before being used as input to the compiler.

QSQLTEMP or QSQLTEMP1 can be moved to a permanent library after the precompile, if you want to compile at a later time. If you change the records of the temporary source file member, the compile attempted later will fail.

SQL package creation
An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Related concepts

Working with SQL packages
An *SQL package* is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on a server.

Related reference

Compiling an application program
The Db2 for i precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

Binding an application
Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called *binding*.

Precompiler commands
The IBM DB2 Query Manager and SQL Development Kit for i licensed program has seven precompiler commands, one for each of the host languages.

Host language	Command
PL/I PRPQ	**CRTSQLPLI**
ILE C language	**CRTSQLCI**
COBOL/400 language	**CRTSQLCBL**
ILE COBOL language	**CRTSQLCBLI**
FORTRAN/400 language	**CRTSQLFTN**
RPG III (part of RPG/400 language)	**CRTSQLRPG**
ILE RPG language	**CRTSQLRPGI**

A separate command for each language exists so each language can have parameters that apply only to that language. For example, the options *APOST and *QUOTE are unique to COBOL. They are not included in the commands for the other languages. The precompiler is controlled by parameters specified when it is called by one of the SQL precompiler commands. The parameters specify how the input is processed and how the output is presented.

You can precompile a program without specifying anything more than the name of the member containing the program source statements as the PGM parameter (for non-ILE precompiles) or the OBJ parameter (for ILE precompiles) of the CRTSQL*xxx* command. SQL assigns default values for all precompiler parameters (which might, however, be overridden by any that you explicitly specify).

The following list briefly describes parameters common to all the CRTSQLxxx commands that are used to support distributed relational database.

RDB
Specifies the name of the relational database where the SQL package option is to be created. If *NONE is specified, then the program or module is not a distributed object and the **Create Structured Query Language Package (CRTSQLPKG)** command cannot be used. The relational database name can be the name of the local database.

RDBCNNMTH
Specifies the type of semantics to be used for CONNECT statements: remote unit of work (RUW) or distributed unit of work (DUW) semantics.

SQLPKG
Specifies the name and library of the SQL package.

USER
Specifies the user name sent to the remote system when starting the conversation. This parameter is used only if a conversation is started as part of the precompile process.

PASSWORD
Specifies the password to be used on the remote system when starting the conversation. This parameter is used only if a conversation is started as part of the precompile process.

REPLACE
Specifies if any objects created as part of the precompile process should be able to replace an existing object.

The following example creates a COBOL program named INVENT and stores it in a library named SPIFFY. The SQL naming convention is selected, and every row selected from a specified table is locked until the end of the unit of recovery. An SQL package with the same name as the program is created on the remote relational database named KC000.

```
CRTSQLCBL PGM(SPIFFY/INVENT) OPTION(*SRC *XREF *SQL)
          COMMIT(*ALL) RDB(KC000)
```

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Compiling an application program

The Db2 for i precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

The compiler command is run specifying the program name, source file name, precompiler created source member name, text, and user profile. Other parameters are also passed to the compiler, depending on the host language.

Related concepts

Precompiling programs with SQL statements
You must precompile and compile an application program containing embedded SQL statements before you can run it. Precompiling such programs is done by an SQL precompiler.

Embedded SQL programming

Related reference

SQL package creation
An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Binding an application

Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called *binding*.

The result of binding is an access plan. The *access plan* is a control structure that describes the actions necessary to satisfy each SQL request. An access plan contains information about the program and about the data the program intends to use. For distributed relational database work, the access plan is stored in the SQL package and managed by the system along with the SQL package.

SQL automatically attempts to bind and create access plans when the result of a successful compilation is a program or service program object. If the compilation is not successful or the result of a compilation is a module object, access plans are not created. If, at run time, the database manager detects that an access plan is not valid or that changes have occurred to the database that might improve performance (for example, the addition of indexes), a new access plan is automatically created. If the server is not a IBM i product, then a bind must be done again using the Create Structured Query Language Package (CRTSQLPKG) command. Binding performs these tasks:

- Revalidates the SQL statements using the description in the database.

During the bind process, the SQL statements are checked for valid table, view, and column names. If a referred to table or view does not exist at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, a negative SQLCODE is returned.

- Selects the access paths needed to access the data your program wants to process.

  In selecting an access path, indexes, table sizes, and other factors are considered when SQL builds an access plan. The bind process considers all indexes available to access the data and decides which ones (if any) to use when selecting a path to the data.

- Attempts to build access plans.

  If all the SQL statements are valid, the bind process builds and stores access plans in the program.

If the characteristics of a table or view your program accesses have changed, the access plan might no longer be valid. When you attempt to use an access plan that is not valid, the system automatically attempts to rebuild the access plan. If the access plan cannot be rebuilt, a negative SQLCODE is returned. In this case, you might have to change the program's SQL statements and reissue the CRTSQL*xxx* command to correct the situation.

For example, if a program contains an SQL statement that refers to COLUMNA in TABLEA and the user deletes and recreates TABLEA so that COLUMNA no longer exists, when you call the program, the automatic rebind is unsuccessful because COLUMNA no longer exists. You must change the program source and reissue the CRTSQL*xxx* command.

Related concepts

Working with SQL packages
An *SQL package* is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on a server.

Related reference

SQL package creation
An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Create Structured Query Language Package (CRTSQLPKG) command

Testing and debugging

Testing and debugging distributed SQL programs is similar to testing and debugging local SQL programs, but certain aspects of the process are different.

If applications are coded so that the relational database names can easily be changed by recompiling the program, by changing the input parameters to the program, or by making minor modifications to the program source, most testing can be accomplished using a single system.

After the program has been tested against local data, the program is then made available for final testing on the distributed relational database network. Consider testing the application locally on the system that will be the server when the application is tested over a remote connection, so that only the program needs to be moved when the testing moves into a distributed environment.

Debugging a distributed SQL program uses the same techniques as debugging a local SQL program. You use the Start Debug (STRDBG) command to start the debugger and to put the application in debug mode. You can add breakpoints, trace statements, and display the contents of variables.

However, to debug a distributed SQL program, you must specify the value of *YES for the UPDPROD parameter. This is because IBM i distributed relational database support uses files in library QSYS and QSYS is a production library. This allows data in production libraries to be changed on the client. Issuing the Start Debug (STRDBG) command on the client only puts the client job into debug mode, so your ability to manipulate data on the server is not changed.

While in debug mode on the client, informational messages are entered in the job log for each SQL statement run. These messages give information about the result of each SQL statement. A list of SQL return codes and a list of error messages for distributed relational database are provided in the Troubleshooting topic.

Informational messages about how the system maximizes processing efficiency of SQL statements are also issued as a result of being in debug mode. Because any maximization occurs at the server, these types of messages do not appear in the client job log. To get this information, the server job must be put in debug mode.

A relatively easy way to start debug mode on the system, if you are using TCP/IP, is to use the QRWOPTIONS data area. However, you cannot specify a specific program to debug with this facility. For details on setup, see QRWOPTIONS data area usage. The data area can be used not only to start debug, but to start job traces, request job logs, display job output and do other things as well. You can even do the QRWOPTIONS setup on an IBM i client, and have the options shadowed to a IBM i platform.

If both the client and server are IBM i products, and they are connected with APPC, you can use the Submit Remote Command (SBMRMTCMD) command to start the debug mode in a server job. Create a DDM file as described in the Setting up DDM files topic. The communications information in the DDM file must match the information in the relational database directory entry for the relational database being accessed. Then issue this command:

```
SBMRMTCMD CMD('STRDBG UPDPROD(*YES)') DDMFILE(ddmfile name)
```

The SBMRMTCMD command starts the server job if it does not already exist and starts the debug mode in that job. Use one of the methods for monitoring relational database activity to examine the server job log to find the job.

The following method for putting the server job into debug mode works with any client and a Db2 for i server with certain restrictions. It depends on being able to pause after the application makes a connection to do setup. It also assumes that what you want to trace or otherwise debug occurs after the connection is established.

- Sign on to the client and find the server job.
- Issue the Start Service Job (STRSRVJOB) command from the interactive job (the job you are using to find the server job) as shown:

  ```
  STRSRVJOB (job-number/user-ID/job-name)
  ```

  The job name for the STRSRVJOB command is the name of the server job. Issuing this command lets you issue certain commands from your interactive job that affect the server job. One of these commands is the Start Debug (STRDBG) command.
- Issue the STRDBG command using a value of *YES for the UPDPROD parameter in the interactive job. This puts the server job into debug mode to produce debug messages on the server job log.

To end this debug session, either end your interactive job by signing off or use the End Debug (ENDDBG) command followed by the End Service Job (ENDSRVJOB) command.

Because the server job must be put into debug before the SQL statements are run, the application might need to be changed to allow you time to set up debug on the server. The server job starts as a result of the application connecting to the server. Your application can be coded to enter a wait state after connecting to the server until debug is started on the server.

If you can anticipate the prestart job that will be used for a TCP/IP connection before it occurs, such as when there is only one waiting for work and there is no interference from other clients, you do not need to introduce a delay.

Related concepts

Monitoring relational database activity
You can use control language (CL) commands, all of which provide similar information, but in different ways, to give you a view of work on the IBM i operating system.

Troubleshooting DRDA and DDM
When a problem occurs accessing a distributed relational database, determine the nature of the problem and whether it is a problem with the application or a problem with the local or remote system.

QRWOPTIONS data area

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related tasks

Setting up DDM files
The IBM i implementation of Distributed Relational Database Architecture (DRDA) support uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Related reference

End Debug (ENDDBG) command

End Service Job (ENDSRVJOB) command

Start Debug (STRDBG) command

Start Service Job (STRSRVJOB) command

SQL CALL statement (stored procedures)
The SQL CALL statement is not actually specific to distributed relational databases, but a discussion of it is included here because its main value is in distributing application logic and processing.

Submit Remote Command (SBMRMTCMD) command

Program references
When a program is created, the IBM i licensed program stores information about all collections, tables, views, SQL packages, and indexes referred to in SQL statements in an SQL program.

You can use the **Display Program References (DSPPGMREF)** command to display all object references in the program. If the SQL naming convention is used, the library name is stored in one of the following ways:

- If the SQL name is fully qualified, the collection name is stored as the name qualifier.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is not specified, the authorization ID of the statement is stored as the name qualifier.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is specified, the collection name specified on the DFTRDBCOL parameter is stored as the name qualifier.

If the system naming convention is used, the library name is stored in one of the following ways:

- If the object name is fully qualified, the library name is stored as the name qualifier.
- If the object is not fully qualified, and the DFTRDBCOL parameter is not specified, *LIBL is stored.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is specified, the collection name specified on the DFTRDBCOL parameter is stored as the name qualifier.

Related reference

Display Program References (DSPPGMREF) command

Working with SQL packages

An *SQL package* is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on a server.

These control structures are used by the server at run time when the application requests data using the SQL statement.

There is no SQL statement for SQL package creation. You can create an SQL package in two ways:

- Using the CRTSQL*xxx* command with a relational database name specified in the RDB parameter
- Using the **Create SQL Package (CRTSQLPKG)** command

It is recommended that user use one of the options above to create an SQL package, however an implicit create package will be attempted if the package is not found. If the user profile exists at the application server, the owner of the package will be set to the owner of the application. If the user profile does not exist at the application server, the owner of the package will be set to the current user profile used to connect .

Related reference

Distributed relational database statements
The statements included with the SQL language specifically support a distributed relational database.

SQL package creation
An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Binding an application
Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called *binding*.

Using the Create SQL Package (CRTSQLPKG) command

You can enter the Create SQL Package (CRTSQLPKG) command to create an SQL package from a compiled distributed relational database program. You can also use this command to replace an SQL package that was created previously.

A new SQL package is created on the relational database defined by the RDB parameter. The new SQL package has the same name and is placed in the same library as specified on the PKG parameter of the CRTSQL*xxx* command.

You do not need the IBM DB2 Query Manager and SQL Development Kit for i licensed program to create an SQL package on a server system.

Related reference

Default collection name
You can specify a default collection name to be used by an SQL program by supplying this name for the DFTRDBCOL parameter on the CRTSQL*xxx* command when you precompile the program.

Create Structured Query Language Package (CRTSQLPKG) command

Managing an SQL package

After an SQL package is created, you can manage it the same way as you manage other objects on the IBM i operating system, with some restrictions.

You can save and restore an SQL package, send it to other systems, and grant and revoke a user's authority to the package. You can also delete it by entering the **Delete Structured Query Language Package (DLTSQLPKG)** command or the DROP PACKAGE SQL statement.

When a distributed SQL program is created, the name of the SQL package and an internal consistency token are saved in the program. These are used at run time to find the SQL package and verify that the SQL package is correct for this program. Because the name of the SQL package is critical for running distributed SQL programs, an SQL package cannot be moved, renamed, duplicated, or restored to a different library.

Displaying information about an SQL package

You can use the **Print SQL Information (PRTSQLINF)** command to display information that is stored in an SQL package.

The information that is displayed includes commitment control level, options, static statements in the package. For instance, the following command displays the SQL package PARTS1 in the SPIFFY collection:

```
PRTSQLINF OBJ(SPIFFY/PARTS1) OBJTYPE(*SQLPKG)
```

Deleting an SQL package using the Delete SQL Package (DLTSQLPKG) command

You can use the Delete Structured Query Language Package (DLTSQLPKG) command to delete one or more SQL packages. You must run the DLTSQLPKG command on the system where the SQL package being deleted is located.

You must have *OBJEXIST authority for the SQL package and at least *EXECUTE authority for the collection where it is located.

There are also several SQL methods to drop packages:

- If you have the IBM DB2 Query Manager and SQL Development Kit for i licensed program installed, use Interactive SQL to connect to the server and then drop the package using the SQL DROP PACKAGE statement.
- Run an SQL program that connects and then drops the package.
- Use Query Management to connect and drop the package.

The following command deletes the SQL package PARTS1 in the SPIFFY collection:

```
DLTSQLPKG  SQLPKG(SPIFFY/PARTS1)
```

To delete an SQL package on a remote IBM i operating system, use the Submit Remote Command (SBMRMTCMD) command to run the Delete Structured Query Language Package (DLTSQLPKG) command on the remote system. You can also use display station pass-through to sign on the remote system to delete the SQL package. If the remote system is not an IBM i operating system, pass through to that system using a remote workstation program and then submit the Delete SQL Package command local to that system.

Related reference

Delete Structured Query Language Package (DLTSQLPKG) command

Submit Remote Command (SBMRMTCMD) command

Modifying package authorizations

For any programs created on IBM i, you can change the users that are authorized to use that package.

This can be done using SQL's GRANT and REVOKE statements:

- GRANT ALL PRIVILEGES ON TABLE table-name TO user (possibly PUBLIC for user)
- GRANT EXECUTE ON PACKAGE package-name (usually the IBM i program name) TO user (possibly PUBLIC for user)

It can also be done entering GRTOBJAUT and RVKOBJAUT commands from the command line.

Using the SQL DROP PACKAGE statement

The DROP PACKAGE statement includes the PACKAGE parameter for distributed relational database. You can issue the DROP PACKAGE statement by embedding it in a program or by using interactive SQL.

When you issue a DROP PACKAGE statement, the SQL package and its description are deleted from the server. This has the same result as a Delete Structured Query Language Package (DLTSQLPKG) command entered on a local system. No other objects dependent on the SQL package are deleted as a result of this statement.

You must have the following privileges on the SQL package to successfully delete it:

- The system authority *EXECUTE on the referenced collection
- The system authority *OBJEXIST on the SQL package

The following example shows how the DROP PACKAGE statement is issued:

```
DROP PACKAGE  SPIFFY.PARTS1
```

A program cannot issue a DROP PACKAGE statement for the SQL package it is currently using.

Related reference

SQL DROP statement

Application development for DDM

This topic lists the programming considerations for DDM, including the CL commands, languages and utilities that you can use for DDM.

DDM files and SQL

You can use IBM i distributed data management (DDM) support to help you do some distributed relational database tasks within a program that also uses SQL distributed relational database support.

It might be faster, for example, for you to use DDM and the Copy File (CPYF) command to get a large number of records rather than an SQL FETCH statement. Also, DDM can be used to get external file descriptions of the remote system data brought in during compilation for use with the distributed relational database application. To do this, you need to use DDM as described in "Initial setup" on page 47.

The following example shows how you can add a relational database directory entry and create a DDM file so that the same job can be used on the server and client.

Notes:

- Either both connections must be protected or both connections must be unprotected for the conversation to be shared.
- By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
Relational Database Directory:

ADDRDBDIRE    RDB(KC000) +
              RMTLOCNAME(KC000)
              TEXT('Kansas City regional database')
```

```
DDM File:

CRTDDMF  FILE(SPIFFY/UPDATE)
         RMTFILE(SPIFFY/INVENTORY)
         RMTLOCNAME(KC000)
         TEXT('DDM file to update local orders')
```

Here is a sample program that uses both the relational database directory entry and the DDM file in the same job on the remote system:

```
CRTSQLxxx PGM(PARTS1) COMMIT(*CHG)  RDB(KC000) RDBCNNMTH(*RUW)

    PROC :PARTS1;
    OPEN  SPIFFY/UPDATE;
       .
       .
       .
    CLOSE SPIFFY/UPDATE;
       .
       .
       .
    EXEC SQL
      SELECT * INTO :PARTAVAIL
               FROM INVENTORY
               WHERE ITEM = :PARTNO;
    EXEC SQL
      COMMIT;
       .
       .
       .
    END PARTS1;
```

Related reference

Copy File (CPYF) command

Using language, utility, and application support for DDM

This topic describes the language, utility, and application program support that is provided for IBM i DDM.

Language-specific information concerning access to Customer Information Control System (CICS) for Virtual Storage files is in topic IBM i-to-CICS considerations with DDM.

Programming language considerations for DDM

DDM is supported by ILE C, ILE COBOL, ILE RPG, PL/I PRPQ and control language (interactive and compiled forms).

DDM considerations for all languages

DDM files can be used as data files or source files by high-level language (HLL) programs.

However, for CL, data description specifications (DDS), and BASIC, if a DDM file is to be used as a source file, the server system must be an IBM i or a System/38, and the file referred to by the DDM file must be defined on the target IBM i or System/38 as a source file. That is, the remote file must have been created either by the **`Create Source Physical File (CRTSRCPF)`** command or as FILETYPE(*SRC) by the **`Create Physical File (CRTPF)`** command. These restrictions are not enforced by the ILE RPG, ILE COBOL, and ILE C compilers, which allow source files to be used from both IBM i and non-IBM i server systems.

If a source file *member* name is specified when the server system is not an IBM i or a System/38, all the HLL compilers end compilation if the name of the source member specified on the SRCMBR parameter is different from the name of the DDM file specified on the SRCFILE parameter.

If programs that accessed local files are to access remote files, certain restrictions might require that a program be changed and recompiled. And, if the server system is not an IBM i or a System/38, externally described data must, in some cases, reside on the local (client) system. All of these restrictions are described under the topic Program modification requirements for DDM.

If the server system is not an IBM i or a System/38, the number of records returned in the open feedback might not be valid.

If you do not specify a library name for the SRCFILE parameter, the first file found in the user's library list with the same name as the file you specified for the SRCFILE parameter is used as the source file.

HLL program input and output operations with IBM i DDM

The high-level language operations are supported by DDM for keyed or nonkeyed operations.

See the information in the following tables.

Table 7. High-level language operations supported by DDM for keyed or nonkeyed operations

	High-level languages			
IBM i database operation	**ILE RPG programming language**	**ILE COBOL programming language**	**BASIC**	**PL/I**
Open file	OPEN	OPEN	OPEN	OPEN
Query file				
Read (keyed access)	CHAIN (key)	READ INVALID KEY	READ KEY	READ EQUAL
Read first/last[1]	*LOVAL *HIVAL	READ FIRST LAST	READ FIRST LAST	READ FIRST LAST
Read next	READ READE[2]	READ <NEXT> AT END	READ	READ NEXT
Read previous	READP	READ PRIOR AT END	READ PRIOR	READ PRV

Table 7. High-level language operations supported by DDM for keyed or nonkeyed operations (continued)

	High-level languages			
IBM i database operation	ILE RPG programming language	ILE COBOL programming language	BASIC	PL/I
Read next or previous[3] Next equal Previous equal Next unique Previous unique			READ = =, PRIOR	READ NXTEQL PRVEQL NXTUNQ PRVUNQ
Read (relative to start)[4]	CHAIN (rrn)	READ RELATIVE KEY	READ REC=	READ KEY
Release record lock	EXCPT or next I/O op	(next I/O op)	(next I/O op)	(next I/O op)
Force end of data	FEOD			
Position file[5]	SETGT SETLL	START KEY GREATER KEY NOT LESS KEY EQUAL	RESTORE	
Update record	UPDAT	REWRITE[6]	REWRITE	REWRITE
Write record	WRITE/ EXCPT	WRITE[6]	WRITE	WRITE
Delete record	DELET	DELETE[6]	DELETE	DELETE
Close file	CLOSE	CLOSE	CLOSE	CLOSE

[1] For the ILE RPG language, if the keyed access path of a file specifies DESCENDING, then *LOVAL gets the last record in the file and *HIVAL gets the first record in the file.

[2] For duplicate keyed files, the ILE RPG language performs a READ NEXT operation and compares the key of the returned record to determine if the record qualifies. If so, the record is returned to the program; if not, an end-of-file indication is returned.

[3] If the remote file is on a non-IBM i, these operations cannot be performed using DDM.

[4] An IBM i application program can open a *keyed* access open data path to a file and then access its records using both keyed and *relative record* access methods. Although DDM supports the *combined-access* access method, a server system (such as System/36) might not. In this case, the IBM i can do relative record accessing of a keyed file on a non-IBM i target server if the server system supports the *combined-by-record-number* access method and if the DDM file specifies that method. The combined-by-record-number access method is specified on an IBM i as ACCMTH(*ARRIVAL *BOTH) on the **Create DDM File (CRTDDMF)** command. If these values are not specified for the DDM file and the server system does not support the combined-access access method, relative record operations to a keyed file are rejected.

[5] Positioning operations (SET*xx* in the ILE RPG language, or START in the ILE COBOL language) do not return the record data to the application program. These operations also cause the file to be opened for random processing.

[6] ILE COBOL operations that change indexed or relative files can lock the record before the operation to make the record eligible.

Table 8. High-level language operations supported by DDM for keyed or nonkeyed operations

	High-level languages	
IBM i database operation	**CL**	**ILE C programming language**
Open file	OPNDBF	FOPEN, FREOPEN
Query file	OPNQRYF	
Read (keyed access)		
Read first/last		
Read next	RCVF	FREAD, FGETC
Read previous		
Read next or previous: Next equal Previous equal Next unique Previous unique		
Read (relative to start)		
Release record lock		(next I/O op)
Force end of data		FFLUSH
Position file	POSDBF	FSEEK, FSETPOS
Update record		FWRITE, FPUTC, FFLUSH
Write record		FWRITE, FPUTC, FFLUSH
Delete record		
Close file	CLOF	FCLOSE

Commitment control support for DDM
IBM i applications can commit or roll back transactions on remote IBM i operating systems.

However, DDM does not support the IBM i journaling commands (**CRTJRN**, **CRTJRNRCV**, and **STRJRNPF**). Before running applications, a user must create a journal on the target IBM i operating systems for recoverable resources to be used under commitment control, start journaling the physical files that are to be opened under commitment control, and issue the **Start Commitment Control (STRCMTCTL)** command on the client system. The **STRCMTCTL** command does not Support the **Notify Object (NTFOBJ)** command for DDM files. Another way to set up journaling on the remote system is to use the **SBMRMTCMD** DDM support to submit the journal commands to the server system to journal the remote files.

For DDM conversations to use two-phase commitment control, the DDM conversations need to be protected. For DDM conversations to be protected, the appropriate DDM file must have been created with the protected conversation (PTCCNV) parameter set to *YES. The protected conversation (PTCCNV) parameter allows a values of *YES when the address type is *SNA and *IP. To set PTCCNV(*YES)) for *IP address type, specify *RDB in the remote location (RMTLOCNAME) parameter and specify an RDB in the relational database (RDB) parameter. The RDB relational database directory entry can be either specify *IP or *SNA. Please refer to the Add RDB Directory Entry (ADDRDBDIRE) and Work with Relational Database Directory Entries (WRKRDBDIRE) commands to add a RDB directory entry.

Using DDM files with commitment control
You need to consider these restrictions when you work with the DDM files that are opened under commitment control.

- If more than one DDM file (with PTCCNV(*NO)) is opened under commitment control, the following items must be the same for each file:

  - Remote location name
  - Local location name
  - Device
  - Mode
  - Remote network ID
  - Transaction program name (TPN)
  - User ID
  - Activation group number
  - Open scope

  The exception to this rule is when all of the DDM files opened under commitment control are scoped to the job level. In this case, the activation group numbers are ignored and do not need to match.
- If a DDM file and a remote SQL object (Distributed Relational Database Architecture, DRDA) are running under commitment control (with PTCCNV(*NO)), the following items must be the same for the file and object:
  - Remote location name
  - Local location name
  - Device
  - Mode
  - Remote network ID
  - TPN
  - User ID
  - Activation group number
  - Open scope
- If a DDM file (with PTCCNV(*YES)) is being opened for output, update, or delete (not opened for input only), then there cannot be any one-phase DDM or DRDA conversations active.
- If a DDM file (with PTCCNV(*YES)) is being used, it must point to a target IBM i that supports two-phase commitment control protocols.
- DDM files (with PTCCNV(*NO)) and local database files cannot be opened under commitment control at the same time within the same activation group.
- DDM files (with PTCCNV(*NO)) and local database files cannot be opened under commitment control at the same time within the same job if commitment control is scoped to the job level.
- To open a DDM file under commitment control and scope it to the job level, you must have specified CMTSCOPE(*JOB) on the **`Start Commitment Control (STRCMTCTL)`** command.
- You cannot use the **`Submit Remote Command (SBMRMTCMD)`** command to call programs that expect commitment control to be scoped to the job level. Because commitment control is always scoped to the activation group level in DDM target jobs, the program fails.
- The **`SBMRMTCMD`** command should not be used to start or end commitment control.
- The server system specified from the system working under commitment control must be another IBM i.

Note: If the communications line fails during a COMMIT operation, the client and server systems will do a ROLLBACK operation. However, the server system might successfully complete the COMMIT operation before the line fails, but the client system will always do a ROLLBACK operation.

Table 9. High-level language commit and rollback commands

Operation	ILE RPG programming language	ILE COBOL programming language	PL/I	CL	ILE C programming language
Commit changes in transaction	COMMIT	COMMIT	PLICOMMIT	COMMIT	_Rcommit
Cancel entire transaction	ROLBK	ROLLBACK	PLIROLLBACK	ROLLBACK	_Rrollback

ILE RPG considerations for DDM
ILE RPG programs and automatic report programs can both refer to DDM files. Generally, DDM file names can be specified in ILE RPG anywhere a database file name can be specified, for both IBM i and non-IBM i server systems.

- DDM file names can be specified on the **Create RPG Program (CRTRPGPGM)** and **Create Auto Report Program (CRTRPTPGM)** commands:
  - To access remote files containing source statements, on an IBM i or a non-IBM i, a DDM file name can be specified on the SRCFILE parameter, and a member name can be specified on the SRCMBR parameter.
    - For IBM i or System/38 server systems, a remote IBM i or System/38 source file (and, optionally, member) can be accessed in the same manner as a local source file and member.
    - For non-IBM i server systems, a remote source file can be accessed if both the PGM and SRCMBR parameter defaults are used on either command. Or, if a member name is specified, it must be the same as the DDM file name specified on the SRCFILE parameter. (The same is true for member names specified either on the /COPY statement of the input specifications used to create an automatic report program or as used by the compiler to include source specifications.)
  - To place the compiler listing in a database file on a server system, a DDM file name can be specified on the PRTFILE parameter of either command.
- A DDM file name and member name can be specified on the OUTFILE and OUTMBR parameters of the **CRTRPTPGM** command, but before the output produced by the command can be stored in the remote file referred to by the DDM file, the remote file must already exist. Also, as with local files, the record format of the remote file must match the required OUTFILE parameter format. Generally, this means that the server system must be an IBM i operating system or a System/38 product.

When an ILE RPG program opens a DDM file on the client system, the following types of I/O operations can be performed on the remote file at the server system, for both IBM i and non-IBM i targets: CHAIN, CLOSE, DELET, EXCPT, FEOD, OPEN, READ, READE, READP, SETGT, SETLL, UPDAT, and WRITE.

Other considerations are:

- If the DDM file is declared in the program to be externally described, the ILE RPG compiler copies the external descriptions of the remote file referred to into the program at compile time. However, if the remote file is not on an IBM i or a System/38, the field declares for the record descriptions do not have meaningful names. Instead, all of the field names are declared as F*nnnnn* and the key fields are declared as K*nnnnn*.

  A recommended method for describing remote files, when the target is not an IBM i or a System/38, is to have the data description specifications (DDS) on the local system and enter a **Create Physical File (CRTPF)** command or a **Create Logical File (CRTLF)** command on the local system. Compile the program using the local file name. Ensure that the remote system's file has the corresponding field types and field lengths.

  To access the remote file, use the **Override with Database File (OVRDBF)** command preceding the program, for example:

  ```
  OVRDBF FILE(PGMFIL) TOFILE(DDMFIL) LVLCHK(*NO)
  ```

- A DDM file is also valid as the file specified in the ILE RPG program that will be used implicitly in the ILE RPG logic cycle.
- A record format name, if used, must match the DDM file name when the server system is not an IBM i or a System/38.
- An ADDROUT file created on a System/36 cannot be used on an IBM i. IBM i System/36-Compatible RPG II uses 3-byte ADDROUT files, and ILE RPG programming language on an IBM i operating system and System/38 uses 4-byte ADDROUT files.

ILE COBOL considerations for DDM

ILE COBOL programs can refer to DDM files. Generally, DDM file names can be specified in ILE COBOL anywhere a database file name can be specified, for server systems that run IBM i or do not run IBM i alike.

- DDM file names can be specified on the Create COBOL Program (CRTCBLPGM) command:
  - To access remote files containing source statements, a DDM file name can be specified on the SRCFILE parameter, and a member name can be specified on the SRCMBR parameter.
    - For IBM i or System/38 server systems, a remote IBM i or System/38 source file (and, optionally, member) can be accessed in the same manner as a local source file and member.
    - For server systems that do not run IBM i, a remote source file can be accessed if both the PGM and SRCMBR parameter defaults are used on the CRTCBLPGM command. Or, if a member name is specified, it must be the same as the DDM file name specified on the SRCFILE parameter.
  - To place the compiler listing in a database file on a server system, a DDM file name can be specified on the PRTFILE parameter of the CRTCBLPGM command.
- DDM file names can be specified as the input and output files for the ILE COBOL SORT and MERGE operation. (The work file for this operation cannot be a DDM file.)
- A DDM file can be used in the ILE COBOL COPY statement when the DDS option on that statement is used to copy one or all of the externally described record formats from the remote file referred to by the DDM file into the program being compiled. If this is done when the remote file is not on an IBM i or a System/38, the field declares for the record descriptions will not have meaningful names. Instead, all of the field names are declared as F*nnnnn* and the key fields are declared as K*nnnnn*.

  A recommended method for describing remote files, when the target is not an IBM i or a System/38, is to have the data description specifications (DDS) on the local system and enter a Create Physical File (CRTPF) command or a Create Logical File (CRTLF) command on the local system. Compile the program using the local file name. Ensure that the remote system's file has the corresponding field types and field lengths.

  To access the remote file, use the Override with Database File (OVRDBF) command preceding the program, for example:

  ```
  OVRDBF FILE(PGMFIL) TOFILE(DDMFIL) LVLCHK(*NO)
  ```
- DDM file names can be specified on a COPY statement:
  - If you do not specify the library name with the file name, the first file found with that file name in the user's library list is used as the include file.
  - If the server system is not an IBM i or a System/38, a DDM file name can be specified as the include file on a COPY statement, but the member name must be the same as the DDM file name.
- If the server system is a System/36, ILE COBOL programming language cannot be used to open a DDM file for output if the associated remote file has logical files built over it. For System/36 files with logical files, the open operation (open output) fails because ILE COBOL programming language attempts to clear the file before using it.

When an ILE COBOL program opens a DDM file on the client system, the following statements can be used to perform I/O operations on the remote file at the server system: CLOSE, DELETE, OPEN, READ, REWRITE, START, and WRITE.

Direct file support with ILE COBOL

The IBM i operating system does not support direct files as one of its file types. However, an ILE COBOL program on IBM i can specify that a file be accessed as a direct file.

The IBM i operating system normally creates direct files as sequential files. An ILE COBOL program on the IBM i defines a file as a direct file by specifying RELATIVE on the SELECT statement. If the program is to open the file for output only (by specifying OUTPUT on the OPEN statement), the file must be created with deleted records and contain no active records. This is also the file's condition when a client system that does not run IBM i (such as System/36) uses DDM to create or clear the direct file on an IBM i, assuming that the file is created as described in the following paragraphs.

IBM i and System/38 support sequential and keyed file types. DDM recognizes sequential, keyed, and direct file types. If a system other than IBM i is to create a direct file on an IBM i using DDM, the DDM architecture command Create Direct File (CRTDIRF) is used.

When the CRTDIRF architecture command is issued from an operating system other than IBM i to create the file, the file is created as a physical file and is designated as a direct file so that, for subsequent direct file access by client systems that do not run IBM i, it is identifiable to the other system as a direct file. If the file is not created in this way, an IBM i cannot later determine whether the file is a direct file or a sequential file, again, because an IBM i does not have direct files as one of its file types.

Therefore, if an ILE COBOL program on a system other than an IBM i or a System/38 needs to access an IBM i or a System/38 file in a direct mode (that is, by relative record number) for output, the file must have been created by the CRTDIRF architecture command.

To support direct files on the IBM i operating system for output only, the ILE COBOL OPEN statement clears and prepares a member of a file that is opened. Therefore, existing IBM i or System/38 files can be accessed by using DDM files by ILE COBOL programs on other IBM i operating systems or System/38 platforms. For server systems that do not run IBM i, relative files opened for output must be defined as direct files, or an error occurs.

In summary:

- If a file is created on the local IBM i as a direct file by a program or user from a system other than IBM i, the file can be accessed as a direct file by an ILE COBOL program from a remote client system that does not run IBM i.
- If a file is created on the local IBM i by a program or user on the same IBM i, it cannot be accessed as a direct file by a system other than IBM i because the server system that runs IBM i cannot determine, in this case, whether the file is a direct or sequential file.
- Any files created by a remote system can be used locally.

PL/I considerations for DDM

Compiled programs can refer to DDM files. In addition, DDM file names can be specified on the **Create PL/I Program (CRTPLIPGM)** command.

- A DDM file name can be specified on the SRCFILE parameter, and a member name can be specified on the SRCMBR parameter, but only if the remote source file is on an IBM i operating system or a System/38 platform. The same is true for specifying DDM file and member names on the %INCLUDE source directive statement. If the remote file referred to by the DDM file is not on an IBM i operating system or a System/38 platform, an error occurs if a DDM file name is specified on the CRTPLIPGM command or %INCLUDE statement.
- When a DDM file is accessed as the source file for a program, the margins used in the compilation of the source are the default values of 2 and 72. No other margin values can be specified.
- If a %INCLUDE DDS directive statement specifies the name of a DDM file, the record descriptions of the remote file are included in the compiled program. However, if the remote file is not on an IBM i operating system or a System/38 platform, the field declares for the record descriptions do not have meaningful names. Instead, all of the field names are declared as F*nnnnn* and the key fields are declared as K*nnnnn*.

A DDM file can be used to refer to remote record files or remote stream files. When a program opens a DDM file on the client system, the following types of statements can be used to perform I/O operations on the remote file at the server system, for both IBM i and non-IBM i targets: OPEN, CLOSE, READ, WRITE, REWRITE, and DELETE statements for processing record files, and GET and PUT statements for processing stream files.

Another consideration is if the server system is not an IBM i operating system or a System/38 product, the POSITION parameter on a keyed READ statement to read from a remote file does not work if a value of NXTEQL, PRVEQL, NXTUNQ, or PRVUNQ is specified for the parameter. (The values of NEXT, PREVIOUS, FIRST, and LAST do work.) All the values are valid if the server system is an IBM i operating system or a System/38 product.

CL command considerations for DDM

Both compiled control language (CL) programs and interactively entered CL commands can refer to DDM files.

Generally, DDM file names can be specified in CL commands anywhere a database file name can be specified for both IBM i and non-IBM i server systems. But there are some limitations.

Listed here are some examples of where DDM file names can be specified:

- DDM file names can be specified on many of the database file-related commands, such as the copy, display, and override file commands.
- DDM file names can be specified on the create file commands to access remote *source* files, but only if the server system is an IBM i operating system or a System/38 product. A DDM file name can be specified on the SRCFILE parameter, and a member name can be specified on the SRCMBR parameter. If the remote source file referred to by the DDM file is not on an IBM i operating system or a System/38 product, an error occurs. The considerations for remote IBM i or System/38 source members are the same as for local source members.
- DDM file names can be specified on the FILE parameter of the **Declare File (DCLF)** command.

When a DDM file name is specified, some commands act on files on the client system, some act on target files, and some parameter values allow you to specify either a source or target file.

ILE C considerations for DDM

ILE C programs can refer to DDM files.

Generally, DDM file names can be specified in ILE C programming language anywhere a database file name can be specified, for both IBM i and non-IBM i server systems.

Specify DDM file names on the **Create C Program (CRTCPGM)** command to do the following items:

- Access remote files on an IBM i operating system or non-IBM i that contains source statements. To do this, specify a DDM file name on the SRCFILE parameter, and a member name on the SRCMBR parameter.

  Notes:

  1. For IBM i or System/38 server systems, you access a remote IBM i or System/38 source file (or member) in the same manner as a local source file and member.
  2. For non-IBM i server systems, access a remote source file by using the same file name for the SRCMBR and the SRCFILE parameters.
- Place the compiler listing in a database file on a server system. To do this, specify a DDM file name on the PRTFILE parameter of the **CRTCPGM** command.

When using ILE C programming language, consider the following items:

- If the server system is not an IBM i operating system or a System/38, you can specify a DDM file name as the include file on the #INCLUDE source directive statement, but the member name must be the same as the DDM file name.
- ILE C programming language only supports sequential I/O operations.
- Although ILE C programming language does not directly support keyed files, key exceptions might occur if you are using a keyed file.

Utility considerations for DDM

These IBM i utilities support DDM for accessing remote files.

Notes:

1. The following utilities do *not* support DDM: IBM i Query, source entry utility (SEU), screen design aid (SDA), and advanced printer function utility.
2. Except when the System/38-compatible database tools or DFU/400 is being used, DDM does not support displaying lists of members in remote files. However, if the server system is an IBM i operating system or a System/38, display station pass-through can be used to perform this function.
3. The SQL/400 licensed program and query management, part of the IBM i licensed program, do not support DDM. However, both support the Distributed Relational Database Architecture (DRDA) in a distributed network.

System/38-compatible database tools

The topic collection describes the System/38-compatible data file utility (DFU/38) and the System/38-compatible query utility (Query/38).

System/38-compatible data file utility (DFU/38)

DFU/38 data entry applications can be created and used with DDM to work with remote files in the same manner as with local files.

If a remote file is on an IBM i operating system or a System/38 product, most DFU/38 functions are performed with the remote file as though it is a local file. When creating or changing a DFU/38 application and the remote file is a logical file, the following consideration applies: either DDM files referring to each remote based-on file must exist on the client system, and the DDM file and library names must match those of the remote based-on files; or, alternatively, physical files with the same file and library names and the same record formats as the remote based-on files must exist on the client system. Because only the record formats are needed from the physical files, they need not contain data. Using this alternative, if the record formats of the remote based-on files are changed, the record formats on the client system must also be changed so that the record formats match.

However, DFU/38 does *not* support non-IBM i or non-System/38 server systems. If you attempt to use DFU/38 with non-IBM i or non-System/38 remote files, you might experience processing problems when trying to change or delete records in such a file. Although the IBM i operating system does not prevent any user from creating and using such an application, the default field descriptions created on the source IBM i for the non-IBM i or non-System/38 remote file would probably be too general to be useful. (These files appear to be physical files with one member, whose member name is the same as the file name. The file has one record format and within that format: one field for the entire record, if it is a nonkeyed file; two fields for keyed files, one for the key and one for the remainder of the record; or more than two fields for keyed files with separate key fields.)

All the DFU/38 commands can be used in applications that access local files or DDM files. And, wherever a local database file name can be specified on any of the DFU command parameters, a DDM file can also be specified, as long as any other limitations are met.

A DDM file name can be specified in the SRCFILE parameter of the **Create DFU Application (CRTDFUAPP)** or **Retrieve DFU Source (RTVDFUSRC)** command, but only if the server system is an IBM i operating system or a System/38 and if the target file is a source physical file.

System/38-compatible query utility (Query/38)

The System/38-compatible query utility (Query/38) can be used with DDM to create and use interactive or batch query applications.

If the server system is an IBM i or a System/38, most of these functions can be performed as though the remote file is a local file. When creating or changing a Query/38 application and the remote file is a logical file, the following consideration applies: either DDM files referring to each remote based-on file must exist on the client system, and the DDM file and library names must match those of the remote based-on files; or, alternatively, physical files with the same file and library names and the same record formats as the remote based-on files must exist on the client system. Because only the record formats are needed from the physical files, they need not contain data. Using this alternative, if the record formats of the remote based-on files are changed, the record formats on the client system must also be changed so that the record formats match.

If the server system is not an IBM i or a System/38, you should refer to a local file for the format and fields that describe the data in the remote file, and then use the **Override Database File (OVRDBF)** command to override the local file with a DDM file when the Query/38 application is run. The local file used to create (or re-create) the query must have the same record format name as the source description of the non-IBM i or non-System/38 target file. The default record format name is the name of the source DDM file.

Although Query/38 can create an application that uses a file on a non-IBM i or non-System/38 system, the default field descriptions created on the source IBM i for the non-IBM i remote file probably would be too general to be useful. (These files appear to be physical files with one member, whose member name is the same as the file name. The file has one record format and within that format: one field for the entire record, if it is a nonkeyed file; two fields for keyed files, one for the key and one for the remainder of the record; or more than two fields for keyed files with separate key fields.)

Non-IBM i or non-System/38 Query/38 example
This example shows how to create a local file and use it to define the data that is to be queried in a non-IBM i or non-System/38 remote file.

Assume that a DDM file named RMTS36FILE exists on your IBM i operating system and it refers to a remote System/36 file that you want to query. You can perform the following steps to determine the attributes of the remote System/36 file; to locally create a physical file that has the attributes of the remote file; and to define, create, and run the Query/38 against the remote file.

1. Use the **Display File Field Description (DSPFFD)** command and specify SYSTEM(*RMT) to display the attributes of the remote file associated with the RMTS36FILE DDM file.

   ```
   DSPFFD  FILE(RMTS36FILE) SYSTEM(*RMT)
   ```

   In this example, the displayed results would show that the remote file's record length is 80 characters, its record format name is RMTS36FILE, and it has two fields: K00001, with 12 characters (starting in position 1), and F00001, with 68 characters (starting in position 13). The K in field K00001 indicates it is the key field for this format.

2. Using the DDS and the preceding information before defining your Query/38 application, create a local physical file and call it LCLS36FILE. The DDS might look something like this:

   ```
         A          R RMTS36FILE
         A            CUSNO          6A
         A            BILLCODE       6A
         A            ADDR1         15A
         A            ADDR2         15A
         A            ADDR3         15A
         A            ZIP            5A
         A            AMTOWE         7S 2
         A            OUTBAL         7S 2
         A            MISC           4A
         A          K CUSNO
         A          K BILLCODE
   ```

   Three main rules must be followed when defining the local file:

   - The record format name must be the same as the record format name displayed by the **Display File Field Description (DSPFFD)** command.
   - Key integrity must be maintained. In this case, the key must be 12 characters long, and must start at the beginning of the file in position 1.
   - The total record length must be the same as the record length displayed by the **DSPFFD** command.

3. Define your Query/38 application using the local file created in step 2. Because the remote file is a non-IBM i file, OPTIMIZE(*NO) should be specified on the query command.

4. Before your Query/38 application is run, issue the following Override Database File (OVRDBF) command:

   ```
   OVRDBF  FILE(LCLS36FILE) TOFILE(RMTS36FILE)
   ```

   When the Query/38 application is run, this command overrides the local file you created with the DDM file that is associated with the desired target file.
5. Run your Query/38 application using the Query Data (QRYDTA) command. The net effect is that a query of the remote file is done using the local file description.

Query/38 output considerations for DDM
Query/38 output to an existing non-IBM i or a non-System/38 target file is possible, but only under specific circumstances.

Query/38 allows output to any local or remote file only if the file is sequential and if its field attributes match those attributes required by the Query/38 application. If both conditions are not met, Query/38 rejects the specified output file before the Query/38 application runs.

Because the client system description of a non-IBM i or a non-System/38 target file is very general, its field attributes probably do not match the attributes required by the Query/38 application. Therefore, in most cases, Query/38 rejects that file if it is specified for output. It works, however, if the Query/38 output consists of one alphanumeric field only, and if the record length of the target file is large enough to hold this field.

Query/38 command considerations for DDM
All the Query/38 commands can be used in applications that access local files or DDM files. And, wherever a local database file name can be specified on any of the Query/38 command parameters, a DDM file can also be specified, as long as any other limitations are met.

Note: If a Query/38 command uses a DDM file associated with a remote file on a non-IBM i or a non-System/38 server system, either the DDM file should specify LVLCHK(*NO) or an OVRDBF command should be used to override that parameter with *NO. This is recommended to avoid level-checking problems with the target file.

A DDM file name can be specified in the SRCFILE parameter of the **Create Query Application (CRTQRYAPP)** or **Retrieve Query Source (RTVQRYSRC)** command, but only if the server system is an IBM i or a System/38 and if the target file is a source physical file.

Query/38 optimization for DDM
Query/38 has an optimization function, but because it causes IBM i database query to be used, the feature cannot be used when the query is performed against a remote file that is not on an IBM i or a System/38.

Because IBM i database query does not exist on non-IBM is or non-System/38s, the optimization function cannot be used by the source IBM i when performing a query against a non-IBM i or a non-System/38 remote file.

Therefore, when a Query/38 application is being created or changed that accesses a remote file on a non-IBM i or a non-System/38, the OPTIMIZE parameter on the **Create Query Application (CRTQRYAPP)**, **Create Query Definition (CRTQRYDEF)**, or **Change Query Definition (CHGQRYDEF)** command must be changed to *NO. Specifying OPTIMIZE(*NO) forces Query/38 to read the file sequentially, which can be done with non-IBM i target files. If the default of *YES is used, an error occurs when the Query/38 application is run.

Similarly, if the **Design Query Application (DSNQRYAPP)** command is used to create and run queries that are to be performed on a non-IBM i target file, the *Optimize Query* prompt on the Application Creation display must be changed from Y to N.

Existing Query/38 application considerations for DDM
You should keep these considerations in mind for existing Query/38 applications.

Existing Query/38 applications, if they are to query remote files, must be re-created in all cases, even if the server system is an IBM i or a System/38. If the server system is an IBM i or a System/38, the re-created application that uses a DDM file is defined and run as if the remote file is a local file. The optimization feature can be used to get the records from the target IBM i or the target System/38.

Data file utility for IBM i
Data file utility (DFU) data entry applications can be created and started with DDM to work with remote files in the same manner as with local files. Most DFU functions are performed with the remote file as though it were a local file.

When you create or change a DFU function of Application Development Tools and the remote file is an IBM i or System/38 logical file, the following consideration applies: either DDM files referring to each remote based-on file must exist on the client system, and the DDM file and library names must match those of the remote based-on files; or, alternatively, physical files with the same file and library names and the same record formats as the remote based-on files must exist on the client system. Because only the record formats are needed from the physical files, they need not contain data. Using this alternative, if the record formats of the remote based-on files are changed, the record formats on the client system must also be changed so that the record formats match. Similar considerations apply when the remote file is a System/36 logical file.

DFU supports IBM i, System/38, and System/36 remote files. However, DFU does not prevent you from using non-IBM i, non-System/38, or non-System/36 remote files and you might experience problems when using such files.

Non-IBM i or System/36 files are program-described files. DFU allows you to use either a local or remote file containing ILE RPG file and input specifications to define these data files.

IBM i database query
The database interactive query function, provided by the IBM i licensed program, supports DDM files.

This support is used by IBM i Access Family and System/38-compatible query utility if OPTIMIZE(*YES) is specified. You can query remote files using the Open Query File (OPNQRYF) command, but only if the remote files are on a target IBM i or a target System/38.

The query utility on the System/38 can be used to query remote files that are not from the IBM i operating system.

Database query allows accessing of either multiple local files or multiple remote files (by using DDM files) at the same time, but not both.

If all the files are remote, they must all reside on the same server system. Also, the DDM files that refer to the remote files must all specify the same remote location information. If this restriction is not met, an error message is displayed to the user of the IBM i Access Family licensed program or to the user of the Open Query File (OPNQRYF) command who requested the query.

Sort utility
The sort utility supports remote file processing with DDM anywhere that it supports local file processing for both IBM i and non-IBM i server systems.

Generally, on the **`Format Data (FMTDTA)`** command, DDM file names can be specified anywhere a database file name can be specified.

- A DDM file name can be specified on the SRCFILE parameter, and a member name can be specified on the SRCMBR parameter, for IBM i or System/38 server systems. If the remote file referred to by the DDM file is not on an IBM i or a System/38, a member name cannot be specified.
- DDM file names can also be specified on the INFILE parameter (to access a remote file as the input file for conversion) or on the OUTFILE parameter (to access a remote file as the output file of the conversion). Both parameters cannot specify DDM file names at the same time.

IBM i Access Family considerations for DDM

The IBM i Access Family supports DDM for accessing remote files, with limitations.

Note: IBM Business Graphics Utility for IBM i does not support DDM.

The transfer function in IBM i Access Family can be used with DDM to transfer data between a personal computer attached to a local IBM i product and another remote system.

When the transfer function is used, the remote system must be running IBM i or System/38. The IBM i Access Family copy commands, **`Copy to PC Document (CPYTOPCD)`** and **`Copy from PC Document (CPYFRMPCD)`**, can be used to copy data on a host server or between host servers.

The figure here shows a personal computer attached to the local IBM i platform. The IBM i Access Family user can access data on remote systems through a DDM file defined on the local system. The IBM i platform with the personal computer attached can only be the client system.

- The IBM i Access Family transfer function can be used by a personal computer user to transfer data from a remote file to the personal computer, or to transfer data from the personal computer to a remote file. Only a personal computer user can start the requests, not an IBM i user.

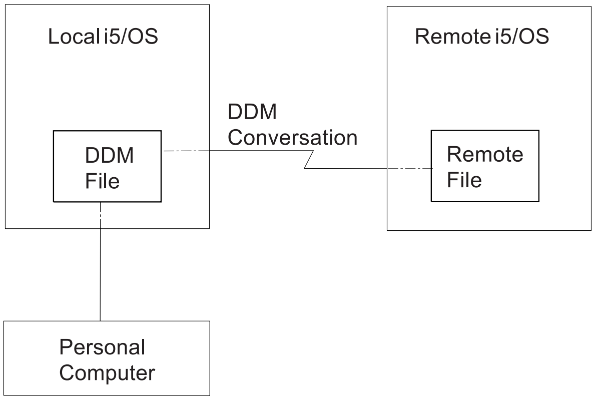


Figure 26. Using DDM with IBM i Access Family

- The IBM i Access Family copy commands can be used with DDM to copy data from a personal computer document located on the local IBM i platform to a database file on the remote IBM i platform, or to copy data to a personal computer document on the local IBM i platform from a database file on the remote IBM i platform.

Note: For IBM i Access Family, database query allows accessing of multiple remote files (by using DDM files) at the same time.

IBM i Access Family transfer function considerations

A personal computer user can use the transfer function in IBM i Access Family and the DDM support on the local IBM i platform to which the personal computer is attached either to transfer data from the personal computer to a remote file, or to transfer data from a remote file to the personal computer.

The remote file must be on a IBM i or a System/38 platform.

When DDM is used to transfer files or data from a remote system to an attached personal computer, the DDM files (that refer to remote files) on the local IBM i platform cannot be joined with local files to transfer data to the personal computer. (That is, data from files on both the remote and local systems cannot be joined.) However, a DDM file can specify a remote file that is a logical join file built over multiple physical files. DDM files that refer to the same server system and use the same remote location information can be joined.

A transfer request that requires group processing does not work if the local system is a System/38 product and the remote system is a IBM i product, or if the local system is a IBM i product and the remote system is a System/38 product.

When DDM is used to transfer a file or data from an attached personal computer to a remote system, a remote file cannot be created on the server system. The remote file must already exist before the data from the personal computer can be transferred. However, because the target must be a IBM i or a

System/38 product, a new member can be added in the remote file before personal computer data is transferred to that file member.

IBM i Access Family copy command considerations

The IBM i CL command **Copy from Personal Computer Document (CPYFRMPCD)** used in IBM i Access Family can be used to copy data *from* a document located either on an IBM i *to* a database file member located on the same IBM i or on a remote IBM i using DDM.

The CL command **Copy to Personal Computer Document (CPYTOPCD)** can be used to copy data *from* a database file member on a local IBM i or a remote IBM i (using DDM) *to* a document on the local IBM i. The remote file can be on a target IBM i or a non-IBM i. To use these commands, specify the name of a DDM file on the:

- TOFILE parameter in the **Copy from PC Document (CPYFRMPCD)** command, to copy a personal computer document to an IBM i physical file.
- FROMFILE parameter in the **Copy to PC Document (CPYTOPCD)** command, to copy a member from an IBM i database file to a personal computer document in a folder.

The following restrictions apply to the CL copy commands for IBM i Access Family:

- For the **CPYFRMPCD** command, a remote file cannot be created on the server system (whether it is an IBM i or a non-IBM i). The remote file must already exist before the personal computer document data can be copied to it. However, if the target is an IBM i or a System/38, a new member can be created for the remote file before the personal computer document data is copied to that file member.
- The **CPYFRMPCD** and **CPYTOPCD** commands are IBM i CL commands and cannot be entered at the DOS prompt from the personal computer.

For more information on the **CPYTOPCD** and the **CPYFRMPCD** commands, see the online help information.

Hierarchical file system API support for DDM

The hierarchical file system (HFS) APIs and the functions that they support are part of the IBM i operating system.

The APIs provide applications with a single, consistent interface to all the hierarchical file systems available on your IBM i. They automatically support the document library services (DLS) file system and can support user-written file systems also.

DDM can be registered under HFS as one of the user-written file systems. DDM, however, only supports the copy stream file (QHFCPYSF) HFS API. To register DDM under HFS, you must run the following command on your IBM i client system, `CALL QTSREGFS`. If no errors occur, DDM is successfully registered with HFS.

Calling DDM using the HFS QHFCPYSF API causes one of two DDM-architected commands to be generated, the **LODSTRF (load stream file)** or **ULDSTRF (unload stream file)** command. Both of these DDM commands are part of the stream file DDM model (STRFIL). If the DDM server system you are working with does not support the STRFIL DDM model, then errors will occur when trying to use this support. DDM uses documents and folders (DLS) on the server to copy stream file data either to (ULDSTRF case) or from (LODSTRF case).

To use the DDM HFS copy stream file support, note the following items:

- Both the source and destination file path names must begin with the string '/QDDM/' to indicate to HFS that DDM is the file system that will handle the copy stream file function.
- The copy information HFS parameter is ignored by DDM, but you still must pass a valid HFS value.
- Either the source or destination file path name parameter must be the name of a DDM file, but not both. The DDM file used must point to a target server that supports the STRFIL DDM file model and the remote file name value must end with the string ' FMS' if the DDM file points to another IBM i.
- The other source or destination file path name parameter that is not a DDM file, must be the name of an existing DLS object (document in a folder) and the name must be followed by the string ' FMS'.
- The maximum source or target path name length supported by DDM is 63 characters. The 63 characters do not include the '/QDDM/' or the ' FMS' possible appendages.

- In the LODSTRF case (source file path name is a local DLS object and target file path name is a DDM file), the local DLS document is read starting at offset zero through the end of the file. Whether the destination file (pointed to by the DDM file) exists or not is dependent on the server system's stream file support.
- In the ULDSTRF case (source file path name is a DDM file and destination file path name is a local DLS object), the local or target DLS document must exist on the IBM i and will have its contents cleared and then written to starting at offset zero.

Here is a copy stream file example that will generate a **LODSTRF** DDM command to a remote server:

```
CRTDDMF FILE(DDMLIB/DDMFILE) +
RMTFILE(*NONSTD 'TARGET/SYSTEM/
SYNTAX/PATHNAME FMS') RMTLOCNAME(RMTSYSNM)
```

In this example, the local DLS object is 'PATH1/PATH2/FOLDER1/DOC1'.

You would call QHFCPYSF with the following parameter list:

```
1   Source file path name = '/QDDM/PATH1/PATH2/FOLDER1/DOC1 FMS'
2   Source file path name length = 34
3   Copy information = valid HFS value that is ignored by DDM
4   Target file path name = '/QDDM/DDMLIB/DDMFILE'
5   Target file path name length = 20
```

Just reverse the source and destination file path names and lengths to generate an **ULDSTRF** DDM command.

The example program in the following example calls DDM HFS API:

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 362.

```
/********************************************************************/
/********************************************************************/
/*  FUNCTION: This program copies a stream file using the QHFCPYSF  */
/*            HFS API.                                              */
/*                                                                  */
/*  LANGUAGE: PL/I                                                  */
/*                                                                  */
/*  APIs USED: QHFCPYSF                                             */
/*                                                                  */
/********************************************************************/
/********************************************************************/
TRANSFER: PROCEDURE(SRCFIL,TRGFIL) OPTIONS(MAIN);

/* parameter declarations                                           */
DCL SRCFIL CHARACTER (73);
DCL TRGFIL CHARACTER (73);

/* API entry declarations                                           */
/*                                                                  */
/* The last parameter, the error code, is declared as FIXED BIN(31) */
/* for the API. This always has a value of zero, specifying that    */
/* exceptions should be returned.                                   */
DCL QHFCPYSF ENTRY(CHAR(73),FIXED BIN(31),CHAR(6),CHAR(73),
                   FIXED BIN(31),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

/********************************************************************/
/*  Parameters for QHFCPYSF                                         */
/********************************************************************/
DCL srclen FIXED BIN(31);
DCL trglen FIXED BIN(31);
DCL cpyinfo CHAR(6);
DCL error_code FIXED BIN(31);

/********************************************************************/
/*  Mainline routine                                                */
/********************************************************************/

srclen = INDEX(SRCFIL,'  ') - 1;
trglen = INDEX(TRGFIL,'  ') - 1;
cpyinfo = '1     ';
error_code = 0;
/* Copy the stream file                                             */
Call QHFCPYSF(SRCFIL,srclen,cpyinfo,TRGFIL,trglen,
              error_code);

END TRANSFER;
```

Figure 27. Program example

Sample command source that can be used with the preceding program:

```
CMD
     PARM    KWD(SRCFIL) TYPE(*CHAR) LEN(73) +
               PROMPT('SOURCE FILE NAME')
     PARM    KWD(TRGFIL) TYPE(*CHAR) LEN(73) +
               PROMPT('TARGET FILE NAME')
```

Using CL and DDS with DDM

This topic contains DDM-related information about specific control language (CL) commands, data description specifications (DDS) considerations, DDS keywords, and DDM user profile authority.

Related concepts

Planning and design for DDM

There are several requirements that must be met for distributed data management (DDM) to be used properly.

DDM-specific CL commands

This topic covers DDM-specific CL commands.

Change DDM File (CHGDDMF) *command*

The **Change DDM File (CHGDDMF)** command changes one or more of the attributes of a DDM file on the local (client) system.

The DDM file is used as a reference file by programs on the IBM i client system to access files located on any server system in the IBM i DDM network.

To use this command, you can enter the command as shown in the following example or select option 2 (Change DDM File) from the Work with DDM Files display.

Example: CHGDDMF command

This command changes the communications mode for the DDM file named SALES stored in the SOURCE library on the client system; the mode is changed to MODEX.

```
CHGDDMF FILE(SOURCE/SALES) MODE(MODEX)
```

Create DDM File (CRTDDMF) *command*

The **Create DDM File (CRTDDMF)** command creates a DDM file on the local (client) system.

The DDM file is used as a reference file by programs on an IBM i to access files located on any remote (server) system in the IBM i DDM network. Programs on the local IBM i know a remote file only by the DDM file's name, not the remote file's actual name. (The DDM file name, however, can be the same as the remote file name.)

The DDM file is also used when a CL command is submitted to the remote server. (The **Submit Remote Command (SBMRMTCMD)** command is used to submit the CL command, and the remote server must be an IBM i or a System/38.) When the **SBMRMTCMD** command is being used, the remote file normally associated with the DDM file is ignored.

The DDM file contains the name of the remote file being accessed and the remote location information that identifies a remote (server) system where the remote file is located. It can also specify other attributes that are used to access records in the remote file.

To use this command, you can enter the command as shown in the following examples or select F6 (Create DDM file) from the Work with DDM Files display.

Example: Create a DDM file to access a file on a System/38

```
CRTDDMF  FILE(SOURCE/SALES)  RMTFILE(*NONSTD 'SALES.REMOTE')
   RMTLOCNAME(NEWYORK)
```

This command creates a DDM file named SALES and stores it in the SOURCE library on the client system. This DDM file uses the remote location NEWYORK to access a remote file named SALES stored in the REMOTE library on a System/38 in New York.

Example: Create a DDM file to access a file member on an IBM i

```
CRTDDMF  FILE(SOURCE/SALES)  RMTLOCNAME(NEWYORK)
   RMTFILE(*NONSTD 'REMOTE/SALES(APRIL)')
```

This command creates a DDM file similar to the one in the previous example, except that now it accesses the member named APRIL in the remote SALES file stored in the REMOTE library on an IBM i.

Example: Create a DDM file to access a file on a System/36

```
CRTDDMF  FILE(OTHER/SALES)  RMTFILE(*NONSTD 'PAYROLL')
   RMTLOCNAME(DENVER)  LVLCHK(*NO)
```

This command creates a DDM file named SALES, and stores it in the library OTHER on the client system. The remote location DENVER is used by the DDM file to access a remote file named PAYROLL on a System/36 in Denver. No level checking is performed between the PAYROLL file and the application programs that access it. Because the ACCMTH parameter was not specified, the access method for the server system is selected by the source IBM i when the DDM file is opened to access the remote file.

Display DDM Files (DSPDDMF) *command*

The Display DDM Files (DSPDDMF) command displays the details of a DDM file. To use this command, you can type the command or select option 5 (Display details) from the Work with DDM Files display.

Reclaim DDM Conversations (RCLDDMCNV) *command*

The **Reclaim DDM Conversations (RCLDDMCNV)** command is used to reclaim all DDM client system conversations that are not currently being used by a source job.

The conversations are reclaimed even if the value of the job's DDMCNV attribute is *KEEP, or if the command is entered within an activation group. The command allows the user to reclaim unused DDM conversations without closing all open files or doing any of the other functions performed by the **Reclaim Resources (RCLRSC)** command.

The **RCLDDMCNV** command applies only to the DDM conversations for the job on the *source* server in which the command is entered. For each DDM conversation used by the source job, there is an associated job on the server system; the target job ends automatically when the associated DDM conversation ends.

Although this command applies to all DDM conversations used by a job, using it does not mean that all of them will be reclaimed. A conversation is reclaimed *only* if it is not being actively used.

Submit Remote Command (SBMRMTCMD) *command*

The **Submit Remote Command (SBMRMTCMD)** command submits a command using DDM to run on the server system.

The remote location information in the DDM file is used to determine the communications line to be used, and thus, indirectly identifies the target server that is to receive the submitted command.

You can use the **SBMRMTCMD** command to send commands to any of the following server systems:

- IBM i
- System/38
- Any server that supports the **Submit System Command (SBMSYSCMD)** DDM command

The **SBMRMTCMD** command can be used to send CL commands (and only CL) to an IBM i or a System/38. It can also be used to send commands to server systems other than IBM i or System/38 servers if the server system supports the DDM architecture Submit System command. The command must be in the syntax of the server system. The **SBMRMTCMD** command cannot be used to send operation control language (OCL) commands to a System/36 target because the System/36 server does not support the function.

The primary purpose of this command is to allow a user or program using the client system to perform file management operations and file authorization activities on files located on a server system. The user must have the proper authority for the server system objects that the command is to operate on. The following actions are examples of what can be performed on remote files using the **SBMRMTCMD** command:

- Create or delete device files
- Grant or revoke object authority to remote files
- Verify files or other objects
- Save or restore files or other objects

Although the command can be used to do many things with files or objects, some are not as useful as others. For example, you can use this command to display the file descriptions or field attributes of remote files, or to dump files or other objects, but the output remains at the server system. Another way to display remote file descriptions and field attributes at the client system is to use the **Display File Description (DSPFD)** and **Display File Field Description (DSPFFD)** commands. Specify the SYSTEM(*RMT) parameter and the names of the DDM files associated with the remote files. This returns the information you want directly to the local server.

A secondary purpose of this command is to allow a user to perform nonfile operations (such as creating a message queue) or to submit user-written commands to run on the server system. The CMD parameter allows you to specify a character string of up to 2000 characters that represents a command to be run on the server system.

Related concepts

Parts of DDM: DDM file
A *DDM file* is a file on the client system that contains the information needed to access a data file on a server system.

IBM i and System/38 server systems on the ***SBMRMTCMD*** *command*
The **SBMRMTCMD** command can submit any CL command that can run in both the batch environment and using the QCAEXEC server program.

That is, a command can be submitted using the **SBMRMTCMD** command if it has both of the following values for the ALLOW attribute:

***BPGM**
The command can be processed in a compiled CL program that is called from batch entry.

***EXEC**
The command can be used as a parameter on the CALL command and get passed as a character string to the server program for processing.

You can look for these possible values using the **Display Command (DSPCMD)** command. (The **SBMRMTCMD** command uses the QCAEXEC or QCMDEXEC system program to run the submitted commands on the server system.) However, because some of these allowable commands require intervention on the server system and might not produce the results expected, you should consider the items listed in the topic Restrictions for the SBMRMTCMD first.

The user must have the proper authority for both the CL command being submitted and for the server system objects that the command is to operate on.

Restrictions for the ***SBMRMTCMD*** *command*
This topic describes restrictions for the **SBMRMTCMD** command.

- Although remote file processing is synchronous within the user's job, which includes two separate jobs (one running on each server), file processing on the server system operates independently of the client system. Commands such as **Override with Database File (OVRDBF)**, **Override with Message File (OVRMSGF)**, and **Delete Override (DLTOVR)** that are dependent on the specific position of a program in a program stack (*recursion level*) or request level might *not* function as expected.

  For example, when multiple recursion levels that involve overrides at each level occur on the client system, and one or more overrides at a given level are submitted to the server system on the **SBMRMTCMD** command, the server system job has no way of knowing the level of the client system job. That is, a server system override can still be in effect after the source server override for a particular recursion level has ended.
- Output (such as spooled files) created by a submitted command exists only on the server system. The output is *not* sent back to the client system.
- Some types of CL commands should *not* be submitted to a target IBM i. The following items are examples of types that are *not* the intended purpose of the **SBMRMTCMD** command and that might produce undesirable results:

- All of the OVR*xxx*F commands that refer to database files, message files, and device files (including communications and save files).
- All of the DSP*xxxx* commands, because the output results remain at the server system.
- Job-related commands like **Reroute Job (RRTJOB)** that are used to control a server system's job. The **Change Job (CHGJOB)** command, however, *can* be used.
- Commands that are used to service programs, like **Service Job (SRVJOB)**, **Trace Job (TRCJOB)**, **Trace Internal (TRCINT)**, or **Dump Job (DMPJOB)**.
- Commands that might cause inquiry messages to be sent to the system operator, like **Start Printer Writer (STRPRTWTR)**. (Pass-through can be used instead.)
- Commands that attempt to change the Independent Auxiliary Storage Pool (ASP) of the target job (for example, SETASPGRP) should not be issued using **Submit Remote Command**.

- Translation is not performed for any *immediate* messages created by the server system, because they are not stored on the server; the text for an immediate message is sent directly to the client system to be displayed. (For all other message types, the server system sends back a message identifier; the message text that exists on the client system for that message identifier is the text that is displayed. This message text is whatever the client system text has been translated to.)
- A maximum of 10 messages, created during the running of a submitted command, can be sent by the server system to the client system. If more than 10 messages are created, an additional *informational* message is sent that indicates where the messages exist (such as in a job log) on the server system. If one of those messages is an *escape* message, the first nine messages of other types are sent, followed by the informational message and the escape message.
- The only types of messages that are sent by the server system are completion, informational, diagnostic, and escape messages.

Examples: SBMRMTCMD command
The examples display different uses for the SBMRMTCMD command.

Example: Submit a command to create another DDM file on the remote server

```
SBMRMTCMD  CMD('CRTDDMF  FILE(SALES/MONTHLY)
   RMTFILE(*NONSTD ''SALES/CAR(JULY)'')
   RMTLOCNAME(DALLAS)')  DDMFILE(CHICAGO)
```

This submitted command creates, on the server system identified by the information in the DDM file named CHICAGO, another DDM file named MONTHLY; the new DDM file is stored in a library named SALES on the server defined by DDMFILE CHICAGO. The new DDM file on the CHICAGO server is used to access a file and *member* on a different server named DALLAS. The accessed file is named CAR in the library SALES and the member name in the file is JULY.

Notice that this CRTDDMF command string contains *three* sets of single quotation marks: one set to enclose the entire command being submitted (required by the CMD parameter on the SBMRMTCMD command), and a double set to enclose the file and member named in the RMTFILE parameter. Because the use of *NONSTD requires that nonstandard file names be enclosed in a set of single quotation marks, this second set of single quotation marks must be doubled because it is within the first set of single quotation marks.

Example: Submit a command to change text in a display file

```
SBMRMTCMD  CMD('CHGDSPF  FILE(LIBX/STANLEY)
   TEXT(''Don''''t forget to pair apostrophes.'')')
   DDMFILE(SMITH)
```

This command changes the text in the description of the display device file named STANLEY stored in library LIBX. Because the submitted command requires an outside set of single quotation marks (for the CMD parameter), each single quotation mark (') or quotation marks ('') normally required in the TEXT parameter for *local* server processing must be doubled again for *remote* server processing. The preceding coding produces a single quotation mark in the text when it is displayed or printed on the remote server.

Example: Submit a command to replace a library list on the remote server

```
SBMRMTCMD  CMD('CHGLIBL  LIBL(QGPL QTEMP SALES EVANS)')
   DDMFILE(EVANS)
```

This command changes the user's portion of the library list being used by the target job associated with the DDM file named EVANS, which is being used by the source job in which this SBMRMTCMD command is being submitted. In that source job, if there are other open DDM files that specify the remote location information, this library list is used for them also.

Example: Override use

The DDMFILE parameter on the SBMRMTCMD command is used to determine which server system the command (CMD parameter) should be sent to. Overrides that apply to the DDM file (not the remote file) are taken into account for this function. For example, if a file override was in effect for a DDM file because of the following commands, which override FILEA with FILEX, then the server system that the Delete File (DLTF) command is sent to is the one associated with the remote location information specified in DDM FILEX (the values point to the DENVER system, in this case).

```
CRTDDMF  FILE(SRCLIB/FILEA)  RMTFILE(SALES/CAR)
   RMTLOCNAME(CHICAGO)
CRTDDMF  FILE(SRCLIB/FILEX)  RMTFILE(SALES/CAR)
   RMTLOCNAME(DENVER)
OVRDBF   FILE(FILEA)  TOFILE(SRCLIB/FILEX)
SBMRMTCMD  CMD('DLTF RMTLIB/FRED')  DDMFILE(SRCLIB/FILEA)
```

This **SBMRMTCMD** command deletes the file named FRED from the DENVER server.

Additional considerations: SBMRMTCMD command
This topic describes additional considerations for the SBMRMTCMD command.

DDM conversations

When a SBMRMTCMD command is run on the server system, it has a server system job associated with it. Successive SBMRMTCMD commands submitted using the same DDM file and DDM conversation might run in the same or different server system jobs, depending on the value of the DDMCNV job attribute. The value of the DDMCNV job attribute determines whether the DDM conversation is dropped or remains active when the submitted function has completed. If the conversation is dropped, the next SBMRMTCMD command runs using a different target job. If several commands are submitted, either DDMCNV(*KEEP) should be in effect, or display station pass-through should be used instead of DDM.

Command syntax verifying

The syntax of the command character string being submitted by the CMD parameter is not verified by the client system. In the case of a user-defined command, for example, the command definition object might or might not exist on the client system.

Command running results

Because the submitted command runs as part of the server system's job, the attributes of that job (such as the library search list, user profile, wait times, and running priority) might cause a different result than if the command were run locally. If you find that you are having difficulty submitting a command and, for example, the reason is that the server system uses a different library list, you can use the SBMRMTCMD command to edit the library list.

Error message handling

For errors detected by the server system when processing the submitted command, the client system attempts to send the same error information that was created on the server system to the user. However, if the client system does not have an equivalent message for the one created on the server system, the message sent to the client system user has the message identifier and is of the message type and severity that was created on the server system. The message text sent for the error is default message text.

If the server system is a system other than an IBM i or System/36, messages sent to the client system have no message identifiers or message types. The only information received from such a server system is the message text and a severity code. When a high severity code is returned from the target server, the client system user receives a message that the SBMRMTCMD command ended abnormally. Other messages sent by the server system are received as informational with no message identifiers.

For example, you might see the following statements in your job log when both the source and target are IBM is:

```
INFO CPI9155 'Following messages created on target system.'
DIAG CPD0028 'Library ZZZZ not found.'
ESCP CPF0006 'Errors occurred in command.'
```

When a server system other than an IBM i returns the same message to an IBM i client system, the job log looks like this:

```
INFO CPI9155 'Following messages created on target system.'
INFO nomsgid 'Library ZZZZ not found.'
INFO nomsgid 'Errors occurred in command.'
ESCP CPF9172 'SBMRMTCMD command ended abnormally.'
```

The server system messages can be viewed on the client system by using pass-through and either the Work with Job (WRKJOB) or Work with Job Log (WRKJOBLOG) command. If the target job ends, the messages are in the server system's output queue, where they can be displayed by the Work with Output Queue (WRKOUTQ) command.

If the SBMRMTCMD command is used to call a CL program on the server system, any escape message that is not monitored and is created by the program is changed into an inquiry message and is sent to the system operator. If you do not want the server system operator to have to respond to this inquiry message before the job can continue, you can do either of the following items on the server system:

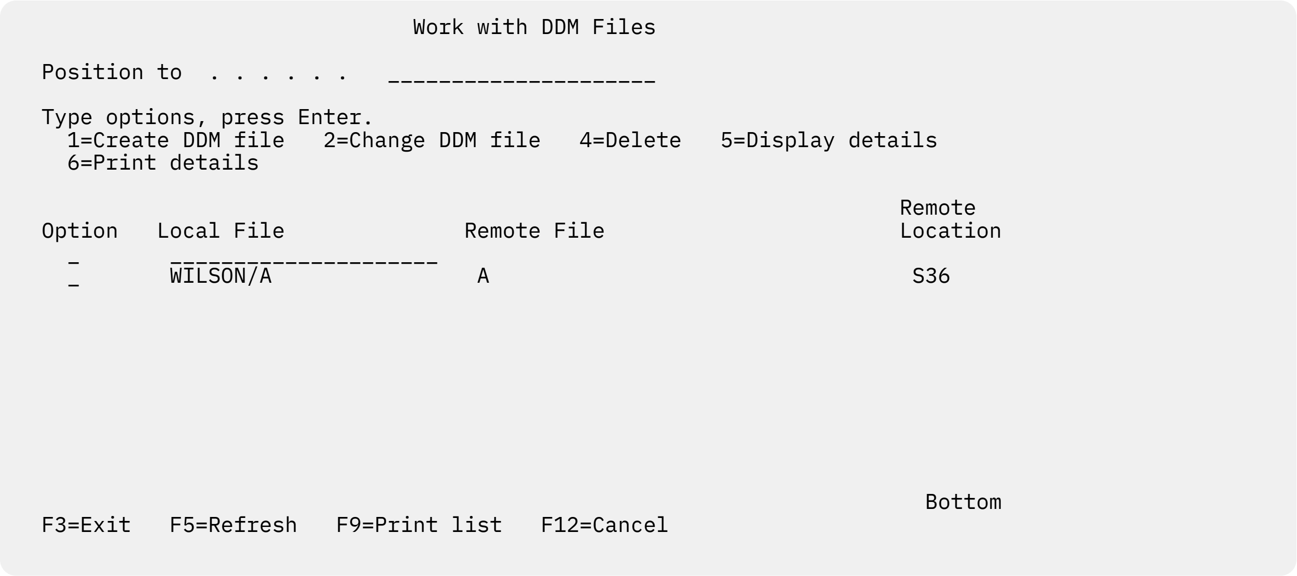
- If you want to specify a default reply for a specific *job*, you can use the INQMSGRPY parameter on either the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command to specify either *DFT or *SYSRPYL in the job description for the target job. You can also do the same thing if you use the SBMRMTCMD command to submit the Change Job (CHGJOB) command to the target server.
- If you want to specify a default reply message for a specific *inquiry message* in the job, you can use the Add Reply List Entry (ADDRPYLE) command (on the server system) to add an entry for that message to the system-wide automatic message reply list (SYSRPYL). Then, if INQMSGRPY(*SYSRPYL) is specified in the job description, this default reply can be sent whenever that inquiry message occurs in the job.

Independent auxiliary storage pools (ASPs)

If the server system has online independent ASPs, the independent ASP group of the target job is established when the conversation is started and might not be changed. User-defined or CL commands that attempt to change the independent ASP group of the target job (for example, SETASPGRP or DLTUSRPRF) might fail if submitted to a server system that has online independent ASPs.

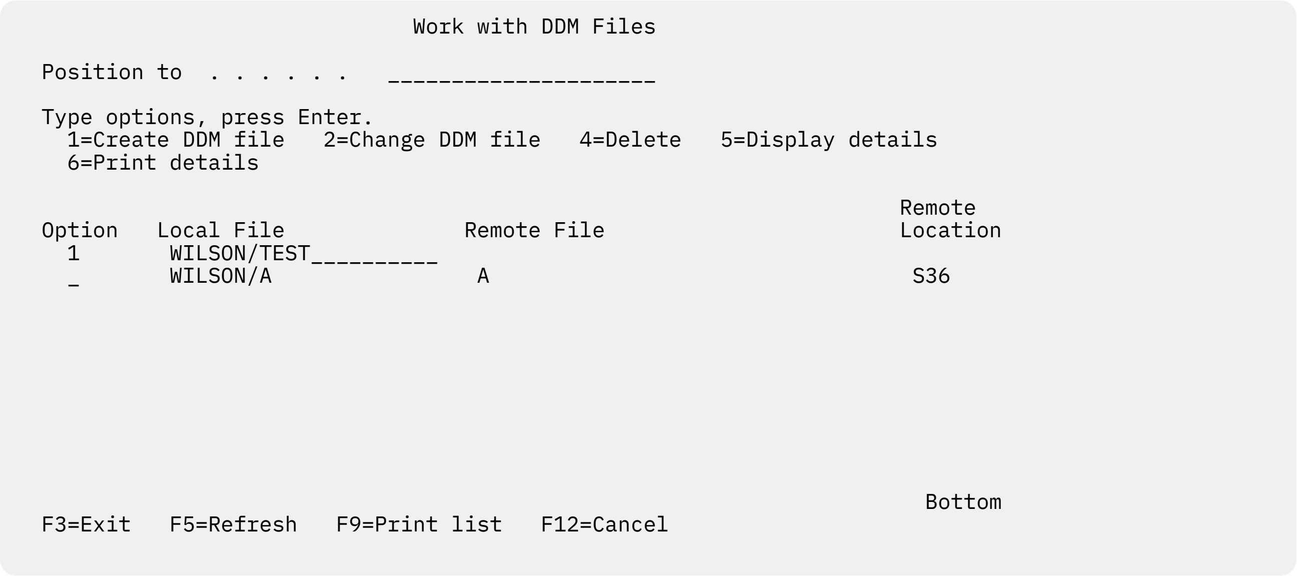
Work with DDM Files (WRKDDMF) command
The Work with DDM Files (WRKDDMF) command allows you to work with existing DDM files from a list display. From the list display, you can change, delete, display, or create DDM files.

For the following displays, it is assumed that you have created DDM files using the Create DDM File (CRTDDMF) command. If you enter the WRKDDMF command and specify library `WILSON` and file A, the following display is shown:

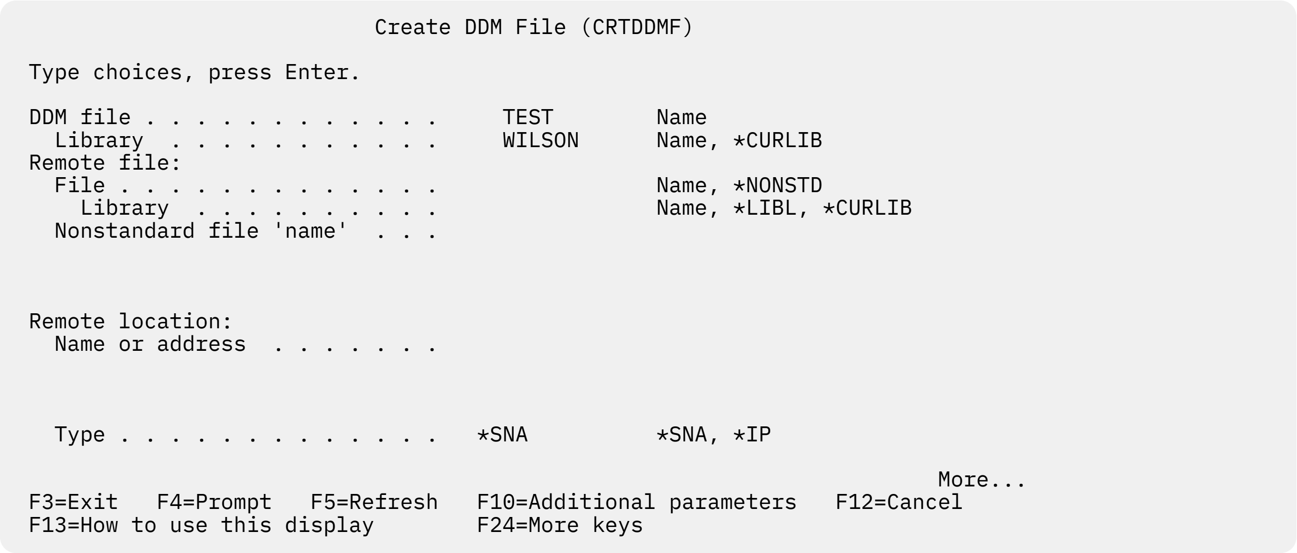


```
                             Work with DDM Files

Position to  . . . . . .   _____________________

Type options, press Enter.
  1=Create DDM file   2=Change DDM file   4=Delete   5=Display details
  6=Print details

                                                                   Remote
Option   Local File              Remote File                       Location
  _       _____________________
  _       WILSON/A                A                                 S36









                                                                     Bottom
F3=Exit   F5=Refresh   F9=Print list   F12=Cancel
```

To create a DDM file using this display, type a 1 in the option column and type the names of the library and file you want to create, then press the Enter key. For example, type a 1 (Create DDM file) in the option field and WILSON/TEST in the local file column of the top list entry (as shown in the following display), and then press the Enter key. The Create DDM File display is shown.

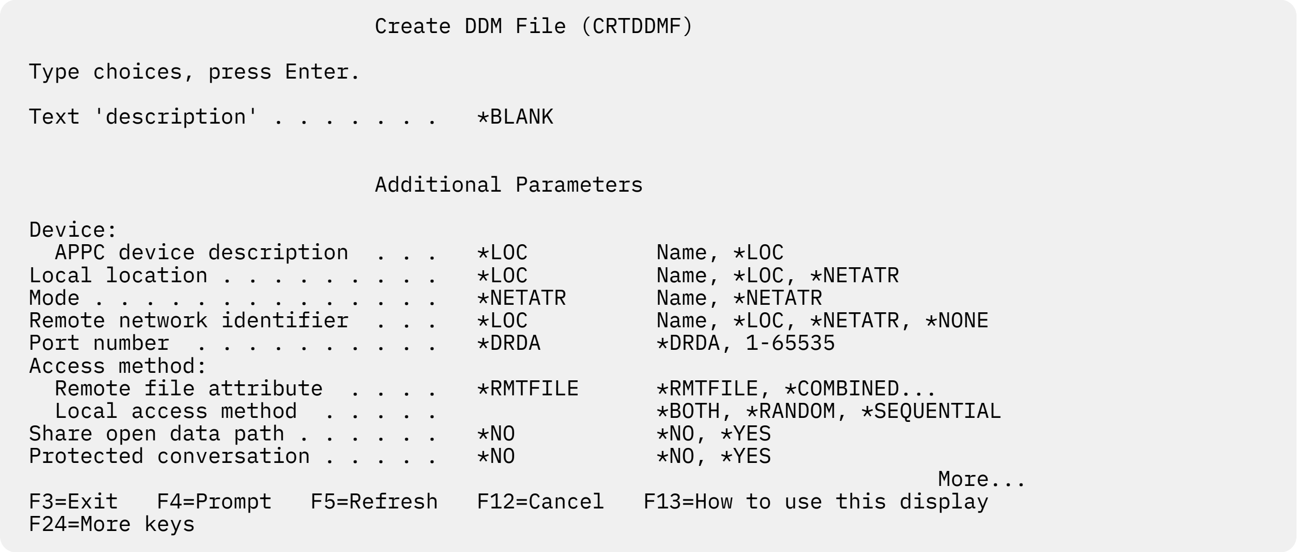


```
                             Work with DDM Files

Position to  . . . . . .   _____________________

Type options, press Enter.
  1=Create DDM file   2=Change DDM file   4=Delete   5=Display details
  6=Print details

                                                                   Remote
Option   Local File              Remote File                       Location
  1       WILSON/TEST__________
  _       WILSON/A                A                                 S36









                                                                     Bottom
F3=Exit   F5=Refresh   F9=Print list   F12=Cancel
```

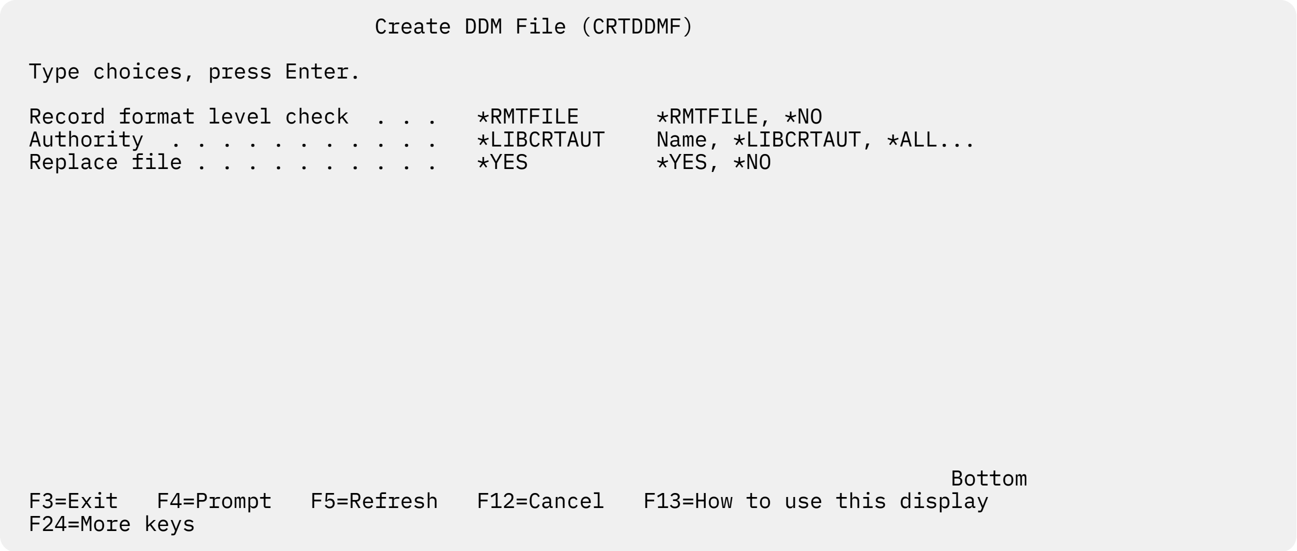


```
                           Create DDM File (CRTDDMF)

Type choices, press Enter.

DDM file . . . . . . . . . . . .     TEST        Name
  Library  . . . . . . . . . . .     WILSON      Name, *CURLIB
Remote file:
  File . . . . . . . . . . . . .                 Name, *NONSTD
    Library  . . . . . . . . . .                 Name, *LIBL, *CURLIB
  Nonstandard file 'name'  . . .



Remote location:
  Name or address  . . . . . . .



  Type . . . . . . . . . . . . .   *SNA          *SNA, *IP

                                                                       More...
F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display        F24=More keys
```

On the Create DDM File display, type the required values, and change or use the default values given. By pressing F10 (Additional parameters), you can page through the command parameters as they are shown on two displays. By pressing the Page Down key, you are shown these additional parameters:



```
                           Create DDM File (CRTDDMF)

Type choices, press Enter.

Text 'description' . . . . . . .   *BLANK


                           Additional Parameters

Device:
  APPC device description  . . .   *LOC          Name, *LOC
Local location . . . . . . . . .   *LOC          Name, *LOC, *NETATR
Mode . . . . . . . . . . . . . .   *NETATR       Name, *NETATR
Remote network identifier  . . .   *LOC          Name, *LOC, *NETATR, *NONE
Port number  . . . . . . . . . .   *DRDA         *DRDA, 1-65535
Access method:
  Remote file attribute  . . . .   *RMTFILE      *RMTFILE, *COMBINED...
  Local access method  . . . . .                 *BOTH, *RANDOM, *SEQUENTIAL
Share open data path . . . . . .   *NO           *NO, *YES
Protected conversation . . . . .   *NO           *NO, *YES
                                                                       More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```



```
                           Create DDM File (CRTDDMF)

Type choices, press Enter.

Record format level check  . . .   *RMTFILE      *RMTFILE, *NO
Authority  . . . . . . . . . . .   *LIBCRTAUT    Name, *LIBCRTAUT, *ALL...
Replace file . . . . . . . . . .   *YES          *YES, *NO
















                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

After you have typed in the values, press the Enter key to process the command and return to the Work with DDM Files display.

If you want to change a DDM file, type a 2 (Change DDM file) on the Work with DDM Files display next to the file that you want to change, or type the option number in the top list entry of the Options column and specify the local file that you want changed. For example, type a 2 (Change DDM file) in the *Option* column of the local file named WILSON/TEST.

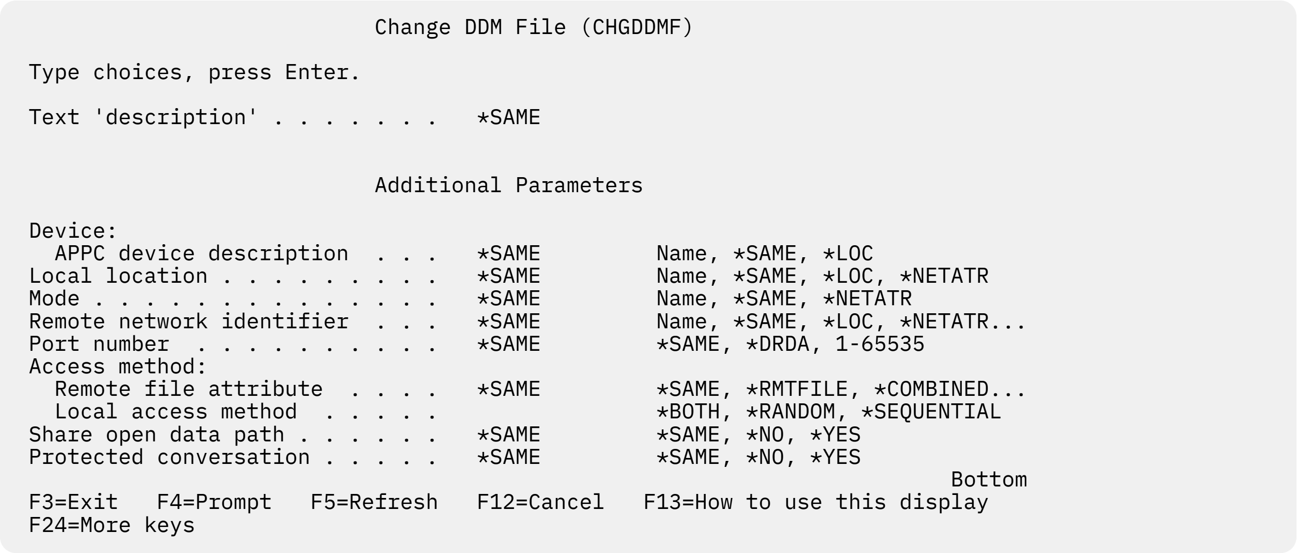
```
                               Work with DDM Files

  Position to  . . . . . .  _____________________

  Type options, press Enter.
    1=Create DDM file   2=Change DDM file   4=Delete   5=Display details
    6=Print details
                                                                     Remote
  Option   Local File               Remote File                      Location
    _       _____________________
    _       WILSON/A                A                                 S36
    2       WILSON/TEST             TESTFILE.TESTLIB                  S38








                                                                       Bottom
  F3=Exit   F5=Refresh   F9=Print list   F12=Cancel
```

Press the Enter key and the Change DDM File display is shown.

For example, if you only want to add a text description, type in the description and press the Enter key. But, if you want to make additional changes, press F10 (Additional parameters), and you can page through the command parameters as they are shown on two displays.

```
                           Change DDM File (CHGDDMF)

Type choices, press Enter.

DDM file . . . . . . . . . . . .    TEST         Name
  Library  . . . . . . . . . . .    WILSON       Name, *LIBL, *CURLIB
Remote file:
  File . . . . . . . . . . . . .   *SAME         Name, *SAME, *NONSTD
    Library  . . . . . . . . . .                 Name, *LIBL, *CURLIB
  Nonstandard file 'name'  . . .



Remote location:
  Name or address  . . . . . . .   *SAME



  Type . . . . . . . . . . . . .   *SAME         *SAME, *SNA, *IP
Record format level check  . . .   *SAME         *SAME, *RMTFILE, *NO
                                                                       More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

If you want to change the mode parameter, type in that value, and then press the Enter key.

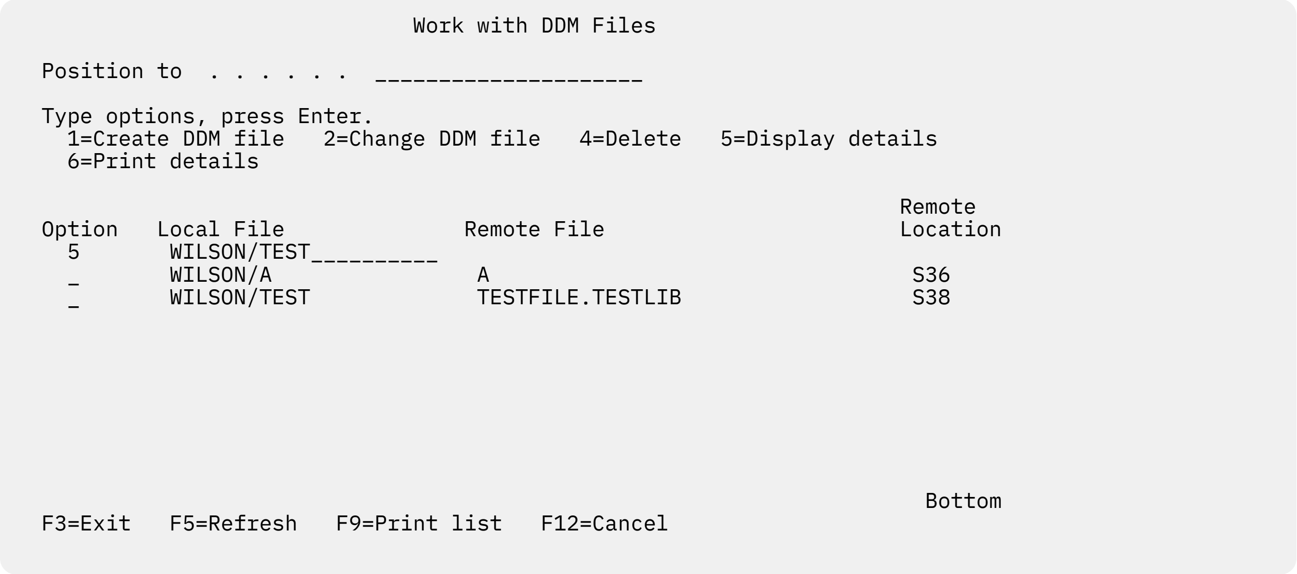


```
                           Change DDM File (CHGDDMF)

Type choices, press Enter.

Text 'description' . . . . . . .   *SAME


                           Additional Parameters

Device:
  APPC device description  . . .   *SAME         Name, *SAME, *LOC
Local location . . . . . . . . .   *SAME         Name, *SAME, *LOC, *NETATR
Mode . . . . . . . . . . . . . .   *SAME         Name, *SAME, *NETATR
Remote network identifier  . . .   *SAME         Name, *SAME, *LOC, *NETATR...
Port number  . . . . . . . . . .   *SAME         *SAME, *DRDA, 1-65535
Access method:
  Remote file attribute  . . . .   *SAME         *SAME, *RMTFILE, *COMBINED...
  Local access method  . . . . .                 *BOTH, *RANDOM, *SEQUENTIAL
Share open data path . . . . . .   *SAME         *SAME, *NO, *YES
Protected conversation . . . . .   *SAME         *SAME, *NO, *YES
                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

After you press the Enter key, you return to the Work with DDM Files display.

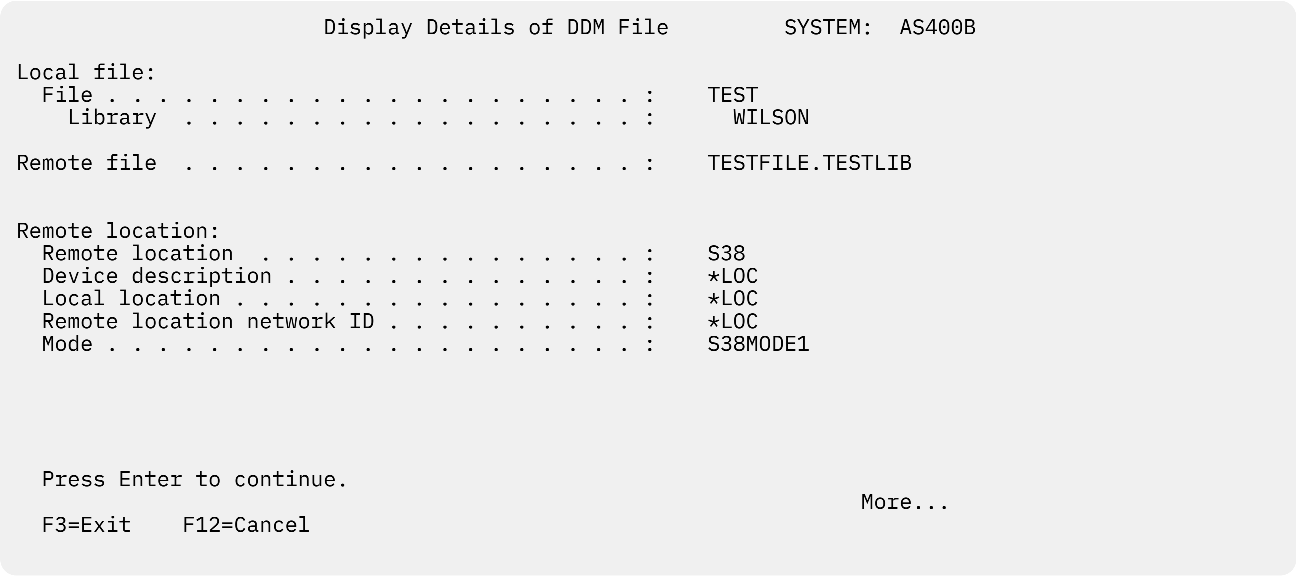
If you want to display the details of a DDM file, type a 5 (Display details) on the Work with DDM Files display next to the file that you want to display, or type the option number in the top list entry of the Options column and specify the local file you want to display. For example, type a 5 (Display details) in the *Option* column and type WILSON/TEST in the *Local File* column of the top list entry.

You can also display the details of a file by using the Display DDM Files (DSPDDMF) command.



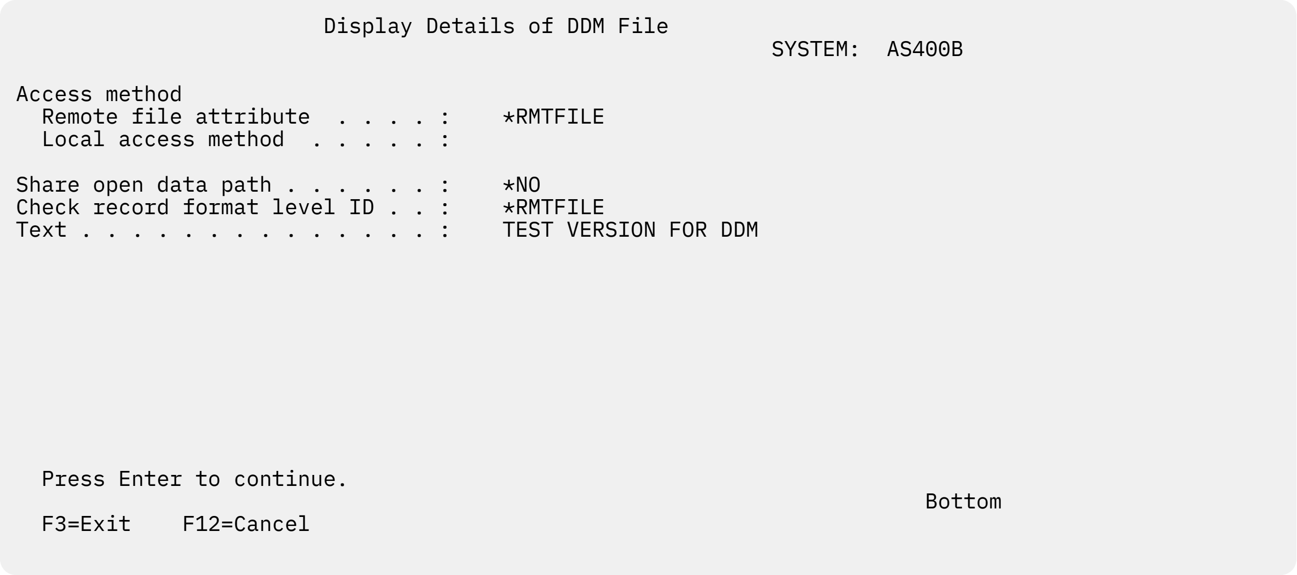
```
                             Work with DDM Files

Position to  . . . . . .  _____________________

Type options, press Enter.
  1=Create DDM file   2=Change DDM file   4=Delete   5=Display details
  6=Print details

                                                                   Remote
Option   Local File              Remote File                       Location
  5       WILSON/TEST__________
  _       WILSON/A                A                                 S36
  _       WILSON/TEST             TESTFILE.TESTLIB                  S38







                                                                     Bottom
F3=Exit   F5=Refresh   F9=Print list   F12=Cancel
```

Press the Enter key and the Display Details of DDM File display is shown.



```
                        Display Details of DDM File         SYSTEM:  AS400B

Local file:
  File . . . . . . . . . . . . . . . . . . . . . :    TEST
    Library  . . . . . . . . . . . . . . . . . . :      WILSON

Remote file  . . . . . . . . . . . . . . . . . . :    TESTFILE.TESTLIB


Remote location:
  Remote location  . . . . . . . . . . . . . . . :    S38
  Device description . . . . . . . . . . . . . . :    *LOC
  Local location . . . . . . . . . . . . . . . . :    *LOC
  Remote location network ID . . . . . . . . . . :    *LOC
  Mode . . . . . . . . . . . . . . . . . . . . . :    S38MODE1





  Press Enter to continue.
                                                                  More...
  F3=Exit    F12=Cancel
```

Page down to see the second display.



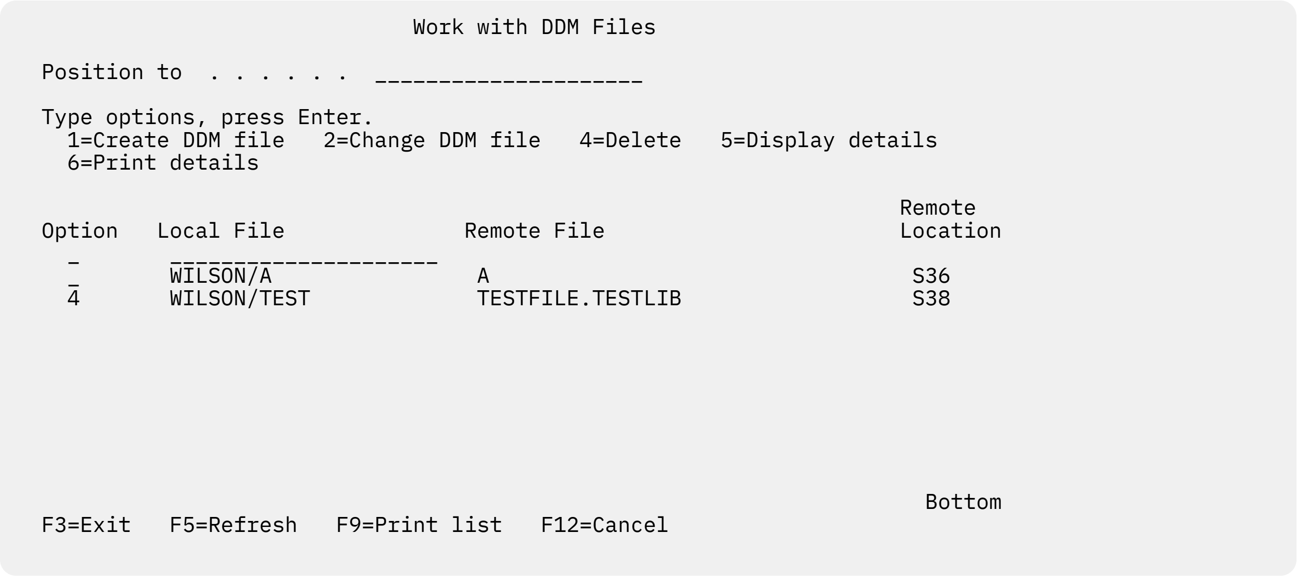
```
                        Display Details of DDM File
                                                           SYSTEM:  AS400B

Access method
  Remote file attribute  . . . . :    *RMTFILE
  Local access method  . . . . . :

Share open data path . . . . . . :    *NO
Check record format level ID . . :    *RMTFILE
Text . . . . . . . . . . . . . . :    TEST VERSION FOR DDM








  Press Enter to continue.
                                                                       Bottom
  F3=Exit    F12=Cancel
```

Press the Enter key to return to the Work with DDM Files display.

In addition to displaying the details of the DDM file, you can *print* the detail information by typing a 6 (Print details) in the *Option* column.

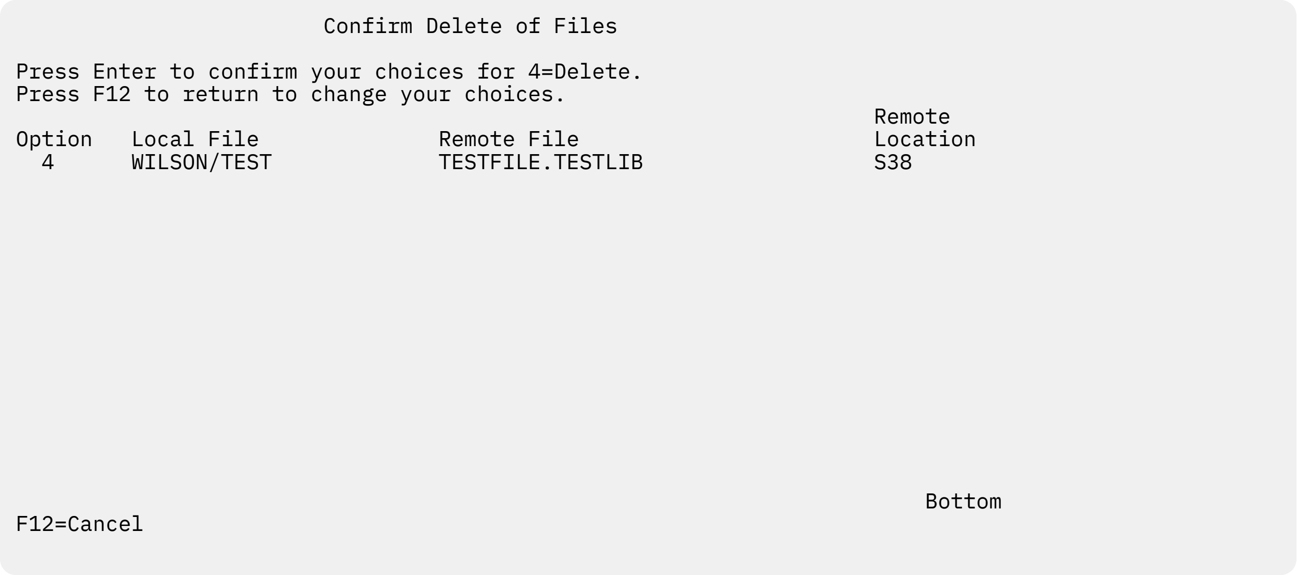
You can also print a list of the DDM files by pressing F9 (Print list).

To *delete* a file or files, type a 4 (Delete) in the *Option* column next to the files you want to delete or in the top list entry and specify the file you want to delete.



```
                             Work with DDM Files

Position to  . . . . . .  _____________________

Type options, press Enter.
  1=Create DDM file   2=Change DDM file   4=Delete   5=Display details
  6=Print details
                                                                   Remote
Option   Local File              Remote File                       Location
  _       _____________________
  _       WILSON/A                A                                 S36
  4       WILSON/TEST             TESTFILE.TESTLIB                  S38







                                                                     Bottom
F3=Exit   F5=Refresh   F9=Print list   F12=Cancel
```

Press the Enter key. You are shown the Confirm Delete of Files display.



```
                        Confirm Delete of Files

Press Enter to confirm your choices for 4=Delete.
Press F12 to return to change your choices.
                                                                   Remote
Option   Local File              Remote File                       Location
  4      WILSON/TEST             TESTFILE.TESTLIB                  S38










                                                                       Bottom
F12=Cancel
```

Choose one of the actions on the display and then press the Enter key. You return to the Work with DDM Files display.

DDM-related CL command considerations

These topics describe DDM-related specifics about CL commands when they are used with DDM files. These topics discuss running the commands on the client system and do not discuss them being submitted to run on the server system by the Submit Remote Command (SBMRMTCMD) command.

Note: You see message CPF9810 if the following items are true about a DDM file:

- The file is created into library QTEMP.
- The file is used by a CL command (such as CPYF).
- A remote file and library was specified in the CL command and the library does not exist on the remote server.

Message CPF9810 indicates that the QTEMP library was not found. However, the library that was not found is the remote library that was specified in the DDM file.

File management handling of DDM files

Because of the way data management handles DDM files, you must be careful when specifying a member name on commands. If a member name is specified, data management first searches for a local database file containing the member specified before looking for a DDM file.

For example, assume the following items:

- DDM file CUST021 is in library NYCLIB.
- Database file CUST021 is in library CUBSLIB.

NYCLIB is listed before CUBSLIB in the user's library list. CUBSLIB/CUST021 contains member NO1. The remote file pointed to by the DDM file contains member NO1. If the following override is used on an **Override with Database File (OVRDBF)** command:

```
OVRDBF FILE(CUST021) MBR(NO1)
```

Data management finds the database file CUBSLIB/CUST021 instead of the DDM file NYCLIB/CUST021.

To avoid this, you can do one of the following things:

- Qualify the TOFILE on the override:

  ```
  OVRDBF FILE(CUST021) TOFILE(NYCLIB/CUST021) MBR(NO1)
  ```

- Remove the library containing the database file from the library list:

  ```
  RMVLIBLE LIB(CUBSLIB)
  ```

- Remove the override and change the remote file name in the DDM file to contain the member name:

  ```
  CHGDDMF FILE(NYCLIB/CUST021)
     RMTFILE(*NONSTD 'XYZ/CUSTMAST(NO1)')
  ```

Allocate Object (ALCOBJ) *command*

When the name of a DDM file is specified on the **Allocate Object (ALCOBJ)** command on the client system, the command allocates the DDM file on the client system and its associated file or file member on a server system.

The command places locks on both the DDM file and the remote file in each pair. (These files are locked on both servers to ensure that they are not changed or deleted while the files or members are locked.) One or more pairs of files (DDM files on the client system and remote files on one or more server systems) can be allocated at the same time.

Each DDM file is always locked with a shared-read (*SHRRD) lock. Shared-read is used for the DDM files regardless of the lock types that might have been specified on the command to lock other local files at the same time.

The lock placed on the *remote file* depends on the type of server system:

- When the target is an IBM i or a System/38, the resulting locks on the remote file are the same as if the file is a local database file. That is, the IBM i or the System/38 remote file is also locked with a shared-read lock, and the member (the one specified, or the first one) is locked with the lock type specified on the command.
- When the target is *not* an IBM i or a System/38, the remote file is locked with the specified lock type, except that some non-IBM i target servers might use a stronger lock than was specified on the command. If an **ALCOBJ** command specifies multiple DDM files, and one or more are on non-IBM i target servers, those remote files are locked with the lock type specified on the command. If a member name is specified for a remote server that does not support members, the lock request is rejected with an error message, unless the member name is the same as the DDM file name.

*Member names and IBM i target servers on the **ALCOBJ** command*
If a member name is specified with the DDM file name on an **ALCOBJ** command, the member (in the remote file) is locked with the lock type specified on the command.

If a member name is also specified in the DDM file itself, the member names on both commands (**ALCOBJ** and **CRTDDMF**) must be the same. If they are different, the lock request is rejected and an error message is sent to the user of the program. The remote file containing the member is locked with a shared-read lock regardless of the lock type specified for the member.

If no member name is specified when a DDM file name is specified on an **ALCOBJ** command for a remote file on an IBM i or a System/38, *FIRST is the default, and the server system attempts to locate and lock the first member in the remote file, the same as if it had been specified by name. If a remote file has no members, the lock request is rejected with an error message.

*Lock multiple DDM files with the **ALCOBJ** command*
One **ALCOBJ** command can be used to specify multiple DDM files that are associated with remote files located on multiple target servers. If it is not possible to lock all the files on all the servers, none are locked.

***ALCOBJ** command completion time with DDM*
When DDM-related files are being allocated, a longer time will be required for the command to complete because of the additional time required for communications to occur between the client and server systems.

You should not, however, increase the wait time specified in the WAIT parameter on the **Allocate Object (ALCOBJ)** command; communications time and the WAIT parameter value have no relationship with each other.

Note: If the **DLTF** command is used to delete the remote file without first releasing (using the **DLCOBJ** command) the locks obtained by the **ALCOBJ** command, the DDM conversation is not reclaimed until the source job has ended.

***Change Job (CHGJOB)** command*
The **Change Job (CHGJOB)** command can be used to change the DDMCNV parameter, which controls whether Advanced Program-to-Program Communication (APPC) or IBM i Access Family conversations allocated for DDM use are to be kept active or automatically dropped when they are not in use by a job. The new value goes into effect immediately for the specified job.

To display the current value of the DDMCNV job attribute, use the **Work with Job (WRKJOB)** command.

***Change Logical File (CHGLF)** command*
The **Change Logical File (CHGLF)** command can be used to change files on the client and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the logical file is changed on the local server.
- When you specify *RMT, the logical file is changed on the remote server. You must specify a DDM file on the FILE parameter.
- When you specify *FILETYPE, a remote file is changed if a DDM file has been specified on the FILE parameter. If a DDM file has not been specified, a local logical file is changed.

Consider the following items when using this command with DDM:

- The FILE parameter is the name of the DDM file that represents the remote logical file being changed. The remote file specified on the DDM file is the logical file that is changed on the remote server (which is also specified in the DDM file).
- For a server system other than an IBM i:
  - All parameters except TEXT are ignored.
  - It is not verified that the remote file is a logical file.

***Change Physical File (CHGPF)** command*

The **Change Physical File (CHGPF)** command can be used to change files on the client and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the physical file is changed on the local system.
- When you specify *RMT, the physical file is changed on the remote system. You must specify a DDM file on the FILE parameter.
- When you specify *FILETYPE, if a DDM file has been specified on the FILE parameter, a remote file is changed. If a DDM file has not been specified, a local physical file is changed.

Consider the following items when using this command with DDM:

- The FILE parameter is the name of the DDM file that represents the remote physical file being changed. The remote file specified in the DDM file is the physical file that is changed on the remote system (which is also specified in the DDM file).
- For a server system other than an IBM i:
  - All parameters except EXPDATE, SIZE, and TEXT are ignored.
  - It is not verified that the remote file is a physical file.

***Change Source Physical File (CHGSRCPF)** command*

The **Change Source Physical File (CHGSRCPF)** command can be used to change files on the client and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the source physical file is changed on the local server.
- When you specify *RMT, the source physical file is changed on the remote server. You must specify a DDM file on the FILE parameter.
- When you specify *FILETYPE, if a DDM file has been specified on the FILE parameter, a remote file is changed. If a DDM file has not been specified, a local source physical file is changed.

Consider the following items when using this command with DDM:

- The FILE parameter is the name of the DDM file that represents the remote source physical file being changed. The remote file specified in the DDM file is the source physical file that is changed on the remote server (which is also specified in the DDM file).
- The CCSID parameter is ignored on a target System/38 server.
- For a server system other than an IBM i, the **CHGSRCPF** command cannot be used to change files.

***Clear Physical File Member (CLRPFM)** command*

The **Clear Physical File Member (CLRPFM)** command can be used with DDM to clear all the records either from a physical file member on a IBM i server system or from a file on a non-IBM i server system. The command works the same way as it does for local files (clearing all data records and deleted records).

Copy commands with DDM

This topic describes the DDM implications of these CL commands.

- **Copy File (CPYF)**
- **Copy from Query File (CPYFRMQRYF)**
- **Copy from Tape (CPYFRMTAP)**
- **Copy Source File (CPYSRCF)**
- **Copy to Tape (CPYTOTAP)**

These commands can be used to copy data or source between files on local and remote servers. You specify with these commands which file to copy from and which file to copy to. The following table shows you what database and device files can be copied between local and remote servers.

Table 10. Copy database and device files

From file	To file
Local or remote database files	Local or remote database files
Local or remote database files	Local device files
Local device files	Local or remote database files

A DDM file is considered a device file that refers to a remote database file. Consider the following items when using these copy commands with DDM:

- DDM conversations are not reclaimed for a job when a copy command produces an error.

  Note: In releases before Version 3 Release 2, copy errors caused the **Reclaim Resources (RCLRSC)** command to be run, which also ran the **Reclaim Distributed Data Management Conversations (RCLDDMCNV)** command. The **RCLRSC** command is still run, but it no longer runs the **RCLDDMCNV** command when a copy error occurs. The DDM conversations will remain unless an explicit **RCLDDMCNV** is specified following the copy command with the error.
- If you specify a DDM file and a local file on the **CPYF** or **CPYSRCF** command, the server does not verify that the remote and local files are not the same file on the client system. If one DDM file is specified, a user can potentially copy to and from the same file.
- A DDM file can be specified on the FROMFILE and the TOFILE parameters for the **CPYF** and **CPYSRCF** commands.

  Note: For the **Copy from Query File (CPYFRMQRYF)**, and **Copy from Tape (CPYFRMTAP)** commands, a DDM file name can be specified only on the TOFILE parameter; for the **Copy to Tape (CPYTOTAP)** command, a DDM file name can be specified only on the FROMFILE parameter.
- If the server system is *not* an IBM i or a System/38:
  - When a file on the local IBM i is copied to a remote file (or vice versa), FMTOPT(*NOCHK) is usually required.
  - When a *source* file on the local IBM i is copied to a remote file (or vice versa), FMTOPT(*CVTSRC) *must* be specified.
- If data is copied to a target System/36 file that has alternative indexes built over it, MBROPT(*REPLACE) cannot be specified. In this case, the copy command attempts to clear the remote file, but it fails because of the alternative indexes.
- When an IBM i file that can contain deleted records is copied to one that cannot contain deleted records, you must specify COMPRESS(*YES), or an error message is sent and the job ends.
- If the remote file name on a DDM file specifies a member name, the member name specified for that file on the copy command must be the same as the member name on the remote file name on the DDM file. In addition, the **Override Database File (OVRDBF)** command cannot specify a member name that is different from the member name on the remote file name on the DDM file.
- If a DDM file does not specify a member name and if the **OVRDBF** command specifies a member name for the file, the copy command uses the member name specified on the **OVRDBF** command.

If the TOFILE parameter is a DDM file that refers to a file that does not exist, **CPYF** creates the file if CRTFILE(*YES) is specified. Listed here are special considerations for remote files created with the **CPYF** or **CPYFRMQRYF** commands:

- If the server system is an IBM i or a System/38, the user profile for the target DDM job must be authorized to the **CRTPF** command on the server system.
- If the server system is a server other than an IBM i, the file specified by the FROMFILE parameter cannot have any file or field CCSIDs other than *HEX or the CCSID of the source job.
- For the **CPYF** command, if the server system is a system other than an IBM i, the FROMFILE parameter cannot be a source file.

- If the server system is a System/38, the TOMBR parameter must be the same as the remote file's name or *FIRST for the copy to be successful. The copy creates a member with the same name as the remote file's name.
- If the server system is other than a System/38 or IBM i, for the copy to be successful, the TOMBR parameter must be *FIRST or specify the DDM file name. For DDM access to the remote file, the file appears to have a member with the same name as the DDM file.
- For an IBM i server system, the TOFILE parameter has all the attributes of the FROMFILE parameter.
- For server systems that are other than IBM is, those attributes on the **CRTPF** command that are ignored are also ignored when the copy command creates the file.
- If the server system is a System/38 and the FROMFILE parameter is a direct file that does not allow deleted records, an attempt is made to copy the records after the last record for the file at its maximum size. The system operator on the System/38 tells the server to either add the records or cancel the copy.
- The **CPYF** or **CPYFRMQRYF** command with CRTFILE(*YES) creates a file on the server system with a size description that is only as large as the server system allows.
- For all copies, if the number of records being copied exceeds the maximum allowed by the to-file, the copy function ends when the maximum is reached.
- For copy commands executed on Version 2 Release 3 or earlier systems that reference a Version 3 Release 1 remote file having a constraint relationship, the ERRLVL parameter will not work for constraint relationship violations. The copy ends regardless of the ERRLVL specified.
- The copy commands allow copying from and to DDM files that reference remote distributed files.

Create Data Area (CRTDTAARA) command
The Create Data Area (CRTDTAARA) command creates a data area and stores it in a specified library. It also specifies the attributes of the data. The data area can be optionally initialized to a specific value.

You can create a DDM data area by specifying *DDM on the TYPE parameter. The DDM data area is used as a reference data area by programs to access data areas located on a remote (server) system in the DDM network. Programs on the local (client) system reference a remote data area by the DDM data area's name, not by the remote data area's name. (The DDM data area name can be the same as the remote data area name.)

The DDM data area (on the client system) contains the name of the remote data area and the name of the remote (server) system on which the remote data area is located.

The DDM data area can be used with the Retrieve Data Area (RTVDTAARA) command and the Change Data Area (CHGDTAARA) command to retrieve and update data areas on remote servers. A DDM data area can also be used with the Retrieve Data Area (QWCRDTAA) API.

Consider the following items when using this command with DDM:

- The RMTDTAARA parameter is the name of the remote data area on the target server. The data area does not need to exist when the DDM data area is created.
- The RMTLOCNAME parameter is the name of the remote location that is used with this object. Multiple DDM data areas can use the same remote location for the server system. RMTLOCNAME must point to a server system that is a IBM i product running at a release of IBM i that supports remote data areas. The possible values for RMTLOCNAME include:
  - *remote-location-name*: Specifies the name of the remote location that is associated with the server system. The remote location, which is used in accessing the server system, does not need to exist when the DDM data area is created, but it must exist when the DDM data area is accessed.
  - *RDB: The remote location information for the relational database entry specified in the relational database (RDB) parameter is used to determine the remote system.
- The DEV parameter is the name of the APPC device description on the source server that is used with this DDM data area. The device description does not need to exist when the DDM data area is created.
- The LCLLOCNAME parameter is the local location name.

- The MODE parameter is the mode name that is used with the remote location name to communicate with the server system.
- The RMTNETID parameter is the remote network ID in which the remote location resides that is used to communicate with the server system.

Consider the following restrictions when using this command with DDM:

- You cannot create a DDM data area using the names *LDA, *GDA, or *PDA.
- You cannot create a data area remotely. This function can be done remotely by using the Submit Remote Command (SBMRMTCMD) command.
- You can remotely display data areas by using the SBMRMTCMD command.
- You can display the contents of remote data areas by using the Display Data Area (DSPDTAARA) command; specify *RMT on the SYSTEM parameter. The data in the data area is displayed in the same format as that used for local data areas, with the exception of the TEXT field, which is the text description provided when the DDM data area was created. If you specify *LCL on the SYSTEM parameter for a DDM data area, the output looks similar to the following display:

```
Data area . . . . . . . :     DDMDTAARA
Library . . . . . . . :        DDMLIB
Type    . . . . . . . . . :   *DDM
Length    . . . . . . . . :   62
Text    . . . . . . . . . :   'This is a DDM data area'

          Value
Offset        *...+....1....+....2....+....3....+....4....+....5
   0         '*LOC      *NETATR SYSTEMA *LOC    *LOC    LCLDTAAR'
  50         'A LCLLIB    '
```

Use the following chart to interpret the values:

Table 11. Offset values

Offset	**DDMDTAARA fields**
1-10	DEV
11-18	MODE
19-26	RMTLOCNAME
27-34	LCLLOCNAME
35-42	RMTNETID
43-52	RMTDTAARA (name)
53-62	RMTDTAARA (library)

Create Data Queue (CRTDTAQ) command

The Create Data Queue (CRTDTAQ) command creates a data queue and stores it in a specified library. Data queues are used to communicate and store data used by several programs either within a job or between jobs. Multiple jobs can send or receive data from a single queue.

The CRTDTAQ command can optionally create a distributed data management (DDM) data queue. This is done by specifying *DDM on the TYPE parameter. The DDM data queue is used as a reference data queue by programs to access data queues located on a remote (server) system in the DDM network. Programs on the local (client) system reference a remote data queue by the DDM data queue's name, not by the remote data queue's name. (The DDM data queue name, however, can be the same as the remote data queue name.)

The DDM data queue (on the client system) contains the name of the remote data queue and the name of the remote (server) system on which the remote data queue is located.

Consider the following items when using this command with DDM:

- The TYPE parameter specifies the type of data queue to be created. A standard data queue or a DDM data queue can be created.
- The RMTDTAQ parameter is the name of the remote data queue on the server system. The data queue does not need to exist when the DDM data queue is created.
- The RMTLOCNAME parameter is the name of the remote location that is used with this object. Multiple DDM data areas can use the same remote location for the server system. RMTLOCNAME must point to a server system that is a IBM i product running at a release of IBM i that supports remote data areas. The possible values for RMTLOCNAME include:
  - *remote-location-name*: Specifies the name of the remote location that is associated with the server system. The remote location, which is used in accessing the server system, does not need to exist when the DDM data area is created, but it must exist when the DDM data area is accessed.
  - *RDB: The remote location information for the relational database entry specified in the relational database (RDB) parameter is used to determine the remote system.
- The DEV parameter is the name of the APPC device description on the client system that is used with this DDM data queue. The device description does not need to exist when the DDM data queue is created.
- The LCLLOCNAME parameter is the local location name.
- The MODE parameter is the mode name that is used with the remote location name to communicate with the server system.
- The RMTNETID parameter is the remote network ID in which the remote location resides that is used to communicate with the server system.

Consider the following restrictions when using this command with DDM:

- Only the API interface for data queues is supported when using DDM data queues. The following APIs are supported:
  - Send to Data Queue (QSNDDTAQ)
  - Receive from Data Queue (QRCVDTAQ)
  - Clear Data Queue (QCLRDTAQ)

  The Retrieve Data Queue Description (QMHQRDQD) and Retrieve Data Queue Messages (QMHRDQM) APIs are not supported for DDM data queues.

  When using the *ASYNC parameter on the Send Data Queue API, messages resulting from errors encountered when accessing the remote data queue are placed in the server system's job log, and a DDM protocol error (CPF9173 - `Error detected in DDM data stream by server system`) is posted in the client system's job log. Look in the server system's job log for the cause of the error and correct the problem before using the remote data queue. Attempts to access the remote data queue after you receive this error message without first correcting the problem will produce unpredictable results.
- You cannot create a data queue remotely. This function can be done remotely by using the Submit Remote Command (SBMRMTCMD) command.

Create Logical File (CRTLF) *command*

The **Create Logical File (CRTLF)** command can be used to create files on the client and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the file is created on the local system.
- When you specify *RMT, the file is created on the remote system. You must specify a DDM file on the FILE parameter.
- When you specify *FILETYPE, if a DDM file has been specified on the FILE parameter, a remote file is created. If a DDM file has not been specified, a local file is created.

Consider the following items when using this command with DDM:

- The parameter FILE is the name of the DDM file that represents the remote logical file being created. The remote file specified in the DDM file is the logical file that is created on the remote system (which is also specified in the DDM file).
- The OPTION and GENLVL parameters have no effect on the remote command sent.
- The files specified on the PFILE or JFILE keywords in the DDS for the logical file must be at the same location as the logical file that is created.
- If *JOB is specified as the value of a parameter or is in the data description specification (DDS) for that file, the attribute of that source job is used for file and field attributes. The attribute of the source job is also used when the default for a file or field attribute is the job attribute.
- For a server system other than an IBM i:
  - The format name is ignored.
  - Only the value of *ALL is supported for the DTAMBRS parameter.
  - These parameters are ignored:
    - AUT
    - FRCRATIO
    - FRCACCPTH
    - LVLCHK
    - MAINT
    - MBR
    - RECOVER
    - SHARE
    - UNIT
    - WAITFILE
    - WAITRCD

    Note: For System/38 targets, the **SBMRMTCMD** command can be used to change these attributes.
  - Only the value of *NONE is supported for the FMTSLR parameter.
  - FILETYPE must be *DATA.
  - If a member name is specified, it must match the DDM file name.
- For an IBM i server system:
  - All parameters of the **CRTLF** command are supported with one restriction: authorization lists are not allowed for the AUT (public authority) parameter. DDM cannot guarantee the existence of the authorization list on the server system or that the same user IDs are in the list if it does exist. The public authority is changed to *EXCLUDE when you use an authorization list as a value for the AUT parameter of the **CRTLF** command.
  - The file names specified in the DTAMBRS parameter must be the names of the DDM files that represent the remote based-on physical files. If a member name was specified as part of the remote file name of the DDM file, then only that member name can be specified. The member names must be the actual remote file member names.

Create Physical File (CRTPF) *command*

The **Create Physical File (CRTPF)** command can be used to create files on the client and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the file is created on the local server.
- When you specify *RMT, the file is created on the remote server. You must specify a DDM file on the FILE parameter.

- When you specify *FILETYPE, if a DDM file has been specified on the FILE parameter, a remote file is created. If a DDM file has not been specified, a local file is created.

Consider the following items when using this command with DDM:

- The FILE parameter is the name of the DDM file that represents the remote file being created. The remote file specified in the DDM file is the file that is created on the remote server (which is also specified in the DDM file).
- The OPTION and GENLVL parameters create the same results as for local processing. These parameters have no effect on the remote command sent.
- If *JOB is specified as the value of a parameter or is in the data description specification (DDS) for that file, the attribute of that source job is used for file and field attributes. The attribute of the source job is also used when the default for a file or field attribute is the job attribute.
- For a server system other than an IBM i:
  - The format name is ignored.
  - These parameters are ignored:
    - AUT
    - CONTIG
    - DLTPCT
    - FRCRATIO
    - FRCACCPTH
    - LVLCHK
    - MAINT
    - MAXMBRS2
    - MBR
    - RECOVER
    - REUSEDLT
    - SHARE
    - UNIT
    - WAITFILE
    - WAITRCD

    Note: For System/38 targets, the **SBMRMTCMD** command can be used to change these attributes.
  - FILETYPE must be *DATA.
  - All other parameters are supported.
  - If a member name is specified, it must match the DDM file name.
  - The only CCSID values that are supported are:
    - *HEX
    - 65535
    - *JOB
    - Process CCSID of the source job

    The file is not created if any other CCSID value is specified.
  - When the DDS keyword VARLEN is used, DDM tries to create a variable-length record file on the server system. There are some specific rules for this keyword.
- On an IBM i server system, all parameters of the **CRTPF** command are supported with one restriction: authorization lists are not allowed for the AUT (public authority) parameter. DDM cannot guarantee the existence of the authorization list on the server system or that the same user IDs are in the list if it does

exist. The public authority is changed to *EXCLUDE when you use an authorization list as a value for the AUT parameter of the **CRTPF** command.

Create Source Physical File (CRTSRCPF) *command*

The **Create Source Physical File (CRTSRCPF)** command can be used to create files on the IBM i source and server systems through the SYSTEM parameter.

Consider the following items when using the SYSTEM parameter values:

- When you specify *LCL, the file is created on the local system.
- When you specify *RMT, the file is created on the remote system. You must specify a DDM file on the FILE parameter.
- When you specify *FILETYPE, if a DDM file has been specified on the FILE parameter, a remote file is created. If a DDM file has not been specified, a local file is created.

Consider the following items when using this command with DDM:

- The FILE parameter is the name of the DDM file that represents the remote file being created. The remote file specified in the DDM file is the file that is created on the remote system (which is also specified in the DDM file).
- The OPTION and GENLVL parameters create the same results as for local processing. These parameters have no effect on the remote command sent.
- If *JOB is specified as the value of a parameter or is in the data description specification (DDS) for that file, the attribute of that source job is used for file and field attributes. The attribute of the source job is also used when the default for a file or field attribute is the job attribute.

All parameters of the CRTSRCPF command are supported with one restriction: authorization lists are not allowed for the AUT (public authority) parameter. DDM cannot guarantee the existence of the authorization list on the server system or that the same user IDs are in the list if it does exist. The public authority is changed to *EXCLUDE when you use an authorization list as a value for the AUT parameter of the CRTSRCPF command.

Deallocate Object (DLCOBJ) *command*

When the name of a DDM file is specified on the **Deallocate Object (DLCOBJ)** command on the client system, the command deallocates the DDM file on the client system and its associated file or file member on a server system.

The command releases the locks that were placed on the paired files on both the client and server systems by the **Allocate Object (ALCOBJ)** command. One or more pairs of files (DDM files on the client system and remote files on one or more server systems) can be deallocated at the same time.

*Member names and IBM i target servers on the **DLCOBJ** command*

All of the information previously discussed in the **ALCOBJ** command description regarding member names applies to the **DLCOBJ** command as well.

Refer to the ALCOBJ command description for the details.

*Unlock multiple DDM files on the **DLCOBJ** command*

One **DLCOBJ** command can be used to specify multiple DDM files that are associated with remote files that might be located on multiple server systems. In most cases, the command attempts to release as many of the specified locks as possible.

For example:

- If one of the DDM files specified on the **DLCOBJ** command refers to a remote file that is not a database file, that lock is not released; but the specified locks on the remote files associated with the other DDM files specified are released if, of course, they are valid.
- If a user tries to release a lock that he did not place on a file by a previous **ALCOBJ** command, that part of the request is rejected and an informational message is returned to the user.

Delete File (DLTF) *command*

The **Delete File (DLTF)** command can be used to delete files on the client and server systems.

The following items should be considered when using the SYSTEM parameter values:

- When you specify *LCL, only local files are deleted. This might include DDM files.
- When you specify *RMT, the file is deleted on the remote server. You must specify a DDM file on the FILE parameter. If a generic name is specified, the remote files corresponding to any DDM files matching the generic name are deleted. (The local DDM files are not deleted.)
- When you specify *FILETYPE, if a DDM file has been specified, the remote file is deleted. If a DDM file has not been specified, the local file is deleted. When you specify generic names, local non-DDM files are deleted first. Remote files for any DDM files matching the generic name are then deleted. Local DDM files are not deleted.

Notes:

1. Structured Query Language/400 (SQL/400) DROP TABLE and DROP VIEW statements work only on local files.
2. If the DLTF command is used to delete the remote file without first releasing (using the **DLCOBJ** command) the locks obtained by the **ALCOBJ** command, the DDM conversation is not reclaimed until the source job has ended.

Display File Description (DSPFD) *command*

The **Display File Description (DSPFD)** command can be used to display (on the client system) the attributes of the DDM file on the client system, the remote file on the server system, or both the DDM file and the remote file. As with local files, the attributes of multiple DDM files, multiple remote files, or both can be displayed by the same command.

Note: Although this discussion mentions only one server system, the files for multiple server systems can be displayed at the same time.

The SYSTEM parameter determines which group of attributes is displayed.

- To display the attributes of DDM files, which are *local* files, the SYSTEM parameter must specify *LCL (the default). If SYSTEM(*LCL) is specified:
  - The FILEATR parameter must either specify *DDM (to display DDM file attributes only) or default to *ALL (to display all file types, including DDM files). The same kind of information is displayed for DDM files (which are on the local system) as for any other types of files on the local server.
  - If FILEATR(*DDM) is specified and the OUTFILE parameter specifies a file name, only local DDM file information is given.
- To display the attributes of remote files, the SYSTEM parameter must specify *RMT. If SYSTEM(*RMT) is specified:
  - The FILEATR parameter must specify *ALL, *PHY, or *LGL.
  - The type of information displayed for remote files depends on what type of server system the files are on. If the target is an IBM i or a System/38, the same type of information displayed for local files on an IBM i or a System/38 can be displayed. If the target is *not* an IBM i or a System/38, all the information that can be obtained through that server's implementation of the DDM architecture that is compatible with the IBM i's implementation is displayed.
- To display the attributes of both DDM and remote files, the SYSTEM parameter must specify *ALL.

Display File Field Description (DSPFFD) *command*

The **Display File Field Description (DSPFFD)** command can be used to display the file, record format, and field attributes of a remote file. To display the remote file attributes, however, you must enter the name of the DDM file associated with the remote file, not the name of the remote file.

Note: Because the DDM file has no field attributes, the **DSPFFD** command cannot specify SYSTEM(*LCL) to display local DDM file information.

If *ALL or a generic file name is specified on the FILE parameter, the **DSPFFD** command can also display information about a group of both local files and remote files, or just a group of local files. In this case, the SYSTEM parameter determines which are displayed.

- To display the attributes of *local non-DDM* files only, the SYSTEM parameter need not be specified because *LCL is the default.
- To display the attributes of *remote* files, the SYSTEM parameter must specify *RMT. If SYSTEM(*RMT) is specified, the field and record format information displayed for remote files depends on what type of server system the files are on.
  - If the target is an IBM i or a System/38, the same information displayed for local files on an IBM i is displayed.
  - If the target is other than a System/38 or IBM i:
    - Fields are F*nnnnn* or K*nnnnn* (where *nnnnn* is some number), based on whether the file is a keyed file or not.
    - The record format name is the DDM file name.

    If the remote file has a record length class of record varying or initially varying, fixed-length field descriptions are displayed.
- To display the attributes of both *local non-DDM files and remote files*, the SYSTEM parameter must specify *ALL. Only remote physical and logical files can be displayed.

Open Query File (OPNQRYF) *command*

You can query remote files using the **Open Query File (OPNQRYF)** command, but only if the remote files are on a target IBM i or a target System/38.

If multiple remote files are specified on one **OPNQRYF** command, they must all exist on the same server system and use the same remote location information.

If the server system is an IBM i or a System/38, a query request is created and sent to the server system using the DDM file that the query refers to. If the server system is other than an IBM i or a System/38, the query request cannot be processed and an error message is created. However, the query utility on the System/38 can be used to query remote files that are other than IBM i files.

If the server system is a System/38 and the source is an IBM i, or if the server system is an IBM i and the source is a System/38, **OPNQRYF** cannot use group-by and join functions. An error results.

Override with Database File (OVRDBF) *command*

The **Override with Database File (OVRDBF)** command can be used with DDM to override (replace) a local database file named in the program with a DDM file; the DDM file causes the associated remote file to be used by the program instead of the local database file.

If a DDM file is specified on the TOFILE parameter and if other parameters are specified that change a file's attributes, the result is that the remote file actually used by the program is used with its attributes changed by the parameter values specified on the **OVRDBF** command.

If the server system is an IBM i or a System/38, existing programs that use the **OVRDBF** command to access remote files work the same as when they access local files. All the **OVRDBF** parameters are processed the same on source and target IBM is.

If end-of-file delay (EOFDLY) is used, it is recommended to end a job with an end-of-file record because if the source job gets canceled, the target job does not get notified. The user must also end the target job.

If the server system is neither an IBM i nor a System/38:

- The following parameters are still valid: TOFILE, POSITION, RCDFMTLCK, WAITFILE, WAITRCD, LVLCHK, EXPCHK, INHWRT, SECURE, SHARE, and SEQONLY.
  - The TOFILE parameter is always processed on the client system. When a DDM file name is specified on this parameter, the program uses the associated remote file instead of the local database file specified in the program.

  - The RCDFMTLCK parameter, if specified, is valid only if both of the following are true of the remote file used: only one type of lock condition can be requested for the remote file, and the record format name in the remote file must be the same as the name of the DDM file.
  - The WAITFILE and WAITRCD parameters have no effect on remote file processing.
- The MBR parameter causes an error if it is specified with a member name that is different than the name of the file containing the member.
- The FRCRATIO and NBRRCDS parameters, if specified, are ignored.
- The FMTSLR parameter, if specified, causes an error when the file being opened is a DDM file.
- The SEQONLY parameter causes records to be blocked on the source side. Records might be lost if the source job is canceled before a block is full.

Reclaim Resources (RCLRSC) *command*

The **Reclaim Resources (RCLRSC)** command, like the **Reclaim DDM Conversations (RCLDDMCNV)** command, can be used to reclaim all DDM conversations that currently have no users in the job.

This can be done even if the DDMCNV job attribute is *KEEP. The **RCLRSC** command, however, first attempts to close any unused files for the appropriate recursion levels, as it would for local files. This action might result in some conversations allocated to DDM being unavailable for the job. For example, if a DDM file is opened using the **Open Database File (OPNDBF)** command, the **RCLRSC** command closes the file and reclaims the conversation.

After the files are closed, any unused DDM conversations are dropped. Whether a conversation can be reclaimed is not affected by the recursion level or activation group in which the **RCLRSC** command is issued.

Rename Object (RNMOBJ) *command*

The **Rename Object (RNMOBJ)** command can be used to rename a remote file.

The following items should be considered when using the SYSTEM parameter values:

- When you specify *LCL, local objects are renamed. This might include DDM files.
- When you specify *RMT, this value applies only to OBJTYPE(*FILE). The DDM file containing the remote file to be renamed is specified on the OBJ parameter.

  The DDM file containing the new name for the remote file is specified on the NEWOBJ parameter. Both DDM files must already exist in the same library (on the client system). The two DDM files must refer to the same server systems and contain the same remote location information. Neither the two local DDM files nor the RMTFILE names in the two DDM files are changed. Specify *LCL to rename the DDM file or use the **Change DDM File (CHGDDMF)** command to change the RMTFILE name in a DDM file.
- When you specify *FILETYPE, this value applies only to OBJTYPE(*FILE). If the file specified in the OBJ parameter is a DDM file, the rules when specifying *RMT apply. If the file is not a DDM file, the rules when specifying *LCL apply.

When renaming remote files for IBM i and System/38 targets, if library names have been specified in the RMTFILE parameter for the two DDM files, the library names must be the same but the file names must be different.

Work with Job (WRKJOB) command

The Work with Job (WRKJOB) command can be used to display two DDM-related items.

These items include:

- The DDMCNV job attribute for the source job.
- The object lock requests (held locks and pending locks) for DDM files that are being used in the client system job. These are shown by choosing option 12 (Work with locks, if active) from the Work with Job menu.

The Job Locks display shows only the locks held for the local DDM files; locks for remote files are not shown. Also, because DDM files do not have members, none are indicated on this display nor on the Member Lock display.

The IBM i operating system does not display any locks for remote files. Locks for the remote file, its members, or its records cannot be displayed by the client system. However, these remote locks can be displayed using pass-through.

The lock condition shown for DDM files is always shared read (*SHRRD) regardless of the lock conditions used for their associated remote files or members.

Work with Object Lock (WRKOBJLCK) *command*

The **Work with Object Lock (WRKOBJLCK)** command can be used to display the object lock requests (held locks and pending locks) for DDM files. This command displays only the locks held for the local DDM files, not locks held for the associated remote files.

An IBM i does not display any locks for remote files; locks for the remote file, its members, or its records cannot be displayed by the client system.

The lock condition shown for DDM files is always shared read (*SHRRD) regardless of the lock conditions used for their associated remote files or members.

DDM-related CL parameter considerations

This topic covers parameter considerations that apply to DDM-related CL commands.

Note: The **Create DDM File (CRTDDMF)** command can be used to create a DDM file. The other create file commands such as **CRTPF** or **CRTxxxF** cannot be used to create a DDM file.

DDMACC parameter considerations

The DDMACC parameter controls how an IBM i, as a server system, handles DDM requests from other servers.

The DDMACC parameter is used on the **Change Network Attributes (CHGNETA)**, **Display Network Attributes (DSPNETA)**, and **Retrieve Network Attributes (RTVNETA)** commands. The value of this server-level parameter determines whether this IBM i can accept DDM requests from other servers.

DDMCNV parameter considerations

The DDMCNV parameter is a job-related parameter that controls whether Advanced Program-to-Program Communication (APPC) or IBM i conversations in the job that is allocated for DDM use (that is, DDM conversations) are to be automatically dropped or kept active for the source job.

The default is to keep the conversation active.

This parameter can drop a conversation when it has no active users. The conversation is unused when:

1. All the DDM files and remote files used in the conversation are closed and unlocked (deallocated).
2. No other DDM-related functions (like the **Submit Remote Command (SBMRMTCMD)** command or the **Display File Description (DSPFD)** command to access the server system) are being done.
3. No DDM-related function has been interrupted (by a break program, for example) while running.

The DDMCNV parameter values are:

***KEEP**

Specifies that once each DDM conversation is started for the source job it is kept active at the completion of a source program request, and it waits for another request to be received from the same or another program running in the job. This is the default value.

***DROP**

Specifies that each DDM conversation started in the source job is to be automatically dropped when both of the following items are true: no request from the client system program(s) running in the job is being processed in the conversation, and all the DDM files are closed and all objects that were allocated are now deallocated.

The DDMCNV parameter is changed by the **Change Job (CHGJOB)** command and is displayed by the **Work with Job (WRKJOB)** command. Also, if the **Retrieve Job Attributes (RTVJOBA)** command is used, you can get the value of this parameter and use it within a CL program.

OUTFILE parameter considerations for DDM

The OUTFILE parameter is used on such commands as the **Display File Description (DSPFD)**, the **Display File Field Description (DSPFFD)**, the **Display Object Description (DSPOBJD)**, and the **Create Auto Report Program (CRTRPTPGM)**. The parameter identifies a database file into which output data created by the command is stored.

When the name of a DDM file is specified on the OUTFILE parameter of these commands, two restrictions apply:

- The remote server must be an IBM i or a System/38. This is necessary to ensure that the associated remote file has the proper format for the output data.
- The remote file associated with the DDM file must already exist. If the remote file does not exist, a message is returned to the user indicating that the remote file must exist before the function can be performed.

If the remote file named on the OUTFILE parameter does exist and is on an IBM i or a System/38, the file will be checked for three conditions before it can be used as an output database file to store displayed output:

- The remote file must be a physical file.
- The remote file must not be a model output file. That is, it cannot be one of the model output files provided with the IBM i operating system which has the required format, but no data.
- The record format name in the remote file must match the model output file record format name. (This condition requires that the remote system be an IBM i or a System/38.)

If all of these conditions are met, the remote file member is cleared. (Output file members *must* be cleared before they can be used again.) If the remote file member does not exist, it is created and the output is stored in it.

DDM-related CL command lists

The control language (CL) commands that have a specific relationship with DDM are grouped in charts in these topics to show the command functions that are available with DDM, those having common limitations when used with DDM, and those that cannot be used with DDM.

Notes:

1. Not all of the CL commands on an IBM i are shown in these topics. Only those that are intended (or recommended) by IBM for use with DDM or those specifically not intended for DDM use are shown. The intended use can be either for commands that are run on the client system that affect a remote file on the server system, or for commands that are submitted to the server system by using the Submit Remote Command (SBMRMTCMD) command to run there.
2. Some of these commands appear in more than one of the following charts.

The charts in these topics show:

- Commands affecting only the DDM file:

  Object-oriented commands that can be used with DDM files, but do not affect the associated remote files. The Create DDM File (CRTDDMF), Change DDM File (CHGDDMF), and Reclaim DDM Conversations (RCLDDMCNV) commands are included in this group.
- Commands affecting both the DDM file and the remote file:
  - File management commands that require that the server system be another IBM i or a System/38. The SBMRMTCMD command is included in this group.
  - Member-related commands that can be used in some way on remote files.
  - Source file commands that can operate on source files while DDM is being used.

These commands, normally used for processing local files, can (transparently to the programs) process remote files when one of their parameters specifies the name of a DDM file.

- Commands that cannot be used with DDM.

Many of these commands, when limited as shown in the charts, can still be submitted by the SBMRMTCMD command to a target server (an IBM i or a System/38 only) to run, but it might not be useful to do so.

Object-oriented commands with DDM
The DDM file object on the source IBM i can be accessed by these object-oriented CL commands. These commands work with DDM files as they normally do with any other files on the local server.

Some of these commands can operate on more than one object, and one or more of them might be DDM files if, for example, a generic file name is specified.

Except as noted in the chart, these commands have no effect on the remote file associated with the DDM file; that is, no reference is made over a communications line to the server system when one of these commands specifies a DDM file.

However, if you do want one of these commands to operate on a remote file (instead of the DDM file), you can use the **Submit Remote Command (SBMRMTCMD)** command to submit the command to run on the server system, if it is an IBM i or a System/38. The results of running the submitted command, in this case, are not sent back to the source server, except for some indication to the client system user (normally a message) about whether the function was performed successfully.

Command name	Descriptive name
CHGDDMF	**Change DDM File**
CHGLF[1,2,3,4]	**Change Logical File**
CHGOBJOWN	**Change Object Owner**
CHGPF[1,2,3,4]	**Change Physical File**
CHGSRCPF[1,2,3,4]	**Change Source Physical File**
CHKOBJ	**Check Object**
CRTDDMF	**Create DDM File**
CRTDUPOBJ	**Create Duplicate Object**
CRTLF[1,2,3]	**Create Logical File**
CRTPF[1,2,3]	**Create Physical File**
CRTSRCPF[1,2,3]	**Create Source Physical File**
CRTS36CBL[6]	**Create S/36 COBOL Program**
CRTS36DSPF[7]	**Create S/36 Display File**
CRTS36MNU[7]	**Create S/36 Menu**
CRTS36MSGF[7]	**Create S/36 Message File**
CRTS36RPG[6]	**Create S/36 RPG II Program**
CRTS36RPGR[7]	**Create Console Display File**
CRTS36RPT[6]	**Create S/36 RPG II Auto Report**
DLTF[1,2,3]	**Delete File**
DMPOBJ	**Dump Object**
DMPSYSOBJ	**Dump System Object**
DSPFD[1,2,3]	**Display File Description**

Command name	Descriptive name
DSPFFD[1,2,3]	**Display File Field Description**
DSPOBJAUT	**Display Object Authority**
DSPOBJD	**Display Object Description**
GRTOBJAUT	**Grant Object Authority**
MOVOBJ	**Move Object**
RCLDDMCNV	**Reclaim DDM Conversations**
RNMOBJ[1,2,3]	**Rename Object**
RSTLIB	**Restore Library**
RSTOBJ	**Restore Object**
RVKOBJAUT	**Revoke Object Authority**
SAVCHGOBJ	**Save Changed Object**
SAVLIB	**Save Library**
SAVOBJ	**Save Object**
WRKJOB[5]	**Work with Job**
WRKOBJLCK[5]	**Work with Object Lock**

[1] When run on the client system, this command does not refer to the remote file when SYSTEM(*LCL) is used.

[2] The remote operation is performed if SYSTEM(*RMT) is specified, or if SYSTEM(*FILETYPE) is specified and the file is a DDM file.

[3] Because DDM file names can be specified on these commands, the **SBMRMTCMD** command is not needed to perform these functions on a target IBM i or a target System/38.

[4] The target must be an IBM i at release 3.0 and above or support Level 2.0 of DDM architecture.

[5] When run on the client system, this command displays any locks on the DDM file, not on the remote file.

[6] This System/36 environment command is supported by DDM.

[7] This System/36 environment command is *not* supported by DDM.

Target IBM i-required file management commands
These CL commands can be used only when the server system is another IBM i or System/38 platform.

Command name	Descriptive name
ADDLFM[1]	**Add Logical File Member**
ADDPFM	**Add Physical File Member**
CHGLFM	**Change Logical File Member**
CHGPFM	**Change Physical File Member**
CPYSRCF	**Copy Source File**
INZPFM	**Initialize Physical File Member**

Command name	Descriptive name
OPNQRYF	**Open Query File**
RGZPFM	**Reorganize Physical File Member**
RMVM	**Remove Member**
RNMM	**Rename Member**

[1] The server system must running the IBM i operating system.

Because DDM file names can be specified on these commands, the Submit Remote Command (SBMRMTCMD) command is not needed to perform these functions on an IBM i or a System/38 server system.

Member-related commands with DDM
Database file operations that apply to a member can be used by DDM.

When the name of a DDM file is specified on any of the following CL commands, IBM i DDM accesses the remote file (and member) referred to by the DDM file. However, as indicated in the chart, some of these commands are valid only when the remote file is on an IBM i or a System/38.

Command name	Descriptive name
ADDPFM[1]	**Add Physical File Member**
ADDLFM[6]	**Add Logical File Member**
ALCOBJ	**Allocate Object**
CHGLFM[1]	**Change Logical File Member**
CHGPFM[1]	**Change Physical File Member**
CLOF	**Close File**
CLRPFM	**Clear Physical File Member**
CPYF[2]	**Copy File**
CPYFRMTAP	**Copy From Tape**
CPYSPLF	**Copy Spooled File**
CPYSRCF[1]	**Copy Source File**
CPYTOTAP	**Copy To Tape**
DCLF	**Declare File**
DLCOBJ	**Deallocate Object**
DSPFD[3]	**Display File Description**
DSPFFD[3]	**Display File Field Description**
DSPPFM	**Display Physical File Member**
INZPFM[1]	**Initialize Physical File Member**
OPNDBF[4]	**Open Database File**
OPNQRYF[1]	**Open Query File**
OVRDBF[5]	**Override Database File**
POSDBF	**Position Database File**

Command name	Descriptive name
RCVF	**Receive File**
RCVNETF	**Receive Network File**
RGZPFM[1]	**Reorganize Physical File Member**
RMVM[1]	**Remove Member**
RNMM[1]	**Rename Member**
SNDNETF	**Send Network File**

[1] The server system must be an IBM i or a System/38.

[2] For other DDM-related considerations about this command, see Copy commands with DDM.

[3] These commands display remote file information if the SYSTEM parameter specifies *RMT or *ALL.

[4] For information about commitment control, see Commitment control support for DDM.

[5] This command does not access the remote file.

[6] The server system must be an IBM i.

The **Submit Remote Command (SBMRMTCMD)** command can also be used to submit some of the commands to a server system.

The **Send Network File (SNDNETF)** and **Receive Network File (RCVNETF)** commands, whenever possible, should run on the server on which the data exists, rather than using a DDM file to access the remote file.

Commands not supporting DDM
These CL commands are not supported for DDM files. However, useful results for some of them might be produced on a target IBM i or a System/38 using DDM if they are submitted on the **Submit Remote Command (SBMRMTCMD)** command to run on the server system.

Command name	Descriptive name
DSNFMT	**Design Format**
DSPCHT	**Display Chart**
DSPDBR	**Display Database Relations**
DSPRCDLCK	**Display Record Locks**
MNGDEVTBL	**Manage Device Table**
MNGPGMTBL	**Manage Program Table**
MNGUSRTBL	**Manage User Table**
RTVQRYSRC	**Retrieve Query Source**
SBMFNCJOB	**Submit Finance Job**

Source file commands
If the server system is an IBM i or a System/38, these CL commands can support a DDM file as a source file (on the SRCFILE parameter).

If the server system is not an IBM i or a System/38, a DDM file name should not be specified on the SRCFILE parameter, because the remote file is neither an IBM i nor a System/38 source file.

These commands can also be affected by file overrides (by using the Override with Database File [OVRDBF] command).

Note: These commands cannot run on the client system to create a file on any server system; they can, however, be submitted to run on the target server using the Submit Remote Command (SBMRMTCMD) command.

Command name	**Descriptive name**
CRTBASPGM	**Create BASIC Program**
CRTBSCF[1]	**Create BSC File**
CRTCBLPGM	**Create COBOL Program**
CRTCLPGM	**Create CL Program**
CRTCMD	**Create Command**
CRTCMNF[1]	**Create Communications File**
CRTCPGM	**Create C Program**
CRTDSPF	**Create Display File**
CRTICFF	**Create Intersystem Communications Function File**[1]
CRTMXDF[2]	**Create Mixed File**
CRTPLIPGM	**Create PL/I Program**
CRTPRTF	**Create Printer File**
CRTPRTIMG[2]	**Create Print Image**
CRTRPGPGM	**Create RPG Program**
CRTRPTPGM	**Create Auto Report Program**
CRTTBL	**Create Table**
FMTDTA	**Format Data**
STRBAS	**Start BASIC**
STRBASPRC	**Start BASIC Procedure**

1

CRTICFF is valid on an IBM i. **CRTCMNF**, **CRTBSCF**, and **CRTMXDF** commands are valid either on System/38 or System/38 environment on an IBM i.

2

If used with the **SBMRMTCMD** command, the target must be a System/38.

DDM-related CL command summary charts
This topic shows summary charts containing most of the control language (CL) commands used with DDM.

Use these commands to determine the DDM job environment to perform remote file processing (by specifying a DDM file name on a file-related parameter of a CL command), or to perform other actions on a remote server by submitting a CL command to the server system on the **Submit Remote Command (SBMRMTCMD)** command.

The charts show which commands:

- Are file-related (that operate on file objects)
- Are object-related (that operate on objects other than files, in addition to file objects)
- Can be performed on the source side or on the target side
- Can be affected by file overrides by using the **Override with Database File (OVRDBF)** command
- Are allowed, and have a useful purpose, to be submitted to a target IBM i to run (by using the **SBMRMTCMD** command), rather than running on the client system

Notes are included in the charts that can be helpful to the DDM user.

The following items describe the kinds of information provided in these charts:

- The first column lists all the CL commands that can be used by DDM: (a) to operate on a remote file identified in a DDM file, or (b) to be submitted on a **SBMRMTCMD** command using a DDM file.
- In the second column, an F means the command is file related, an O means it is related to IBM i objects other than files, and a blank means neither of these.
- In the third column, an S means the command operates on objects on the source side, and a T means it operates on objects on the target side. For example, with the create commands that create a file or program using a DDM file as a source file, the T indicates that a source file on the server system is used for the creation; the command runs on the client system and creates a file or program on the client system, but uses a source file on the target server to do it.

  If neither S nor T is shown, the name of a DDM file should *not* be specified on the command; the command should not run on the *source* server as a DDM function. However, the command might be useful when submitted on the **SBMRMTCMD** command to run on the *target* server (see the last column).
- In the last two columns, an X indicates that the command is valid and useful when used with the command indicated at the top (**OVRDBF** or **SBMRMTCMD**) of the column. A blank indicates that the command is not valid.

Generally, when the server system is an IBM i or a System/38, any CL command that can be used in either a batch job or batch program can be specified on the **SBMRMTCMD** command. If a command has a value of *BPGM *and* *EXEC specified for the ALLOW attribute, which you can display by using the **Display Command (DSPCMD)** command, that command *can* be submitted by the **SBMRMTCMD** command. (The **SBMRMTCMD** command uses the QCAEXEC server program to run the submitted commands on the target server.)

Notes:

1. The **SBMRMTCMD** command can be used to send commands to an IBM i, System/38, or any other server system that supports the submit remote command function. The command submitted must be in the syntax of the server system.
2. Although most of the commands listed in this chart can be submitted to a remote server with the **SBMRMTCMD** command, several can just as easily be run on the client system specifying a DDM file name.
3. IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
4. All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
5. All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Table 12. DDM-related CL commands

Command name	Related to file, object or both	Affects objects on source, target or both	OVRDBF command	SBMRMTCMD command[1]
ADDLFM	F	T[2]		X
ADDPFM	F	T[3]		X
ALCOBJ	F O	S T		X
CHGDFUDEF		T		X
CHGDTA		T		
CHGJOB				X
CHGLF	F	S T		X
CHGLFM	F	T[3]		X
CHGNETA				X
CHGOBJOWN	F O	S		X
CHGPF	F	S T		X
CHGPFM	F	T[3]		X
CHGQRYDEF		T		
CHGSRCPF	F	S T		X
CHKOBJ	F O	S		X
CLOF	F	T	X	X
CLRPFM	F	T		X
COMMIT	F	S T		X[11]
CPYF	F	S T	X	X
CPYFRMDKT	F	S T	X	X[4]
CPYFRMQRYF	F	S T	X	X
CPYFRMTAP	F	S T	X	X[4]
CPYSPLF	F	T		X
CPYSRCF	F	S T	X	X
CPYTODKT	F	S T	X	X[4]
CPYTOTAP	F	S T	X	X[4]
CRTBASPGM		T		X
CRTCBLPGM		T		X
CRTCLPGM		T		X
CRTCMD		T		X
CRTDFUAPP		T		X
CRTDFUDEF		T		
CRTDSPF	F	T		X
CRTDUPOBJ	O	S	X	X
CRTICFF	F	T		X
CRTLF	F	S T		X
CRTPF	F	S T	X	X
CRTPLIPGM		T		X
CRTPRTF	F	T		X
CRTPRTIMG		T		X

Table 12. DDM-related CL commands (continued)

Command name	Related to file, object or both	Affects objects on source, target or both	OVRDBF command	SBMRMTCMD command[1]
`CRTQRYAPP`		T		X
`CRTQRYDEF`		T		
`CRTRPGPGM`		T		X
`CRTRPTPGM`		T		X
`CRTSRCPF`	F	S T	X	X
`CRTTBL`		T		X
`DCLF`	F	T		
`DLCOBJ`	F O	S T		X
`DLTDFUAPP`				X
`DLTF`	F	S T		X
`DLTQRYAPP`				X
`DMPOBJ`	F O	S		X[5]
`DMPSYSOBJ`	O	S		X[5]
`DSNDFUAPP`		T		
`DSNQRYAPP`		T		
`DSPDTA`		T		
`DSPFD`	F	S T		X[5]
`DSPFFD`	F	S T		X[5]
`DSPNETA`				X
`DSPOBJAUT`	F O	S		X[5]
`DSPOBJD`	F O	S		X[5]
`DSPPFM`	F	T		
`ENDCMTCTL`	F	S T		X[11]
`FMTDTA`		T		X
`GRTOBJAUT`	F O	S		X
`INZPFM`	F	T[2]		X
`MOVOBJ`	O	S		X
`OPNDBF`[6]	F	T	X	
`OPNQRYF`	F	T	X	X
`OVRDBF`	F	S		[7]
`POSDBF`	F	T		X
`QRYDTA`		T		X
`RCVF`	F	T		
`RCVNETF`	F			X
`RGZPFM`	F	T		X
`RMVM`	F	T		X
`RNMM`	F	T		X
`RNMOBJ`	F O	S T[8]		X
`ROLLBACK`	F	S T		X[11]
`RSTLIB`		S		X[9]
`RSTOBJ`	F O	S		X[9]
`RTVDFUSRC`		T		X

Table 12. DDM-related CL commands (continued)

Command name	Related to file, object or both	Affects objects on source, target or both	OVRDBF command	**SBMRMTCMD** command[1]
RTVQRYSRC		T		X
RVKOBJAUT	F O	S		X
SAVCHGOBJ	O	S		X[9]
SAVLIB		S		X[9]
SAVOBJ	F O	S		X[9]
SBMDBJOB		T		X
SNDNETF	F	T		X
STRBAS		T		X
STRBASPRC	O	T		X
STRCMTCTL	F	S T		X[11]
STRDBRDR		T		X
WRKJOB	O			X[5]
WRKOBJLCK[10]	F O	S		X[5]

Notes:

1

The use of the **SBMRMTCMD** command is not valid with *any* of the commands in these charts unless the server system is an IBM i or a System/38.

2

This member-related command can be used only if the server system is an IBM i.

3

This member-related command can be used only if the server system is an IBM i or a System/38.

4

These commands require intervention on the server system to load a tape and they might not produce the results expected.

5

When submitted to the server system, these commands produce output on the server system only; the output is not sent to the client system.

6

OPNDBF command: For more information about commitment control restrictions, see Commitment control support for DDM.

7

OVRDBF command: Although this command works when submitted on the **SBMRMTCMD** command to a target IBM i or a System/38, it is *not* recommended.

8

RNMOBJ command: OBJTYPE*FILE must be specified.

9

When submitted to the server system, these commands require server system resources when tapes are used to produce the output.

10

WRKOBJLCK command: This command displays any locks on the DDM file, not the remote file.

11

This command will work, but its use is not recommended.

Data description specifications considerations for DDM

Data description specifications (DDS), which is used to externally describe the fields and record formats, can also be used with DDM to describe the file and record formats of a remote file.

IBM i target considerations for DDM

This topic covers target considerations for DDM.

As with any database file, DDS might or might not be used to externally describe the attributes of the remote file when it is created on the remote IBM i. If DDS is used, then the source server program uses those attributes when it accesses the remote file (by using the DDM file). If DDS is not used, then the file's attributes must be described in the program.

When a client system program that accesses a file on a target IBM i is compiled (or recompiled), the existing DDM file is used to establish communications with the target server, and the remote file is actually accessed during compilation to extract its file and record attributes. Whether DDS is used to describe the file or not, level check identifiers are created during compilation and are included in the compiled program. These values are then used when the program is run and LVLCHK(*RMTFILE) is in effect for the DDM file.

Whether DDS is used to describe a remote IBM i file or not, a client system program can have its own field and record format definitions provided in the program, or the program can substitute the definitions of another client system file that is created using DDS. Either can be done if LVLCHK(*NO) is in effect in the DDM file or specified in an **`Override with Database File (OVRDBF)`** command used at program run time. LVLCHK(*NO) need only be used when the record format used on the client system is different than that of the remote IBM i file.

Non-IBM i target considerations for DDM

DDS can be used with a non-IBM i file only if the local IBM i program is compiled using a local IBM i file that has the same record format name as the DDM file being used.

After the program is compiled, the local file can be overridden by a DDM file that accesses the remote file. LVLCHK(*NO) must be specified in the DDM file or in an OVRDBF command.

If no DDS exists on the local server to describe the remote file, the program must describe the fields. The **`Display File Field Description (DSPFFD)`** command can be used to display the field attributes of the remote file. LVLCHK(*NO) should be specified in the DDM file or in an **OVRDBF** command.

If LVLCHK(*RMTFILE) is specified or assumed, the program must be compiled (or recompiled) using a DDM file that accesses the remote file. The IBM i then creates a record format and fields for the remote file. The names of the fields that are created are of the type K*nnnnn* for keyed fields and F*nnnnn* for nonkeyed fields.

DDM-related DDS keywords and information

The information about DDS keywords that relates specifically to DDM is provided in this topic.

- Considerations for creating local files:
  - The following DDS keywords *cannot* specify the name of a DDM file: REFACCPTH, and FORMAT.
  - The DDS keywords REF and REFFLD *can* specify the names of DDM files to refer to remote files; however, the remote files must be on an IBM i or a System/38. When a DDM file name is specified as the database file name in either keyword, it refers to the DDM file on the client system, and the referred to field name and record format name refer to a field and record format used in the remote file on the server system.
- Considerations for creating logical files when the remote server is not an IBM i:
  - At least one key field must be specified in the record format for the logical file.
  - Only one file can be specified on the PFILE keyword.
  - SELECT and OMIT functions are not supported.
  - Logical join files are not supported.

  - Field names of remote physical files have the naming convention of F00001, F00002, F00003, and so forth (F*nnnnn*) for nonkeyed fields and K00001, K00002, K00003, and so forth (K*nnnnn*) for keyed fields.

    The exception to this naming convention is when the server system is a System/38 and the physical file was created locally. In this case, the field names are the same as the field names specified when the physical file was created.
  - All the fields defined for the logical file must be specified in the same order as defined in the physical file. This can be done by default.
  - The SST keyword can be used to access partial fields of the physical file. The use of two or more substring fields is required to define the entire physical field. In addition, the partial fields must be in the same order as defined in the substring field of the physical file.
  - The CONCAT keyword can be used to group physical file fields into one logical field. The concatenation order of the fields must be in the same order as defined in the physical file.
  - The fields of the physical file must be specified in the same order as defined in the physical file.
- Considerations for using the VARLEN DDS keyword when creating files on a non-IBM i server system:
  - Target server must support variable-length record files.
  - Only one variable-length field is allowed in the file format and it must be the last field.
  - The field with the VARLEN keyword must not be a key field.
- PFILE and JFILE considerations for creating remote files:
  - The record format name specified for the physical file in the DDM file on the JFILE or PFILE keyword must be the same name as the DDM file that represents the remote physical file.
  - When creating a logical file, the file specified on PFILE or JFILE must be a DDM file, and the location for each physical file in the DDM file on the JFILE or PFILE keyword must be the same as the location of the DDM file for the logical file. In other words, the physical files and logical file must be on the same remote server.

  If the remote server is a release 1.0 or 1.2 IBM i, attempting to create a file using the FCFO keyword will fail.
- When the server is not an IBM i, these keywords are either ignored or not supported for *logical* files:

ABSVAL
ACCPTH
ALIAS
ALL
ALTSEQ
CHECK
CMP
COLHDG
COMP
DIGIT
DYNSLT

EDTCDE
EDTWRD
FCFO
FLTPCN
FORMAT
JDFTVAL
JDUPSEQ
JFILE
JFLD
JOIN
JREF

LIFO
NOALTSEQ
RANGE
REFSHIFT
RENAME
SIGNED
TEXT
TRNTBL
VALUES
ZONE

- When the server is not an IBM i, these keywords are either ignored or not supported for *physical* files:

ABSVAL
ALTSEQ
CHECK
CMP
COLHDG
COMP
DIGIT

EDTCDE
EDTWRD
FCFO
FLTPCN
FORMAT
LIFO
NOALTSEQ

RANGE
REFSHIFT
SIGNED
TEXT
VALUES
ZONE

DDM user profile authority

IBM i users are not allowed to perform functions equivalent to CL commands on remote IBM is using DDM without the proper command authorization.

The user profiles associated with the target jobs must have *OBJOPR authority to the following CL commands to start the equivalent operation on the remote IBM i:

Command name	Descriptive name
ADDLFM	**Add Logical File Member**
ADDPFM	**Add Physical File Member**
ALCOBJ	**Allocate Object**
CHGLF	**Change Logical File**
CHGLFM	**Change Logical File Member**
CHGPF	**Change Physical File**
CHGPFM	**Change Physical File Member**
CRTLF	**Create Logical File**
CRTPF	**Create Physical File**
DLTF	**Delete File**
INZPFM	**Initialize Physical File Member**
RGZPFM	**Reorganize Physical File Member**
RMVM	**Remove Member**
RNMM	**Rename Member**
RNMOBJ	**Rename Object**

DDM commands and parameters

This topic classifies DDM commands and parameters.

For additional information about DDM subsets, see the *DDM Architecture: Implementation Planner's Guide* or the *DDM Architecture: Reference*.

Note: The abbreviation *KB* appears throughout the tables in these topics. It represents a quantity of storage equal to 1024 bytes.

Subsets of DDM architecture supported by IBM i DDM

The IBM i supports these subsets of the DDM architecture.

Supported DDM file models

IBM i DDM supports these DDM file models.

- Alternate index file (ALTINDF)
- Direct file (DIRFIL)
- Directory file (DIRECTORY)
- Keyed file (KEYFIL)
- Sequential file (SEQFIL)
- Stream file (STRFIL)

By using the above file models, the IBM i supports access to the IBM i physical and logical files. The following table shows how DDM file models and IBM i data files correspond.

Table 13. IBM i data files

DDM file model	Corresponding IBM i data file
Alternate index file (ALTINDF)	Logical file with one format
Direct file (DIRFIL)	Nonkeyed physical file
Directory file (DIRECTORY)	Folder management services (FMS) folders or data management libraries
Keyed file (KEYFIL)	Keyed physical file
Sequential file (SEQFIL)	Nonkeyed physical file
Stream file (STRFIL)	Folder management services (FMS) document

Alternate Index File (ALTINDF)
IBM i DDM supports access to a logical file by using the DDM alternate index file model.

A logical file allows access to the data records stored in a physical file by using an alternate index defined over the physical file. Only single format logical files can be accessed through IBM i DDM. Logical files with select/omit logic can be accessed, but records that are inserted might not be retrievable if they are omitted by the select/omit logic.

Supported record classes

An IBM i alternate index file can have fixed-length record (RECFIX) or variable-length record (RECVAR) for storage.

Once a non-IBM i source server opens a file on the IBM i target using variable-length record access, the IBM i target continues to send and receive variable-length records on all subsequent I/O operations.

Note: IBM i DDM supports the DDM file transfer commands **`Load Record File (LODRECFIL)`** and **`Unload Record File (ULDRECFIL)`** for all of the file models except alternate index file.

Direct file (DIRFIL)
IBM i DDM supports access to nonkeyed physical files via the DDM direct file model.

The support has the following characteristics:

- Delete Capabilities: An IBM i direct file is delete capable or nondelete capable. A nondelete capable file must have an active default record.
- Supported Record Classes: An IBM i direct file can have a fixed-length record (RECFIX) or variable-length record (RECVAR) for storage. Once a non-IBM i source server opens a file on the IBM i target using variable-length record access, the IBM i target continues to send and receive variable-length records on all subsequent I/O operations.

  Note: The IBM i does not support the concept of a direct file. IBM i DDM creates a direct file by creating a nonkeyed physical file and initializing it, with deleted or active default records, to the maximum size requested. No extensions to the file are allowed.

Directory file (DIRECTORY)
IBM i DDM supports access to a folder management services folder or a data management library via the DDM directory file model. Folders can be created, opened, renamed, closed, or deleted. Libraries can be created, renamed, or deleted.

Keyed file (KEYFIL)
IBM i DDM supports access to keyed physical files via the DDM keyed file model.

An IBM i keyed file can have fixed-length record (RECFIX) or variable-length record (RECVAR) for storage. Once a non-IBM i source server opens a file on the IBM i target using variable-length record access, the IBM i target continues to send and receive variable-length records on all subsequent I/O operations.

Sequential file (SEQFIL)
The IBM i supports access to nonkeyed physical files via the DDM sequential file model.

The support has the following characteristics:

- The sequential file can be delete or nondelete capable on an IBM i.
- The sequential file on an IBM i can have a fixed-length record (RECFIX) or variable-length record (RECVAR) for storage. Once a non-IBM i source server opens a file on the IBM i target using variable-length record access, the IBM i target continues to send and receive variable-length records on all subsequent I/O operations.

Stream file (STRFIL)
IBM i DDM supports access to a folder management services document by means of the DDM stream file model.

Supported DDM access methods
IBM i DDM supports these DDM access methods. DDM abbreviations for the access methods are given in parentheses.

- Combined access method (CMBACCAM)
- Combined keyed access method (CMBKEYAM)
- Combined record number access method (CMBRNBAM)
- Directory access method (DRCAM)
- Random by key access method (RNDKEYAM)
- Random by record number access method (RNDRNBAM)
- Relative by key access method (RELKEYAM)
- Relative by record number access method (RELRNBAM)
- Stream access method (STRAM)

See the following table for a summary of the access methods that IBM i DDM supports for each DDM file model. For a description of these access methods, refer to the *DDM Architecture: Implementation Planner's Guide, GC21-9528*.

Table 14. Supported access methods for each DDM file model

		DDM file models					
Term	**Access method**	**Sequential file**	**Direct file**	**Keyed file**	**Alternate index file**	**Stream file**	**Directory file**
CMBACCAM	Combined access	N	T	T	N		
CMBKEYAM	Combined keyed			T	T		
CMBRNBAM	Combined record number	T	T	T	N		
DRCAM	Directory						T
RELKEYAM	Relative by key			T	T		
RELRNBAM	Relative by record number	T	T	T	N		
RNDKEYAM	Random by key			T	T		
RNDRNBAM	Random by record number	T	T	T	N		
STRAM	Stream					T	

Table 14. Supported access methods for each DDM file model (continued)

Term	Access method	DDM file models					
		Sequential file	Direct file	Keyed file	Alternate index file	Stream file	Directory file

Note:

N
= Not supported

T
= Target DDM supported

Blank
= Not applicable

DDM commands and objects

These topics describe the DDM command parameters that an IBM i supports for each DDM architecture command.

For more detailed information about these parameters, see the *DDM Architecture: Reference*, SC21-9526.

The description of the commands might include:

- Limitations for the use of each command
- Objects that the client system might send to the server system
- Objects that the server system might return to the client system
- DDM parameters that the IBM i supports for the command and how the IBM i responds to each parameter

The commands listed here are supported. Level 1.0, Level 2.0 and Level 3.0 indicate which level of the DDM architecture is supported by the commands.

CHGCD (Change Current Directory) *Level 2.0*

This command changes the current path. The path is a string of folders. The current path is added to the front of a file or directory name if it does not start with a slash.

This command is not sent by a source IBM i.

Parameter name	Source	Target
AGNNAM	N/A	Ignored
DRCNAM[1]	N/A	IBM i name

[1] Name formats are server defined. The architecture specifies that a directory name length of zero indicates the root directory for the **Change Current Directory** command. For other commands, a directory name length of zero indicates the *current* directory, which might or might not be the root directory at the time the command is issued.

CHGEOF (Change End of File) *Level 2.0 and Level 3.0*

This command changes the end-of-file mark of a document. The end may be truncated or expanded.

A source IBM i does not send this command.

Parameter name	Source	Target
DCLNAM	N/A	Program defined
EOFNBR	N/A	Supported
EOFOFF	N/A	Supported

CHGFAT (Change File Attribute) *Level 2.0*
This command changes the attributes of a file, document, or folder.

Parameter name	Stream file	Directory	Sequential, direct, and keyed files	Alternate index file
DTAFMT	T			
FILCHGDT	T	T	N	N
FILCHGFL	T	N	N	
FILINISZ	N		S, T	
FILEXNSZ	N		S, T	
FILEXPDT			S, T	
FILHDD	T	T	N	N
FILMAXEX	N		S, T	
FILPRT	T	N		
FILSYS	T	T	N	N
DELCP			N	N
GETCP	T		N	
INSCP			N	
MODCP	T		N	
TITLE	T	T	S, T	S, T

N = Not supported
T = Target DDM supported
S = Source DDM supported
Blank = Not applicable				

CLOSE (Close File) *Level 1.0 and Level 2.0*
This command ends the logical connection between the client system and the data set accessed on the server system. When the target DDM begins running this command, it must close the data set regardless of the reply message returned.

Parameter name	Source	Target
DCLNAM	Program defined	Program defined
SHDPRC	Not sent	Supported
Note: Names are implementation defined.		

CLRFIL (Clear File) *Level 1.0 and Level 2.0*
This command clears an existing file and re-initializes it as if it had just been created.

Parameter name	Source	Target
FILNAM	Target defined	IBM i
OVRDTA	Not sent	False only
Name formats are server defined.		

CLSDRC (Close Directory) *Level 2.0*

This command closes a folder. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM	N/A	Program defined
Names are implementation defined.		

CPYFIL (Copy File) *Level 2.0*

This command copies one document to another document. If the new document does not exist, it might be created. This command is not sent by a source IBM i.

Parameter name	Source	Target
ACCORD	N/A	Ignored
BYPDMG	N/A	Ignored
BYPINA	N/A	Ignored
CPYNEW[1]	N/A	Supported
CPYOLD[2]	N/A	Supported
DCLNAM[3]	N/A	Program defined
FILNAM[4]	N/A	IBM i name
NEWFILNM[4]	N/A	IBM i name

1. CPYNDT only supported parameter value. All others are rejected with VALNSPRM.
2. CPYERR only supported parameter value. All others are rejected with VALNSPRM.
3. Names are implementation defined.
4. Name formats are server defined.

CRTAIF (Create Alternate Index File) *Level 1.0 and Level 2.0*

This command creates an alternate index file on the server system.

Parameter name	Source	Target
BASFILNM[1]	Program defined	IBM i name
DUPFILOP	Not sent	Supported
FILCLS[2]	Not sent	Ignored
FILHDD	Not sent	Ignored
FILNAM[3]	Program defined	IBM i name
FILSYS	Not sent	Ignored
KEYDEF[4]	Sent	Supported
KEYDUPCP	Sent	Supported
RTNCLS[5]	Not sent	Supported
TITLE	Sent	Supported

Parameter name	Source	Target
1 Name formats are server defined.		
2 Only ALTINDF is valid for CRTAIF command.		
3 Name formats are server defined.		
4 IBM i maximum key length is 2000.		
5 Library QTEMP is used for temporaries.		

`CRTDIRF (Create Direct File)` *Level 1.0 and Level 2.0*
This command creates a direct file on the server system.

Parameter name	Source	Target
ALCINIEX	Sent	Ignored
DCLNAM[1]	Not sent	Supported
DELCP[2]	Sent	Supported
DFTREC	Sent	Supported
DFTRECOP	Sent	Supported
DUPFILOP	Not sent	Supported
FILCLS[3]	Not sent	Ignored
FILEXNSZ[4]	Sent	Supported
FILEXPDT[5]	Sent	Supported
FILHDD	Not sent	Ignored
FILINISZ[4]	Sent	Supported
FILMAXEX[6]	Sent	Supported
FILNAM[7]	Program defined	IBM i name
FILSYS	Not sent	Ignored
GETCP	Sent	Supported
INSCP[8]	Sent	Supported
MODCP	Sent	Supported
RECLEN[9]	Sent	Supported
RECLENCL	Sent	Supported
:row.	RTNCLS[10]	Not sent
Supported		
TITLE	Sent	Supported

Parameter name	Source	Target

[1] Names are implementation defined.

[2] Value must be TRUE unless DFTRECOP (DFTSRCIN) is specified.

[3] Only DIRFIL is valid for **CRTDIRF** command.

[4] IBM i default is 1,000 records.

[5] IBM i default is *NONE.

[6] IBM i default is 3.

[7] Name formats are server defined.

[8] Only TRUE is valid.

[9] IBM i maximum record length = 2**15-2. (2 to the power of 15 minus 2)

[10] Library QTEMP is used for temporaries.

CRTDRC (Create Directory) *Level 2.0*
This command creates folders or libraries on the server system, based on the name received. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
DRCNAM[2]	N/A	IBM i name
FILCLS[3]	N/A	Ignored
FILPRT[4]	N/A	Supported
RTNCLS	N/A	PRMFIL only
TITLE	N/A	Supported

[1] Names are implementation defined.

[2] Name formats are server defined.

[3] Only DIRECTORY is valid for **CRTDRC** command.

[4] FALSE only for libraries.

CRTKEYF (Create Keyed File) *Level 1.0 and Level 2.0*
This command creates a keyed file on the server system.

Parameter name	Source	Target
ALCINIEX	Sent	Ignored

Parameter name	Source	Target
DCLNAM[1]	Not used	Supported
DELCP	Sent	Supported
DFTREC	Not sent	Supported
DFTRECOP	Not sent	Supported
DUPFILOP	Not sent	Supported
FILCLS[2]	Not sent	Ignored
FILEXNSZ[3]	Sent	Supported
FILEXPDT[4]	Sent	Supported
FILHDD	Not sent	Ignored
FILINISZ[3]	Sent	Supported
FILMAXEX[5]	Sent	Supported
FILNAM[6]	Program defined	IBM i name
FILSYS	Not sent	Ignored
GETCP	Sent	Supported
INSCP	Sent	Supported
KEYDEF[7]	Sent	Supported
KEYDUPCP	Sent	Supported
MODCP	Sent	Supported
RECLEN[8]	Sent	Supported
RECLENCL	Sent	Supported
RTNCLS[9]	Not sent	Supported
TITLE	Sent	Supported

[1] Names are implementation defined.

[2] Only KEYFIL is valid for **CRTKEYF** command.

[3] IBM i default is 1,000 records.

[4] IBM i default is *NONE.

[5] IBM i default is 3.

[6] Name formats are server defined.

[7] IBM i maximum key length is 2000.

[8] IBM i maximum record length = 2**15-2. (2 to the power of 15 minus 2)

[9] Library QTEMP is used for temporaries.

Note: When a CRTKEYF request is received by an IBM i target server, the new keyed file reuses deleted records when it is created. If duplicate keys are allowed (KEYDUPCP=TRUE sent), the order of the duplicate keys is not guaranteed.

CRTSEQF (Create Sequential File) *Level 1.0 and Level 2.0*
This command creates a sequential file on the server system.

Parameter name	Source	Target
ALCINIEX	Sent	Ignored
DCLNAM[1]	Not sent	Supported
DELCP	Sent	Supported
DFTREC	Not sent	Supported
DFTRECOP	Not sent	Supported
DUPFILOP	Not sent	Supported
FILCLS[2]	Not sent	Ignored
FILEXNSZ[3]	Sent	Supported
FILEXPDT[4]	Sent	Supported
FILHDD	Not sent	Ignored
FILINISZ[3]	Sent	Supported
FILMAXEX[5]	Sent	Supported
FILNAM[6]	Program defined	IBM i name
FILSYS	Not sent	Ignored
GETCP	Sent	Supported
INSCP	Sent	Supported
MODCP	Sent	Supported
RECLEN[7]	Sent	Supported
RECLENCL	Sent	Supported
RTNCLS[8]	Not sent	Supported
TITLE	Sent	Supported

Parameter name	Source	Target

1

Names are implementation defined.

2

Only SEQFIL is valid for **CRTSEQF** command.

3

IBM i default is 1,000 records.

4

IBM i default is *NONE.

5

IBM i default is 3.

6

Name formats are server defined.

7

IBM i maximum record length = 2**15-2. (2 to the power of 15 minus 2.)

8

Library QTEMP is used for temporaries.

CRTSTRF (Create Stream File) *Level 2.0*

This command creates a stream file on the server system. This command is not sent by a source IBM i.

Parameter name	Source	Target
ALCINIEX	N/A	Ignored
DCLNAM[1]	N/A	Program defined
DTAFMT	N/A	Supported
DUPFILOP	N/A	Supported
FILCLS[2]	N/A	Ignored
FILEXNSZ	N/A	Ignored
FILEXPDT	N/A	Ignored
FILHDD	N/A	Supported
FILINISZ	N/A	Ignored
FILMAXEX	N/A	Ignored
FILNAM[3]	N/A	IBM i name
FILPRT	N/A	Supported
FILSYS	N/A	Supported
GETCP	N/A	Supported
MODCP	N/A	Supported
RTNCLS	N/A	Supported
TITLE	N/A	Supported

Parameter name	Source	Target

1

Names are implementation defined.

2

Only STRFIL is valid for **CRTSTRF** command.

3

Name formats are server defined.

DCLFIL (Declare File) *Level 1.0 and Level 2.0*

This command associates a declared name (DCLNAM) with a collection of object-oriented parameters in the target agent. This collection is stored by the receiving agent for later use. At the time it is received, the command does not affect objects currently opened by the agent. The primary access to the DCLFIL collection is the DCLNAM parameter.

Parameter name	Source	Target
AGNNAM[1]	Not sent	Ignored
DCLNAM[2]	Program defined	Program defined
DRCNAM[3]	Not sent	IBM i name
FILEXNSZ[4]	Not sent	Ignored
FILMAXEX[4]	Not sent	Ignored
FILNAM[3]	Program defined	IBM i name

1

Only one agent on an IBM i.

2

Names are implementation defined.

3

Name formats are server defined.

4

Create value is used.

DELDCL (Delete Declared Name) *Level 1.0*

This command deletes a declared agent name.

Parameter name	Source	Target
AGNNAM	Not sent	Ignored
DCLNAM[1]	Program defined	Program defined

[1] Names are implementation defined.

DELDRC (Delete Directory) *Level 2.0*

This command deletes a folder or a library. This command is not sent by a source IBM i.

Parameter name	Source	Target
DELDRCOP[1]	N/A	DRCEMP or DRCANY
DRCNAM[2]	N/A	IBM i name
OVRDTA	N/A	FALSE only

Parameter name	Source	Target

1

DRCALL not supported.

2

Name formats are server defined. Generic names are not supported.

DELFIL (Delete File) *Level 1.0 and Level 2.0*

This command deletes a file or document.

Parameter name	Source	Target
FILNAM[1]	Target defined generics allowed	IBM i name
OVRDTA[2]	Not sent	FALSE only
SHDONL[3]	Not sent	Supported

1

Name formats are server defined. Generic names are only allowed for documents.

2

The IBM i does not support overwriting.

3

FALSE only for files.

DELREC (Delete Record) *Level 1.0*

This command deletes the record that currently has an update intent placed on it. It does this without affecting the current cursor position.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined

[1]Names are implementation defined.

EXCSAT (Exchange Server Attributes) *Level 1.0 and Level 2.0*

This command exchanges information between servers, such as the server's class name, architectural level of each class of managers it supports, server's product release level, server's external name, and server's name.

Parameter name	Source	Target
EXTNAM	Sent	Supported
MGRLVLS	Sent	Supported
SPVNAM	Not sent	Ignored
SRVCLSNM	Sent	Supported
SRVNAM	Sent	Supported
SRVRLSLV	Sent	Supported

The following reply object is returned:

EXCSATRD

Server attributes reply data

FILAL *and* ***FILATTRL (File Attribute List)*** *Level 1.0, Level 2.0, and Level 3.0*
This is a list of file attributes that DDM might request on an LSTFAT, OPEN, or GETDRCEN. Some parameters are only valid for specific file types.

Table 15. File attribute list

Parameter name	Source	Target
ACCMTHLS	Requested	Supported
BASFILNM[1]	Requested	IBM i name
DELCP	Requested	Supported
DFTREC	Requested	Supported
DTAFMT	Not requested	Supported
EOFNBR	Requested	Supported
EOFOFF	Not requested	Supported
FILBYTCN	Not requested	Supported
FILCHGDT	Requested	Supported
FILCHGFL	Not requested	Supported
FILCLS	Requested	Supported
FILCRTDT	Requested	Supported
FILEXNCN	Requested	Supported
FILEXNSZ	Requested	Supported
FILEXPDT	Requested	Supported
FILHDD	Not requested	Supported
FILINISZ	Requested	Supported
FILMAXEX	Requested	Supported
FILNAM	Requested	Supported
FILPRT	Not requested	Supported
FILSIZ	Requested	Supported
FILSYS	Not requested	Supported
GETCP	Requested	Supported
INSCP	Requested	Supported
KEYDEF	Requested	Supported
KEYDUPCP	Requested	Supported
LSTACCDT	Not requested	Not supported
LSTARCDT	Requested	Supported
MAXARNB	Requested	Not supported
MODCP	Requested	Supported
RECLEN	Requested	Supported
RECLENCL	Requested	Supported
RTNCLS[2]	Not requested	PRMFIL
SHDEXS	Not requested	Supported

Table 15. File attribute list (continued)

Parameter name	Source	Target
STRSIZ	Not requested	Supported
TITLE[3]	Requested	Supported

[1] Name formats are server defined. Qualified name if FILCLS is ALTINDF.

[2] Unless the library is QTEMP.

[3] Maximum length of text is 50 characters for data file, 44 for document or folder.

FRCBFF (Force Buffer) *Level 2.0*

This command forces the data of the referred object to nonvolatile storage.

Parameter name	Source	Target
DCLNAM[1]	Requested	Program defined

[1] Names are implementation defined.

GETDRCEN (Get Directory Entries) *Level 2.0*

This command gets a list of folders, documents or both. This command is not sent by a source IBM i.

Parameter name	Source	Target
BGNNAM[1]	N/A	IBM i name
DCLNAM[2]	N/A	Program defined
FILATTRL	N/A	Supported
FILCLS	N/A	DIRECTORY or STRFIL only
FILHDD	N/A	Supported
FILSYS	N/A	Supported
MAXGETCN	N/A	Supported
NAME[1]	N/A	IBM i name

[1] Name formats are server defined.

[2] Names are implementation defined.

The following reply object is possible:

FILAL
File attribute list

GETREC (Get Record at Cursor) *Level 1.0*

This command gets and returns the record indicated by the current cursor position.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported

Parameter name	Source	Target
RTNINA[2]	As required	Supported
UPDINT	Not sent	Supported

[1] Names are implementation defined.

[2] Application dependent.

The following reply objects are possible:

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum =2**15-2) (2 to the power of 15 minus 2.)

RECORD
Fixed length record (maximum length 2**15-2)

GETSTR (Get Substream) *Level 2.0 and Level 3.0*
This command gets stream data from a document. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
STRLEN	N/A	Supported
STROFF	N/A	Supported
STRPOS	N/A	Supported

[1] Names are implementation defined.

INSRECEF (Insert at EOF) *Level 1.0*
This command inserts a record at the end of the file.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
KEYVALFB	Requested	Supported
RECCNT[2]	As required	Supported
RECNBRFB	Requested	Supported
RLSUPD	Always FALSE	Supported
UPDCSR	Not sent	Supported

[1] Names are implementation defined.

[2] Application dependent.

The following command objects are possible:

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECORD
Fixed length record (maximum length 2**15-2)

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

`INSRECKY (Insert Record by Key Value)` *Level 1.0*
This command inserts one or more records according to their key values wherever there is space available in the file.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
RECCNT	As required	Supported
RECNBRFB	Requested	Supported
RLSUPD	Always FALSE	Supported
UPDCSR	Not sent	Supported
[1]Names are implementation defined.		

The following command object is possible:

RECORD
Fixed length record (maximum length 2**15-2) (2 to the power of 15 minus 2)

Because the IBM i does not support variable length records, only the following reply object is possible:

RECNBR
Record number

`INSRECNB (Insert Record at Number)` *Level 1.0*
This command inserts one or more records at the position specified by the record number parameter.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
KEYVALFB	Requested	Supported
RECCNT	As required	Supported
RECNBR	Sent	Supported
UPDCSR	Not sent	Supported
[1]Names are implementation defined.		

The following command objects are possible:

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECORD
Fixed length record (maximum length 2**15-2)

The following reply object is possible:

KEYVAL
Key value

LCKFIL (Lock File) *Level 1.0 and Level 2.0*
This command locks the file for subsequent use by the requester.

Parameter name	Source	Target
FILNAM[1]	Target name	IBM i name
LCKMGRNM	Not used	Ignored
RQSFILLK	Sent	Supported
WAIT	Sent	Supported
[1]Name formats are server defined.		

LCKSTR (Lock Substream) *Level 2.0 and Level 3.0*
This command locks a stream file substream. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
RQSSTRLK	N/A	EXCSTRLK and SHRSTRLK only
STRLOC	N/A	Supported
STROFF	N/A	Supported
WAIT[2]	N/A	Supported
1 Names are implementation defined. **2** The WAIT parameter is neither rejected nor performed.		

LODRECF (Load Record File) *Level 1.0 and Level 2.0*
This command puts a whole record file on the server system.

Parameter name	Source	Target
FILNAM[1]	Sent	IBM i name
[1]Name formats are server defined.		

The following command objects are possible:

RECAL
Record attribute list

RECCNT
Record count

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECORD
Fixed length record (maximum length 2**15-2)

LODSTRF (Load Stream File) *Level 2.0*
This command sends a whole stream file from the client system to the server system. This command is sent by a source IBM i when using the copy stream file HPS API.

Parameter name	Source	Target
FILNAM[1]	Sent	IBM i name
[1]Name formats are server defined.		

The following command objects are possible:

STREAM
Stream

STRSIZ
Stream size

LSTFAT (List File Attributes) *Level 1.0, Level 2.0, and Level 3.0*
This command retrieves selected attributes of a file, document, or folder.

Parameter name	Source	Target
FILATTRL	Sent	Supported
FILNAM[1]	Target name	IBM i name
DCLNAM[2]	Not sent	Supported

[1] Name formats are server defined.

[2] Names are implementation defined.

The following reply object is possible:

FILAL
List file attributes reply data

MODREC (Modify Record with Update Intent) *Level 1.0*
This command changes the record that currently has update intent placed on it without affecting the current cursor position.

Parameter name	Source	Target
ALWMODKY	Sent	Supported
DCLNAM[1]	Sent	Program defined

[1] Names are implementation defined.

The following command object is possible:

RECORD
Fixed length record (maximum length 2**15-2) (2 to the power of 15 minus 2)

OPEN (Open File) *Level 1.0 and Level 2.0*
This command establishes a logical connection between the using program on the client system and the object on the server system.

Parameter name	Source	Target
ACCINTLS	Sent	Supported
ACCMTHCL	Sent	Supported
DCLNAM[1]	Program defined	Program defined
FILATTRL	Not sent	Supported
FILSHR	Sent	Supported
PRPSHD	Not sent	Supported for stream files only

[1] Names are implementation defined.

OPNDRC (Open Directory) *Level 2.0*
This command opens a folder on the server system. This command is not sent by a source IBM i.

Parameter name	Source	Target
ACCMTHCL	N/A	DRCAM only
DCLNAM[1]	N/A	Program defined
[1]Names are implementation defined.		

PUTSTR (Put Substream) *Level 2.0 and Level 3.0*
This command puts stream data into a document. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
STROFF	N/A	Supported
STRPOS	N/A	Supported
[1]Names are implementation defined.		

The following command object is possible:

STREAM
Stream

QRYCD (Query Current Directory) *Level 2.0*
This command returns the current directory. This command is not sent by a source IBM i.

Parameter name	Source	Target
AGNNAM	N/A	Ignored

The following reply object is possible:

DRCNAM
Directory name

Note: A directory name length of zero indicates that the root directory is the *current* directory.

QRYSPC (Query Space) *Level 2.0*
This command returns the amount of space available to a user. This command is not sent by a source IBM i.

Parameter name	Source	Target
AGNNAM	N/A	Ignored

The following reply object is possible:

QRYSPCRD
Query space reply data

RNMDRC (Rename Directory) *Level 2.0*
This command renames a folder or database library. The command does not support moving folders, and is not sent by a source IBM i.

Parameter name	Source	Target
DRCNAM	N/A	IBM i name
NEWDRCNM	N/A	IBM i name

Parameter name	Source	Target
Note: Name formats are server defined. Generic names are not allowed.		

`RNMFIL (Rename File)` *Level 1.0 and Level 2.0*
This command changes the name of an existing database file or document and can also be used for moving documents.

Parameter name	Source	Target
FILNAM[1]	Sent	IBM i name
NEWFILNM[2]	Sent	IBM i name
1 Name formats are server defined. Generic names are allowed for documents only. **2** Name formats are server defined.		

`SBMSYSCMD (Submit server Command)` *Level 4.0*
This command submits a server command, in the target control language syntax, to the server system.

Parameter name	Source	Target
SYSCMD[1]	Sent	Supported
[1]Command string to be run.		

`SETBOF (Set Cursor to Beginning of File)` *Level 1.0*
This command sets the cursor to the beginning-of-file position of the file.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
[1]Names are implementation defined.		

`SETEOF (Set Cursor to End of File)` *Level 1.0*
This command sets the cursor to the end-of-file position of the file.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
[1]Names are implementation defined.		

`SETFRS (Set Cursor to First Record)` *Level 1.0*
This command sets the cursor to the first record of the file.

Parameter name	Source	Target
BYPINA[1]	As required	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

Parameter name	Source	Target

1
Application dependent.

2
Names are implementation defined.

3
IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECNBR
Record number

RECORD
Record

SETKEY (Set Cursor by Key) *Level 1.0*
This command positions the cursor based on the key value supplied and the relational operator specified for RELOPR.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVAL[2]	Max = 2000	Max = 2000
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RELOPR	Sent	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

1
Names are implementation defined.

2
Maximum key size allowed by an IBM i.

3
IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETKEYFR (Set Cursor to First Record in Key Sequence) *Level 1.0*
This command sets the cursor to the first record in the key sequence.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNREC[2]	Sent	Supported
UPDINT[2]	Sent	Supported

[1] Names are implementation defined.

[2] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETKEYLM (Set Key Limits) *Level 1.0*
This command sets the limits of the key values for subsequent **SETKEYNX** and **SETNXTKE** commands. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
KEYHLM[2]	N/A	Supported
KEYLLM[2]	N/A	Supported

[1] Names are implementation defined.

[2] Application dependent.

SETKEYLS (Set Cursor to Last Record in Key Sequence) *Level 1.0*
This command sets the cursor to the last record of the file in key sequence order.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported

Parameter name	Source	Target
RECNBRFB	Requested	Supported
RTNREC[2]	Sent	Supported
UPDINT[2]	Sent	Supported

[1] Names are implementation defined.

[2] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETKEYNX (Set Cursor to Next Record in Key Sequence) *Level 1.0*

This command sets the cursor to the next record of the file in the key sequence order following the record currently indicated by the cursor.

Parameter name	Source	Target
BYPDMG[1]	Not sent	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECCNT[1]	As required	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETKEYPR (Set Cursor to Previous Record in Key Sequence) *Level 1.0*
This command sets the cursor to the previous record of the file in the key sequence order preceding the record currently indicated by the cursor.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECCNT[2]	As required	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Names are implementation defined.

[2] Application dependent.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETLST (Set Cursor to Last Record) *Level 1.0*
This command sets the cursor to the last record of the file.

Parameter name	Source	Target
BYPINA[1]	As required	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

Parameter name	Source	Target

1
Application dependent.

2
Names are implementation defined.

3
IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2)

RECNBR
Record number

RECORD
Record

SETMNS (Set Cursor Minus) *Level 1.0*

This command sets the cursor to the record number of the file indicated by the cursor minus the number of record positions specified by CSRDSP.

Parameter name	Source	Target
ALWINA[1]	As required	Supported
CSRDSP[1]	Sent	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNINA[1]	As required	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

1
Application dependent.

2
Names are implementation defined.

3
IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECNBR
Record number

RECORD
Record

SETNBR (Set Cursor to Record Number) *Level 1.0*
This command sets the cursor to the record of the file indicated by the record number specified by **RECNBR**.

Parameter name	Source	Target
ALWINA[1]	As required	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBR	Sent	Supported
RTNINA[1]	As required	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECORD
Record

SETNXT (Set Cursor to Next Number) *Level 1.0*
This command sets the cursor to the next record of the file with a record number one greater than the current cursor position.

Parameter name	Source	Target
BYPDMG[1]	Not sent	Supported
BYPINA[1]	As required	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported

Parameter name	Source	Target
RECCNT[1]	As required	Supported
RECNBRFB[1]	As required	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECNBR
Record number

RECORD
Record

SETNXTKE (Set Cursor to Next Record in Key Sequence with a Key Equal to Value Specified) *Level 1.0*
This command positions the cursor to the next record in the key sequence if the key field of that record has a value equal to the value specified in the KEYVAL parameter. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined
HLDCSR	N/A	Supported
KEYVAL[2]	N/A	Max = 2000
KEYVALFB	N/A	Supported
RECNBRFB	N/A	Supported
RTNREC[3]	N/A	Supported
UPDINT[3]	N/A	Supported

[1] Names are implementation defined.

[2] Maximum key size allowed by an IBM i.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETPLS (Set Cursor Plus) Level 1.0

This command sets the cursor to the record number of the file indicated by the cursor plus the integer number of records specified by **CSRDSP**.

Parameter name	Source	Target
ALWINA[1]	As required	Supported
CSRDSP[1]	Sent	Supported
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNINA[1]	As required	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] IBM i preferred implementation.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECNBR
Record number

RECORD
Record

SETPRV (Set Cursor to Previous Record) Level 1.0

This command sets the cursor to the record of the file with a record number one less than the current cursor position.

Parameter name	Source	Target
BYPINA[1]	As required	Supported

Parameter name	Source	Target
DCLNAM[2]	Program defined	Program defined
HLDCSR	Requested	Supported
KEYVALFB	Requested	Supported
RECCNT[1]	As required	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported
UPDINT[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] IBM i preferred implementation.

The following reply objects are possible:

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2 **15-2)

RECNBR
Record number

RECORD
Record

SETUPDKY (Set Update Intent by Key Value) *Level 1.0*
This command places an update intent on the record that has a key value equal to the key value specified by KEYVAL.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
KEYVAL[2]	Max = 2000	Max = 2000
KEYVALFB	Requested	Supported
RECNBRFB	Requested	Supported
RTNREC[3]	Sent	Supported

[1] Names are implementation defined.

[2] Maximum key size allowed by an IBM i.

[3] Only RTNREC(FALSE) is supported.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECNBR
Record number

RECORD
Record

SETUPDNB (Set Update Intent by Record Number) *Level 1.0*
This command places an update intent on the record of the file indicated by the record number specified by **RECNBR**.

Parameter name	Source	Target
ALWINA[1]	As required	Supported
DCLNAM[2]	Program defined	Program defined
KEYVALFB	Requested	Supported
RECNBR	Sent	Supported
RTNINA[1]	As required	Supported
RTNREC[3]	Sent	Supported

[1] Application dependent.

[2] Names are implementation defined.

[3] Only RTNREC(FALSE) is supported.

The following reply objects are possible:

KEYVAL
Key value

RECAL
Record attribute list

RECINA
Inactive record (-1 not supported, maximum = 2**15-2) (2 to the power of 15 minus 2)

RECORD
Record

ULDRECF (Unload Record File) *Level 1.0*
This command sends records from a target record file to the source.

Parameter name	Source	Target
ACCDORD[1]	Sent	Supported
BYPDMG[1]	Sent	Supported
FILNAM[2]	Sent	IBM i name
RTNINA[1]	Sent	Supported

[1] Application dependent.

[2] Name formats are server defined.

The following reply objects are possible:

RECAL
Record attribute list

RECCNT
Record count

RECINA
Inactive record (-1 not supported, maximum = 2 **15-2) (2 to the power of 15 minus 2)

RECORD
Record

ULDSTRF (Unload Stream File) *Level 2.0*
This command sends a document from the target to the source. This command is sent by a source IBM i when using the Copy Stream File (QHFCPYSF) HFS API.

Parameter name	Source	Target
BYPDMG	Not sent	FALSE only
FILNAM[1]	Sent	IBM i name
STRORD	Not sent	Supported
[1]Name formats are server defined.		

The following reply objects are possible:

STREAM
Stream

STRPOS
Stream position

STRSIZ
Stream size

UNLFIL (Unlock File) *Level 1.0 and Level 2.0*
This command releases explicit file locks held by the requester on the file.

Parameter name	Source	Target
FILNAM[1]	Target name	IBM i name
LCKMGRNM	Not sent	Ignored
RLSFILLK	Sent	Supported
[1]Name formats are server defined.		

UNLIMPLK (Unlock Implicit Record Lock) *Level 1.0*
This command releases all implicit record locks currently held by the cursor.

Parameter name	Source	Target
DCLNAM[1]	Program defined	Program defined
[1]Names are implementation defined.		

UNLSTR (Unlock Substreams) *Level 2.0 and Level 3.0*
This command unlocks a stream file substream. This command is not sent by a source IBM i.

Parameter name	Source	Target
DCLNAM[1]	N/A	Program defined

Parameter name	Source	Target
STRLOC	N/A	Supported
[1]Names are implementation defined.		

User profile authority

The user profile associated with target IBM i jobs must be authorized to the equivalent CL commands before the DDM command can be processed. The target job's user profile must be authorized to use the CL commands listed here before DDM requests can be processed.

Table 16. User profile authority CL commands

DDM command received	DDM command description	Object type	Authorized CL command
CHGDRC	Change Current Directory	FLR	NONE
CHGFAT	Change File Attributes	PFILE LF DOC/FLR	CHGPF CHGLF NONE
CLOSE	Close File	FILE DOC	NONE [1] NONE
CLRFIL	Clear File	FILE DOC	NONE NONE
CLSDRC	Close Directory	FLR	NONE
CPYFIL	Copy File	DOC	NONE
CRTAIF	Create Alternate Index File	LF	CRTLF
CRTDIRF	Create Direct File	PF	CRTPF
CRTKEYF	Create Key File	PF	CRTPF
CRTSEQF	Create Sequential File	PF	CRTPF
CRTSTRF	Create Stream File	DOC	NONE
CRTDRC	Create Directory	LIB FLR	CRTLIB CRTFLR
DELFIL	Delete File	FILE DOC	DLTF NONE
DELDRC	Delete Directory	LIB FLR	DLTLIB NONE
GETDRCEN	Get Directory Entry	DOC/FLR	NONE
LCKFIL	Lock File	FILE	ALCOBJ
LODRECF	Load (Put) Records to File	FILE	NONE [2]
LSTFAT	List File Attributes	FILE DOC/FLR	NONE [3] NONE
OPEN	Open File	FILE DOC	NONE [1] NONE
OPENDRC	Open Directory	FLR	NONE
QRYSPC	Query Space Available to User	USRPRF	NONE [4]
RNMDRC	Rename Directory	FLR LIB	NONE RNMOBJ
RNMFIL	Rename File	FILE DOC MBR	RNMOBJ NONE RNMM
UNLFIL	Unlock File	FILE	NONE [5]
ULDRECF	Unload Records From File	FILE	NONE [2]

Table 16. User profile authority CL commands (continued)

DDM command received	DDM command description	Object type	Authorized CL command

1

Authorization to a command is not verified because there are means other than using a command interface by which IBM i users can open and close files.

2

Command authorization is not verified because there is not a direct, one-to-one mapping between a CL command and the DDM **LODRECF/ULDRECF** command.

3

Authorization to the **DSPFD** and **DSPFFD** commands is not verified because it cannot be determined which command should be verified. In addition, the conditions under which the DDM command was issued by the client system are not known.

4

The space available to a user can be obtained by issuing the **DSPUSRPRF** command, but this is only a small piece of the data available through the use of this command.

5

Authorization to the CL **DLCOBJ** command is not checked because if the remote user was able to allocate files, DDM must be able to deallocate them.

The following table is an explanation of the object type codes used in the preceding table.

Table 17. Object type codes definition

Object type	Object type definition
DOC	Document
FLR	Folder
PF	Physical file
LF	Logical file
LIB	Library
MBR	Member
SRCF	Source physical file
USRPRF	User profile

IBM i-to-CICS considerations with DDM

This topic describes programming considerations for accessing CICS remote files with IBM i DDM.

Note: A System/370 host must have installed CICS/OS/VS Version 1.7 or later and CICS/DDM Version 1.1.

IBM i languages, utilities, and licensed programs

The IBM i languages, utilities, and licensed programs discussed in this topic can access remote CICS files.

- Programs written in the following languages on the IBM i can access remote CICS files:

  ILE C programming language
  See "ILE C considerations" on page 222.

  CL
  See "IBM i CL considerations" on page 216.

  ILE COBOL programming language
  See "ILE COBOL considerations" on page 220.

  See "PL/I considerations" on page 218.

ILE RPG programming language
See “ILE RPG considerations” on page 222.

- Programs written in BASIC might cause results that cannot be predicted when accessing remote CICS files.
- IBM i Query can access the remote entry sequence data set (ESDS), relative record data set (RRDS), and key sequence data set (KSDS). However, IBM i Query cannot access virtual storage access method (VSAM) files through DDM.
- The IBM i Access Family licensed program might cause unpredictable results when accessing remote CICS files on behalf of IBM i languages, utilities, and licensed programs.

Note: Some of the high-level languages provide access to the server database I/O feedback area. When accessing a remote VSAM RRDS, this area will contain the relative record number. However, when accessing other types of VSAM data sets, the relative record number is not known and a value of -1 is returned as the relative record number.

Additional considerations might apply when accessing CICS files that are to be read or written by a System/370 host due to the way a System/370 host stores data. For example, the System/370 host representation of a floating-point number is different from the IBM i representation of a floating-point number.

Create DDM File (CRTDDMF) considerations
For applications running on the IBM i operating system that access remote files, the programmer must use the CRTDDMF command to create an object called a DDM file.

The ACCMTH parameter of this command shows which DDM access method should be used when opening the remote file. If *RMTFILE is used, IBM i DDM selects an access method that is compatible with:

- The type of VSAM data set being accessed
- The access methods supported by CICS/DDM for the VSAM data set

The table below shows how the possible values for the ACCMTH parameter correspond to VSAM data sets.

Table 18. ACCMTH parameter of IBM i ***CRTDDMF*** *command*

ACCMTH parameter values	**VSAM data set organization**			
	ESDS	**RRDS**	**KSDS**	**VSAM Path**
*ARRIVAL	R	R	E	E
*KEYED	E	E	R	R
*BOTH	E	O	O	O
*RANDOM	E	O	O	O
*SEQUENTIAL	R	O	O	O
*COMBINED	E	O	E	E

Where:

R
Shows the parameter is required for accessing the VSAM data set.

O
Shows the parameter is optional for accessing the VSAM data set.

E
Shows the parameter causes an IBM i message.

To improve performance, the IBM i user might want to supply values other than *RMTFILE for the ACCMTH parameter. To avoid server messages, use the values specified in Table 18 on page 215 when accessing remote VSAM data sets.

The value specified for the RMTFILE file parameter must be the same as the value specified for the DATASET parameter of the CICS DFHFCT macro when the VSAM data set is defined to the CICS system.

IBM i CL considerations

Besides the information in this topic collection, consider these topics when using IBM i CL commands to access VSAM data sets on a remote CICS system.

Note: Commands that do not appear in the following topics have no considerations besides those stated in this topic collection.

Allocate Object (ALCOBJ)

Unless the CICS system programmer replaces the CICS/DDM-supplied exclusive file lock program with a special version of the program, a lock condition value of *SHRRD or *SHRUPD must be used when using the Allocate Object (ALCOBJ) command to allocate a remote VSAM data set. All other lock condition values result in an IBM i system message being issued.

Clear Physical File Member (CLRPFM)

The **Clear Physical File Member (CLRPFM)** command cannot clear a VSAM data set on a remote CICS system.

Copy File (CPYF)

The Copy File (CPYF) command can access remote VSAM data sets defined to a CICS system.

However, listed here is what you need to pay attention to:

- When the TOFILE parameter is a remote VSAM data set:
  - The CRTFILE parameter must have a value of *NO.
  - The MBROPT parameter must have a value of *ADD.
  - The FMTOPT parameter must have a value of *NOCHK.
- When the TOFILE parameter is a remote VSAM ESDS or KSDS, the COMPRESS parameter must have a value of *YES.

***CPYTOTAP**, **CPYFRMTAP** and **CPYSPLF** commands*

The **Copy to Tape (CPYTOTAP)**, and the **Copy from Tape (CPYFRMTAP)** commands access remote VSAM data sets defined to a CICS system.

However, *ADD must be used for the MBROPT parameter. The **Copy Spool File (CPYSPLF)** command cannot access remote VSAM data sets.

Deallocate Object (DLCOBJ)

The **Deallocate Object (DLCOBJ)** command can release any lock successfully acquired for a remote VSAM data set.

***DSPFD** and **DSPFFD** commands*

The **Display File Description (DSPFD)** and **Display File Field Description (DSPFFD)** commands can display information about a remote VSAM data set.

However, the information for many of the fields is not available to CICS/DDM and is not returned to IBM i DDM. Refer to the following table for the fields that CICS/DDM does not return:

Table 19. CICS/DDM file and file field descriptions

Field	Value
Creation date	Zeros

Table 19. CICS/DDM file and file field descriptions (continued)

Field	Value
Current number of records	Zeros
Data space in bytes	Zeros
File level identifier	Zeros
File text description	Zeros
Format level identifier	Blanks
Last change date and time	Zeros
Member creation date	Zeros
Member expiration date	*NONE
Member level identifier	Zeros
Member size	*NOMAX
Number of deleted records	Zeros
Text description	Blanks
Total deleted records	Zeros
Total member size	Zeros
Total records	Zeros

Note: The values displayed do not represent actual data about the file. They act as default values for the information that CICS/DDM does not return.

When the type of file is logical, the information displayed shows that unique keys are not required. In fact, CICS/DDM does not know whether unique keys are required or not.

Sometimes, the IBM i user needs to know the type of VSAM data set being accessed. By using the following information, which is displayed by the DSPFD command, it is possible for the IBM i user to make this decision:

- If the type of file is logical, the VSAM data set is a VSAM path.
- If the type of file is physical and the access path is keyed, the VSAM data set is KSDS.
- In all other cases, the VSAM data set is either an RRDS or an ESDS. If the IBM i user must know whether it is an RRDS or an ESDS, the CICS system programmer should be contacted.

Display Physical File Member (DSPPFM)

The **Display Physical File Member (DSPPFM)** command can be used to access remote relative record data set (RRDS). It does not work for other types of VSAM data sets.

Open Database File (OPNDBF)

The Open Database File (OPNDBF) command can open a remote VSAM data set.

However, a system message is created on the IBM i if *ARRIVAL is used for the ACCPTH parameter and the remote data set is a VSAM key-sequenced data set (KSDS) or a VSAM path.

Override with Database File (OVRDBF)

The Override with Database File (OVRDBF) command can override a local database file with a remote VSAM data set.

However, the following items must be considered:

- A POSITION value of *RRN works when the remote VSAM data set is an RRDS. For all other types of VSAM data sets, *RRN causes a system message.

- A POSITION value of *KEYB or *KEYBE causes an IBM i system message to be issued when the remote file is a VSAM path.
- Unless the CICS system programmer replaces the CICS/DDM-supplied exclusive file lock program, the RCDFMTLCK parameter must have a lock condition value of *SHRRD or *SHRUPD. All other lock condition values result in a system message being issued.
- CICS/DDM does not return the actual expiration date of the file to IBM i DDM. Instead, it returns a special value that indicates the expiration date is not known. This is true even if the EXPCHK parameter has a value of *YES.

Receive Network File (RCVNETF)

The **Receive Network File (RCVNETF)** command can access a VSAM data set defined to a remote CICS system. However, the MBROPT parameter must have a value of *ADD.

Language considerations for IBM i and CICS

IBM i application programmers using ILE COBOL, ILE C, IBM i System/36-Compatible RPG II, or ILE RPG languages should be aware of the information in these topics.

PL/I considerations

These topics summarize the limitations that exist when using PL/I to access remote VSAM data sets from the IBM i. These limitations should be considered in addition to those already stated in this topic collection.

PL/I open file requests

The IBM i DDM user can access remote CICS files with PL/I programs.

When opening the file with the RECORD file attribute, the program must use the file attributes specified in the table below. By noting the values that appear in this table, you can determine the difference between accessing an IBM i database file and a remote VSAM data set.

Note: Remote files can also be opened with the PL/I STREAM file attribute. However, if the STREAM file attribute is used to open a VSAM KSDS, a server message occurs. This happens because records in a VSAM KSDS cannot be processed in arrival sequence.

Unless the CICS system has replaced the CICS/DDM exclusive file locking program, you cannot use the EXCL and EXCLRD file locking options for the ENVIRONMENT parameter when opening a remote VSAM data set.

Table 20. PL/I file attributes

	VSAM data set organization			
PL/I file attributes	**ESDS**	**RRDS**	**KSDS**	**VSAM Path**
SEQUENTIAL	R	O	O	O
DIRECT	E	O	O	O
SEQL KEYED	E	O	O	O
INPUT	O	O	O	O
OUTPUT	O	O	O	E
UPDATE	O	O	E	E
CONSECUTIVE	R	R	E	E
INDEXED	–	–	R	R

Table 20. PL/I file attributes (continued)

	VSAM data set organization			
PL/I file attributes	**ESDS**	**RRDS**	**KSDS**	**VSAM Path**
Where: **R** Shows the attribute is required for accessing the VSAM data set. **O** Shows the attribute is optional for accessing the VSAM data set. **E** Shows the attribute is allowed by PL/I but the open fails when accessing the VSAM data set. **–** Shows the option is valid for keyed files only.				

PL/I input/output requests
This topic discusses the types of PL/I input/output requests and their restrictions.

Read requests

- A KEYSEARCH parameter value of BEFORE or EQLBFR is not supported by CICS/DDM when accessing a VSAM path. However, these parameter values are supported when accessing a VSAM KSDS.
- A POSITION parameter value of PREVIOUS and LAST is not supported when accessing a VSAM path. However, these parameter values are supported when accessing a VSAM KSDS.
- Because the DIRECT or SEQUENTIAL KEYED attributes cannot be used to open a VSAM ESDS, it is not possible to access records by relative record number or to have the relative record number returned by using the KEY and KEYTO parameters.
- Because VSAM KSDS and VSAM alternate indexes are always defined as a single key field, the NBRKEYFLDS parameter should not be used.

Write requests

- The KEYFROM parameter does not work when writing a record to a VSAM RRDS.
- The WRITE request does not work when writing a record to VSAM KSDS that already contains a record with the same key value.
- Because the OUTPUT or UPDATE file attribute cannot be used to open a VSAM path, it is not possible to write records to a VSAM path. Instead, the application program must write the record by using the base data set of the VSAM path.

Rewrite requests

- The REWRITE does not work if rewriting a record of a VSAM KSDS when the key value of the record is changed.
- Because the UPDATE file attribute cannot be used to open a VSAM path, it is not possible to rewrite records in a VSAM path. Instead, the application program must rewrite the record by using the base data set of the VSAM path.

Delete requests

- The DELETE does not work when deleting the record of a VSAM ESDS.
- Because the UPDATE file attribute cannot be used to open a VSAM path, it is not possible to delete records in a VSAM path. Instead, the application program must delete the records by using the base data set of the VSAM path. However, if the base data set of the VSAM path is a VSAM ESDS, the DELETE does not work.

ILE COBOL considerations
These topics summarize the limitations that exist when using the ILE COBOL programming language to access remote VSAM data sets from the IBM i.

Unless otherwise stated, these limitations apply to both the System/36 and the IBM i. These limitations are in addition to those already stated in this topic collection.

ILE COBOL SELECT clause
IBM i users can access remote CICS files with the ILE COBOL programming language.

However, the ILE COBOL SELECT clause must use the file organizations and access methods specified in the following table.

Table 21. ILE COBOL file organizations and access methods

ILE COBOL programming language	**VSAM data set organization**				
Program-given access methods	**ESDS**	**RRDS**	**KSDS**	**VSAM Path**	**Program-given organization**
Sequential	Sequential	X	X	E	E
Relative	Sequential Random Dynamic	E E E	X X X	E E E	E E E
Indexed	Sequential Random Dynamic	– – –	– – –	X X X	X X X

Where:

X
Shows the access method is allowed.

E
Shows that ILE COBOL programming language allows the access method but that the open fails when accessing the VSAM data set. An IBM i message is created.

–
Shows the option is never valid for nonkeyed files. An IBM i message occurs whenever indexed file organization is selected for any nonkeyed file. This is true even when the file is a local file.

Notes:

1. When accessing a VSAM path, the WITH DUPLICATE phrase should be used.
2. When accessing a VSAM KSDS, the WITH DUPLICATE phrase should not be used.

ILE COBOL statements
These topics describe considerations you should be aware of when using ILE COBOL statements to access remote VSAM data sets.

ILE COBOL OPEN statement

When accessing remote CICS files, the ILE COBOL OPEN statement must use the open modes that are specified in the following table.

Table 22. Use ILE COBOL programming language to open a CICS file

ILE COBOL open mode	VSAM data set organization			
	ESDS	**RRDS**	**KSDS**	**VSAM Path**
Input	X	X	X	X
Output	E	E	E	E
Input/Output	X	X	X	E
Extend	X	–	–	–

Where:

X
Shows that the open mode is allowed.

E
Shows that the open mode is allowed, but that the open operation fails when the ILE COBOL OPEN statement accesses the VSAM data set. A message occurs on the IBM i operating system.

–
Shows the open mode is not applicable.

ILE COBOL READ statement

- The PRIOR phrase and the LAST phrase are not supported by CICS/DDM when accessing a VSAM path. It is supported when accessing a VSAM KSDS.
- Because the RELATIVE file organization can only be used to open a VSAM RRDS, it is not possible to access records by relative record number or to have the relative record number returned from the remote file unless the remote file is a VSAM RRDS.

ILE COBOL WRITE statement

- The WRITE statement does not work when the ILE COBOL program is running on the IBM i operating system and the file was opened with a RELATIVE file organization.
- The WRITE statement does not work when writing a record to a VSAM KSDS and the data set already contains a record with the same key value.
- Because the input/output and output open modes cannot be used to open a VSAM path, it is not possible to write records to a VSAM path. Instead, the application program must write the record by using the base data set of the VSAM path.

ILE COBOL REWRITE statement

- The REWRITE statement does not work when rewriting the record of a VSAM KSDS and the key value of the record is changed.
- Because the input/output open mode cannot be used to open a VSAM path, it is not possible to rewrite records in a VSAM path. Instead, the application program must rewrite the record by using the base data set of the VSAM path.

ILE COBOL START statement

- The START statement does work if the open mode is INPUT.

ILE COBOL DELETE statement

- Because the sequential file organization must be used to open a VSAM ESDS, it is not possible to delete records in a VSAM ESDS.

- Because the input/output open mode cannot be used to open a VSAM path, it is not possible to delete records in a VSAM path. Instead, the application program must delete the record by using the base data set of the VSAM path. However, if the base data set of the VSAM path is a VSAM ESDS, the DELETE does not work.

ILE C considerations
These topics summarize considerations for using the ILE C programming language to access remote VSAM data sets from the IBM i operating system.

ILE C open considerations

Because ILE C programming language supports only sequential I/O, open operations will fail if KSDS or VSAM paths are opened.

Open mode considerations

This topic shows the open mode considerations when using ILE C programming language.

Table 23. Using ILE C programming language to open a CICS file

	VSAM data set organization			
ILE C open mode	**ESDS**	**RRDS**	**KSDS**	**VSAM Path**
r, rb	X	X	X	X
w, wb	E	E	E	E
w+, wb+, w+b, a+, ab+, a+b, r+, rb+, r+b, a, ab	X	X	X	E
a, ab	X	—	—	—

Where:

X
Open mode is allowed.

E
Open mode is allowed by ILE C programming language, but the open fails when accessing the VSAM data set.

—
Open mode is not applicable.

ILE RPG considerations
These topics summarize the limitations that exist when using the IBM i System/36-Compatible RPG II or the ILE RPG programming language on the IBM i operating system to access remote VSAM data sets.

These limitations are in addition to those already stated in this topic collection.

File description specifications
IBM i users can access remote VSAM data sets with the IBM i System/36-Compatible RPG II language or the ILE RPG programming language.

However, not all ILE RPG processing methods selected by the file description specifications can access remote VSAM data sets. Refer to the following tables to determine which file description specifications to use. If a file description specification other than what appears in these tables is used, CICS/DDM rejects the request for opening the file.

Table 24. ILE RPG processing methods for remote VSAM ESDS

Type of processing	Column number						
	15	16	19	28	31	32	66
Consecutive	I I I I U U U	P S T D P S D	F F F F F F F				
Add Records Only	O						A

Table 25. ILE RPG processing methods for remote VSAM RRDS

Type of processing	Column number						
	15	16	19	28	31	32	66
Consecutive	I I I U U U	P S D P S D	F F F F F F				
Random by Chain See note. See note.	I I U U	C C C C	F F F F	R R R R			A A
Random by Addrout	I I I U U U	P S F P S F	F F F F F F	R R R R	I I I I		
Consecutive, Random or both See note. See note.	I I U U	F F F F	F F F F				A A
Note: A **K** must be used in column 53 and **RECNO** must be in columns 54 through 59 to indicate relative record number processing.							

Table 26. ILE RPG processing methods for VSAM KSDS

	Column number						
Type of processing	**15**	**16**	**19**	**28**	**31**	**32**	**66**
Sequential	I	P	F	L		I	A
By key, no add	I	S	F	L		I	A
By key, no add	I	D	F	L		I	A
By key, no add	I	P	F	L		I	A
By key, with add	I	S	F	L		I	A
By key, with add	I	D	F	L		I	A
By key, with add	U	P	F	L		I	A
By key, no add	U	S	F	L		I	A
By key, no add	U	D	F	L		I	
By key, no add	U	P	F			I	
By key, with add	U	S	F			I	
By key, with add	U	D	F			I	
By key, with add	I	P	F			I	
By limits	I	S	F			I	
By limits	I	F	F			I	
By limits	U	D	F			I	
By limits	U	P	F			I	
By limits	U	D	F			I	
By limits	U	F	F			I	
By limits	I	F	F			I	
By limits, adds	I	F	F			I	
By limits, adds							
Random by Chain	I	C	F	R		I	A
No adds	I	C	F	R		I	A
With adds	U	C	F	R		I	
No adds	U	C	F	R		I	
With adds							
Random by Addrout	I	P	F	R	I	I	
	I	S	F	R	I	I	
	U	P	F	R	I	I	
	U	S	F	R	I	I	
Sequential, Random or both	I	F	F			I	A
By key, no add	I	F	F			I	A
By key, with add	U	F	F			I	
By key, no add	U	F	F			I	
By key, no add							
Add Records Only	O					I	A

Table 27. Processing methods for remote VSAM paths

Type of processing	Column number						
	15	16	19	28	31	32	66
Sequential	I	P	F	L		I	
By key, no add	I	S	F	L		I	
By key, no add	I	D	F	L		I	
By key, no add	I	P	F	L		I	
By limits	I	S	F			I	
By limits	I	F	F			I	
By limits	I	D	F			I	
By limits							
Random by Chain No adds	I	C	F	R		I	
Random by Addrout							
	I	P	F	R	I	I	
	I	S	F	R	I	I	
Sequential, Random or both By key, no add	I	F	F			I	

ILE RPG input/output operations
Be aware of these restrictions when accessing remote VSAM data sets.

- Records can be read or added by relative record number only when the remote VSAM data set is an RRDS. Random processing by relative record number can be used only when processing a VSAM RRDS.
- A request to delete a record in an ESDS does not work because ESDS is never delete-capable.
- The processing method cannot use the update or output specification in column 15 when the remote VSAM data set is a VSAM path. Instead, all add, update, or delete record requests must be made by using the base data set of the VSAM path. However, if the base data set of the VSAM path is a VSAM ESDS, the DELETE does not work.
- The READP operation code cannot be used to read the previous record in a VSAM path.
- Because VSAM KSDS does not allow duplicate keys, a request to add a record that duplicates the key of an existing record in a KSDS does not work.
- When accessing a KSDS, an update request that changes the key value of the record does not work.
- For ILE RPG programming language, *HIVAL can be used to obtain the last record of a remote KSDS. However, *HIVAL does not work when accessing a VSAM path.

Using DDM on IBM i versus other IBM systems

This topic describes the DDM differences between IBM i and System/36, and IBM i and System/38.

IBM i and System/36 DDM differences

This topic consists of a list of differences between the IBM i and System/36.

- The network resource directory (NRD) procedures are not supported on the IBM i.
  - The System/36 NRD used one or more libraries containing DDM files on the IBM i. One System/36 NRD entry is equivalent to one DDM file on the IBM i.
    - Use the **`Work with DDM Files (WRKDDMF)`** command in place of the EDITNRD and DELNRD procedures.
    - Use the **`Display DDM Files (DSPDDMF)`** command in place of the LISTNRD procedure.
    - Use the **`Restore Object (RSTOBJ)`** command in place of the RESTNRD procedure.

- Use the **Save Object (SAVOBJ)** command in place of the SAVNRD procedure.
  - If an NRD entry on the System/36 refers to a remote file, and a SAVE or RESTORE procedure is requested, the System/36 saves or restores the data of the remote file. The IBM i **Save Object (SAVOBJ)** or **Restore Object (RSTOBJ)** command saves or restores only the definition of the DDM file, not the remote data.
  - When using date-differentiated files on the System/36, the most current file is selected. When running from a System/36 to an IBM i, the NRD entry must specify a member name of *LAST to access the last member of the database file, which is the file with the most recent date. If no member name is specified, *FIRST is used.
- The remote label in the NRD on a System/36 source must be in the syntax required by the server system.
- The number of records allocated for a file differs between the System/36 and the IBM i. File space allocation on the System/36 is in block units (2560 bytes) while on the IBM i, file space allocation is by number of records. For example, if a user asks for a sequential file capable of storing ten 100-byte records, the System/36 allocates 1 block of file space (2560 bytes), and uses as much of the file space as possible (2500 bytes), giving the user twenty-five 100-byte records.

  The IBM i allocates exactly 10 records. If the user took advantage of this allocation method on the System/36, the file (nonextendable) created using DDM on the IBM i might be too small.
- The DDM **Change End-of-File (CHGEOF)** command used on the System/36 is not supported on the IBM i.
- The IBM i does not support writing over data on the **Delete File (DLTF)** command. If an IBM i user accessing a System/36 wants to overwrite the data, an application program must be written on the IBM i, or the user must access the target System/36 and perform the delete operation with the erase operation.
- A System/36 client system cannot create direct files on an IBM i target server that are nondelete-capable. The IBM i does not support files that are nondelete-capable for all file organizations.
- The System/36 does not allow a record with hex FF in the first byte to be inserted into a delete-capable file. The IBM i allows this.
- When running System/36 applications to another System/36, the application waits indefinitely for the resource to become available. When running System/36 applications to or from an IBM i, these applications might result in time-outs while waiting for a resource to become available.
- Direct files are extendable on the System/36 but not on the IBM i. If a direct file is created on the IBM i as extendable, the file is allocated with three extents, but is never extended beyond the initial size plus three extents.
- The System/36 relay function is *not* supported whenever one of the servers in the network along the path from the client system to the server system is not a System/36. The IBM i *never* supports the relay function; Advanced Peer-to-Peer Networking (APPN) must be used.
- Key fields on the System/36 are considered to be strictly character fields. The System/36 does not recognize a key field as being packed or zoned. Unexpected results might occur if source IBM i application programs refer to packed fields within a System/36 file. Because of the way packed numbers are stored and the fact that the System/36 does not recognize key fields as packed on relative keyed operations, records might be returned in a different order than expected or no record might be found when one is expected.

  As an example, the ILE RPG SETLL statement might fail unexpectedly with record not found or might select a record other than the record expected when using packed fields to a System/36 file. Only character and unsigned numeric fields should be used as key fields.

IBM i and System/38 DDM differences

This topic consists of a list of differences between IBM i and System/38 DDM.

- DDM files can be created either in the System/38 environment or on the IBM i operating system.

- The **Submit Remote Command (SBMRMTCMD)** command must be in the syntax of the server system, even in the System/38 environment. For example, when a System/38 product submits commands to the IBM i operating system, the IBM i syntax must be used.
- The remote file name must be in the syntax of the server system.
- The default value for the DDMACC parameter on the **Change Network Attributes (CHGNETA)** command on the System/38 is *REJECT. The default value for the DDMACC parameter on the IBM i operating system is *OBJAUT.
- On the System/38 platform, files are created as FIFO (first in, first out) or LIFO (last in, first out). The default for creating a file is FIFO on the System/38.

  When running System/38 applications that are dependent on duplicate keys being FIFO or LIFO to the IBM i operating system, you should specify either FIFO or LIFO when creating your IBM i files because there is no default for IBM i files. This means the IBM i operating system looks for an available index path to share, which can be either FIFO or LIFO.
- Keyed files containing fields other than character (zoned or packed) created by using DDM on a remote System/38 might result in the fields defined as character fields. This might produce unexpected results when such a file is processed using relative keyed operations. Because the file is created with fields that are not packed, records might be returned in a different order than expected or no record can be found when one is expected.

  As an example, the ILE RPG SETLL statement might fail unexpectedly with record not found or might select a record other than the record expected when using packed fields to a System/38 file. Only character and unsigned numeric fields should be used as key fields for files that are created by using DDM on the remote System/38.
- To support adding a record by the relative record number operation, an ILE RPG program is required for DDM to do a READ CHAIN(RRN) operation followed by a WRITE operation. The file must be opened for read and update authorities, and the user must have read and update data authorities to the file.

  Format selector programs on adding a record by the relative record number operation are only supported by the IBM i operating system. Incompatibilities might arise for those users who have a format selector program for a logical file if they do direct file processing.

DRDA and DDM administration

As an administrator for a distributed relational database, you are responsible for work that is done on several systems.

Work that originates on your local system as a client can be monitored in the same way as any other work is monitored on the IBM i operating system.

When you are tracking units of work being done on the local system as a server, you use the same tools but look for different kinds of information.

This topic discusses ways that you can administer the distributed relational database work being done across a network. Most of the commands, processes, and other resources discussed here do not exist just for distributed relational database use. They are tools provided for any IBM i operations. All administration commands, processes, and resources discussed here are included with the IBM i licensed program, along with all of the Db2 for i functions. The IBM i work management functions provide effective ways to track work on several systems.

Related tasks

Setting up DDM files

The IBM i implementation of Distributed Relational Database Architecture (DRDA) support uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Monitoring relational database activity

You can use control language (CL) commands, all of which provide similar information, but in different ways, to give you a view of work on the IBM i operating system.

Working with jobs in a distributed relational database

The **`Work with Job (WRKJOB)`** command presents the Work with Job menu. This menu allows you to select options to work with or to change information related to a specified job. Enter the command without any parameters to get information about the job you are currently using.

Specify a job to get the same information pertaining to it by entering its name in the command like this:

```
WRKJOB JOB(job-number/user-ID/job-name)
```

You can get the information provided by the options on the menu about whether the job is on a job queue, output queue, or active. However, a job is not considered to be on the system until all of its input has been completely read in. Only then is an entry placed on the job queue. The options for the job information are:

- Job status attributes
- Job definition attributes
- Spooled file information

Information about the following options can be shown only when the job is active:

- Job run attributes
- Job log information
- Program stack information
- Job lock information
- Library list information
- Open file information
- File override information
- Commitment control status
- Communications status
- Activation groups
- Mutexes

Option 10 (Display job log) gives you information about an active job or a job on a job queue. For jobs that have ended you can usually find the same information by using option 4 (Work with spooled files). This presents the **`Work with Spooled Files`** display, where you can use option 5 to display the file named QPJOBLOG if it is on the list. The **`Work with Job (WRKJOB)`** command presents the Work with Job menu.

Related reference

Work with Job (WRKJOB) command

Resolving loop, wait, or performance problems

If a request takes longer than expected to complete, check the application requester (AR) first.

Working with user jobs in a distributed relational database
If you know the user profile (user name) being used by a job, you can use the **Work with User Jobs (WRKUSRJOB)** command to display or change job information. Enter the command without any parameters to get a list of the jobs on the system with your user profile.

You can specify any user and the job status to shorten the list of jobs by entering its name in the command like this:

```
WRKUSRJOB USER(KCDBA)
```

The Work with User Jobs display appears with names and status information of user jobs running on the system (*ACTIVE), on job queues (*JOBQ), or on an output queue (*OUTQ). The following display shows the active and ended jobs for the user named KCDBA:

```
Work with User Jobs                       KC105
03/29/92  16:15:33
Type options, press Enter.
2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
8=Work with spooled files   13=Disconnect

Opt   Job         User         Type      -----Status------   Function
__    KC000       KCDBA        CMNEVK    OUTQ
__    KC000       KCDBA        CMNEVK    OUTQ
__    KC000       KCDBA        CMNEVK    OUTQ
__    KC000       KCDBA        CMNEVK    OUTQ
__    KC000       KCDBA        CMNEVK    ACTIVE
__    KC0001      KCDBA        CMNEVK    ACTIVE              *  -PASSTHRU
__    KC0001      KCDBA        INTER     ACTIVE              CMD-WRKUSRJOB




Bottom
Parameters or command
===>
F3=Exit      F4=Prompt   F5=Refresh   F9=Retrieve   F11=Display schedule data
F12=Cancel   F21=Select assistance level
```

This display lists all the jobs on the system for the user, shows the status specified (*ALL in this case), and shows the type of job. It also provides you with eight options (2 through 8 and 13) to enter commands for a selected job. Option 5 presents the Work with Job display described in the preceding paragraphs.

The **Work with User Jobs (WRKUSRJOB)** command is useful when you want to look at the status of the DDM TCP/IP server jobs if your system is using TCP/IP. Run the following command:

```
    WRKUSRJOB QUSER *ACTIVE
```

Page down until you see the jobs starting with the characters QRWT. If the system is active, you should see one job named QRWTLSTN, and one or more named QRWTSRVR (unless prestart DRDA jobs are not run on the system). The QRWTSRVR jobs are prestart jobs. If you do not see the QRWTLSTN job, run the following command to start it:

```
    STRTCPSVR *DDM
```

If you see the QRWTLSTN job and not the QRWTSRVR jobs, and the use of DRDA prestart jobs has not been disabled, run the following command to start the prestart jobs:

```
    STRPJ subsystem QRWTSRVR
```

Before V5R2, the subsystem that QRWTSRVR normally ran in was QSYSWRK. After V5R1, QRWTSRVR runs in QUSRWRK.

Related reference
Work with User Jobs (WRKUSRJOB) command

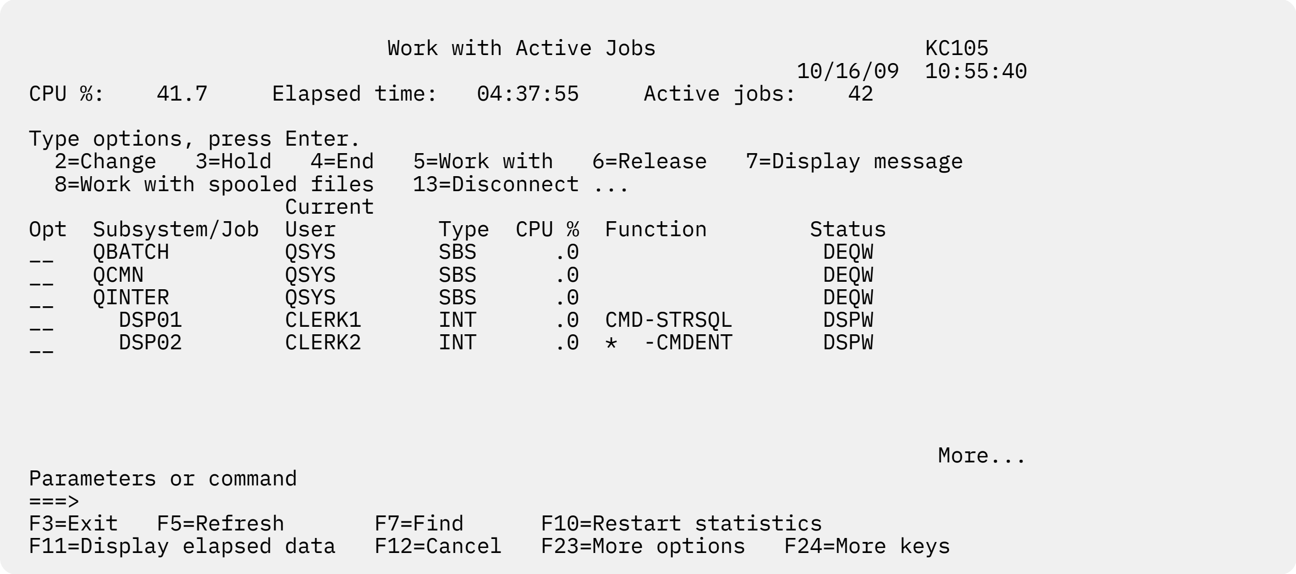
Resolving loop, wait, or performance problems
If a request takes longer than expected to complete, check the application requester (AR) first.

Working with active jobs in a distributed relational database

Use the **Work with Active Jobs (WRKACTJOB)** command if you want to monitor the jobs running for several users, or if you are looking for a job and you do not know the job name or the user ID.

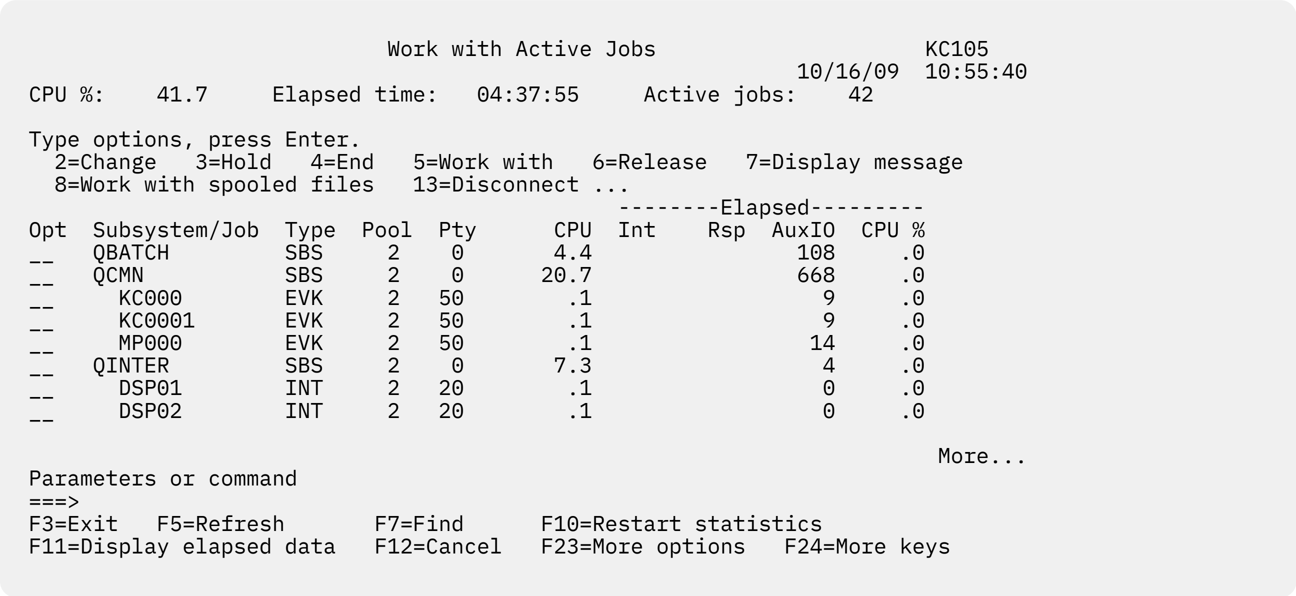
When you enter this command, the Work with Active Jobs display appears. It shows the performance and status information for jobs that are currently active on the system. All information is gathered on a job basis and grouped by subsystem.

The display here shows the Work with Active Jobs display on a typical day at the KC105 system:



```
                            Work with Active Jobs                     KC105
                                                            10/16/09  10:55:40
CPU %:    41.7     Elapsed time:   04:37:55     Active jobs:    42

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...
                    Current
Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                   DEQW
__   QCMN           QSYS        SBS      .0                   DEQW
__   QINTER         QSYS        SBS      .0                   DEQW
__     DSP01        CLERK1      INT      .0  CMD-STRSQL       DSPW
__     DSP02        CLERK2      INT      .0  *  -CMDENT       DSPW




                                                                       More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

When you press F11 (Display elapsed data), the following display is provided to give you detailed status information.



```
                            Work with Active Jobs                     KC105
                                                            10/16/09  10:55:40
CPU %:    41.7     Elapsed time:   04:37:55     Active jobs:    42

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...
                                              --------Elapsed---------
Opt  Subsystem/Job  Type  Pool  Pty      CPU  Int    Rsp  AuxIO  CPU %
__   QBATCH         SBS     2    0       4.4                108     .0
__   QCMN           SBS     2    0      20.7                668     .0
__     KC000        EVK     2   50        .1                  9     .0
__     KC0001       EVK     2   50        .1                  9     .0
__     MP000        EVK     2   50        .1                 14     .0
__   QINTER         SBS     2    0       7.3                  4     .0
__     DSP01        INT     2   20        .1                  0     .0
__     DSP02        INT     2   20        .1                  0     .0
                                                                       More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

The Work with Active Jobs display gives you information about job priority and system usage as well as the user and type information you get from the Work with User Jobs display. You also can use any of 11 options on a job (2 through 11 and 13), including option 5, which presents you with the Work with Job display for the selected job. Use the **Work with Active Jobs (WRKACTJOB)** command if you want to monitor the jobs running for several users or if you are looking for a job and you do not know the job name or the user ID.

Another method to view information about job priority and system usage is to use System i Navigator. To do this, follow these steps:

1. Select **Databases** in the System i Navigator interface.
2. Select a remote database you want to view information about.
3. Right-click and select **Properties**. This opens a properties window with the information displayed.

Related reference

Work with Active Jobs (WRKACTJOB) command

Improving distributed relational database performance through the system
Achieving efficient system performance requires a proper balance among system resources. Overusing any resource adversely affects performance. This topic describes the commands that are available to help you observe the performance of your system.

Resolving loop, wait, or performance problems
If a request takes longer than expected to complete, check the application requester (AR) first.

Working with commitment definitions in a distributed relational database

Use the **`Work with Commitment Definitions (WRKCMTDFN)`** command if you want to work with the commitment definitions on the system.

A *commitment definition* is used to store information about commitment control when commitment control is started by the **`Start Commitment Control (STRCMTCTL)`** command. These commitment definitions might or might not be associated with an active job. Those not associated with an active job have been ended, but one or more of its logical units of work have not yet been completed.

The **`Work with Commitment Definitions (WRKCMTDFN)`** command can be used to work with commitment definitions based on the job name, status, or logical unit of work identifier of the commitment definition.

On the STATUS parameter, you can specify all jobs or only those that have a status value of *RESYNC or *UNDECIDED. *RESYNC shows only the jobs that are involved with resynchronizing their resources in an effort to re-establish a synchronization point; a *synchronization point* is the point where all resources are in consistent state.

*UNDECIDED shows only those jobs for which the decision to commit or roll back resources is unknown.

On the LUWID parameter, you can display commitment definitions that are working with a commitment definition on another system. Jobs containing these commitment definitions are communicating using an APPC-protected conversation. An LUWID can be found by displaying the commitment definition on one system and then using it as input to the **`Work with Commitment Definitions (WRKCMTDFN)`** command to find the corresponding commitment definition.

You can use the **`Work with Commitment Definitions (WRKCMTDFN)`** command to free local resources in jobs that are undecided, but only if the commitment definitions are in a Prepared (PRP) or Last Agent Pending (LAP) state. You can force the commitment definition to either commit or roll back, and thus free up held resources; control does not return to the program that issued the original commitment until the initiator learns of the action taken on the commitment definition.

You can also use the **`Work with Commitment Definitions (WRKCMTDFN)`** command to end synchronization in cases where it is determined that resynchronization cannot be completed with another system.

Related concepts

Troubleshooting transactions and commitment control

Related reference

Start Commitment Control (STRCMTCTL) command

Work with Commitment Definitions (WRKCMTDFN) command

Tracking request information with the job log of a distributed relational database

Every IBM i job has a job log that contains information related to requests entered for a job.

The information in a job log includes:

- Commands that were used by a job
- Messages that were sent and not removed from the program message queues
- Commands in a CL program if the program was created with LOGCLPGM(*JOB) and the job specifies LOGCLPGM(*YES) or the CL program was created with LOGCLPGM(*YES)

At the end of the job, the job log can be written to a spooled file named QPJOBLOG and the original job log is deleted. You can control what information is written in the job log by specifying the LOG parameter of a job description.

The way to display a job log depends on the status of the job. If the job has ended and the job log is not yet printed, find the job using the **Work with User Jobs (WRKUSRJOB)** command, then select option 8 (Display spooled file). Find the spooled file named QPJOBLOG and select option 5 (Display job log). You can also display a job log by using the **Work with Job (WRKJOB)** command and other options on the Work with Job display.

If the batch or interactive job is still active, or is on a job queue and has not yet started, use the **WRKUSRJOB** command to find the job. The **Work with Active Jobs (WRKACTJOB)** command is used to display the job log of active jobs and does not show jobs on job queues. Select option 5 (Work with job) and then select option 10 (Display job log).

To display the job log of your own interactive job, do one of the following things:

- Enter the **Display Job Log (DSPJOBLOG)** command.
- Enter the **Work with Job (WRKJOB)** command and select option 10 (Display job log) from the Work with Job display.
- Press F10 (Display detailed messages) from the Command Entry display to display messages that are shown in the job log.

When you use the **Display Job Log (DSPJOBLOG)** command, you see the Job Log display. This display shows program names with special symbols, as follows:

>>

The running command or the next command to be run. For example, if a CL or high-level language program was called, the call to the program is shown.

>

The command has completed processing.

. .

The command has not yet been processed.

?

Reply message. This symbol marks both those messages needing a reply and those that have been answered.

Related tasks

Printing a job log
Every IBM i job has a job log that contains information related to requests entered for that job. When a user is having a problem at a client, the information in the job log might be helpful in diagnosing the problem.

Related reference

Display Job Log (DSPJOBLOG) command

Work with Active Jobs (WRKACTJOB) command

Work with Job (WRKJOB) command

Work with User Jobs (WRKUSRJOB) command

Distributed relational database messages

If an error message occurs at either a server or a client, the system message is logged on the job log to indicate the reason for the failure.

Locating distributed relational database jobs

When you are looking for information about a distributed relational database job on a client and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

You can also use this command on the server system, but be aware that the user profile on the server might be different from that used by the client. For TCP/IP servers, the user profile that qualifies the job name will always be QUSER, and the job name will always be QRWTSRVR. The Display Log (DSPLOG) command can be used to help find the complete server job name. The message will be in the following form:

```
    DDM job 031233/QUSER/QRWTSRVR servicing user XY on 10/02/97 at 22:06
```

If there are several jobs listed for the specified user profile and the relational database is accessed using DRDA, enter option 5 (Work with job) to get the Work with Job display. From this display, enter option 10 (Display job log) to see the job log. The job log shows you whether this is a distributed relational database job and, if it is, to which remote system the job is connected. Page through the job log looking for one of the following messages (depending on whether the connection is using APPC or TCP/IP):

CPI9150
: Database conversation started over SNA.

CPI9160
: Database connection started over TCP/IP.

The second level text for message CPI9150 and CPI9160 contains the job name for the server job.

If you are on the server and you do not know the job name, but you know the user name, use the Work with User Jobs (WRKUSRJOB) command. If you do not specify a user, the command returns a list of the jobs under the user profile you are using. For TCP/IP, the user profile in the job name will always be QUSER. On the Work with User Jobs display, use these columns to help you identify the server jobs that are servicing APPC connections.

1
: The job type column shows jobs with the type that is listed as CMNEVK for APPC communications jobs.

2
: The status column shows if the job is active or completed. Depending on how the system is set up to log jobs, you might see only active jobs.

3
: The job column provides the job name. The job name on the server is the same as the device being used.

```
Work with User Jobs                      KC105
03/29/92  16:15:33
Type options, press Enter.
2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
8=Work with spooled files   13=Disconnect

Opt   Job          User         Type      -----Status------   Function
__    KC000        KCDBA        CMNEVK    OUTQ
__    MP000        KCDBA        CMNEVK    OUTQ
__    MP000        KCDBA        CMNEVK    OUTQ
__    KC000        KCDBA        CMNEVK    OUTQ
__    KC000        KCDBA        CMNEVK    ACTIVE
__    KC0001       KCDBA        INTER     ACTIVE              CMD-WRKUSRJOB
      3                         1         2
```

If you are looking for an active server job and do not know the user name, the Work with Active Jobs (WRKACTJOB) command gives you a list of those jobs for the subsystems active on the system. The following example shows some items to look for.

```
Work with Active Jobs                     KC105
03/29/92  16:17:45
CPU %:   41.7     Elapsed time:   04:37:55     Active jobs:   102

Type options, press Enter.
2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
8=Work with spooled files   13=Disconnect

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QBATCH         QSYS        SBS      .0                  DEQW
4  QCMN           QSYS        SBS      .0                   WDEQ
__     KC0001       KCCLERK     EVK      .0  *                EVTW
       5                     6
```

4
: Search the subsystem that is set up to handle the server jobs. In this example, the subsystem for server jobs is QCMN. The subsystem for TCP/IP server jobs is QSYSWRK prior to V5R2, and QUSRWRK after V5R1.

5
: For APPC server jobs, the job name is the device name of the device that is created for server use.

6
: The job type listed is normally EVK, started by a program start request. For TCP/IP server jobs, the job type is PJ (unless DRDA prestart jobs are not active on the system, in which case the job type is BCI).

When you have located a job that looks like a candidate, enter option 5 to work with that job. Then select option 10 from the Work with Job Menu to display the job log. Distributed database job logs for jobs that are accessing the server from a Db2 for i client contain a statement near the top that reads:

CPI3E01
: Local relational database accessed by *(system name)*.

After you locate a job working on the server, you can also trace it back to the client if the client is a IBM i product. One of the following messages will appear in your job log; place the cursor on the message you received:

CPI9152
: Target DDM job started by the client system.

CPI9162
: Target job assigned to handle DDM connection started by client over TCP/IP.

When you press the help key, the detailed message for the statement appears. The client job named is the job on the client that caused this job.

Related tasks

Printing a job log
Every IBM i job has a job log that contains information related to requests entered for that job. When a user is having a problem at a client, the information in the job log might be helpful in diagnosing the problem.

Related reference

Display Log (DSPLOG) command

Work with Active Jobs (WRKACTJOB) command

Work with User Jobs (WRKUSRJOB) command

Resolving loop, wait, or performance problems
If a request takes longer than expected to complete, check the application requester (AR) first.

Operating remote systems

As an administrator in a distributed relational database, you might have to operate a remote IBM i product.

For example, you might have to start or stop a remote server. The IBM i product provides options that help you ensure that a remote system is operating when it needs to be. Of course, the simplest way to ensure that a remote system is operating is to allow the remote location to power on their server to meet the distributed relational database requirements. But, this is not always possible. You can set up an automatic power-on and power-off schedule or enable a remote power-on to a remote server.

The system provides several ways to do this either in real time or at previously scheduled times. More often, you might need to perform certain tasks on a remote system as it is operating. The three primary ways that you can do this is by using display station pass-through, the Submit Remote Command (SBMRMTCMD) command, or stored procedures.

The Submit Remote Command (SBMRMTCMD) command submits a CL command using Distributed Data Management (DDM) support to run on the server. You first need to create a DDM file. The remote location information of the DDM file is used to determine the communications line to be used. Thus, it identifies the server that is to receive the submitted command. The remote file associated with the DDM file is not involved when the DDM file is used for submitting commands to run on the server.

The Submit Remote Command (SBMRMTCMD) command can submit any CL command that can run in both the batch environment and through the QCAEXEC system program; that is, the command has values of *BPGM *and* *EXEC specified for the ALLOW attribute. You can display the ALLOW attributes by using the Display Command (DSPCMD) command.

The primary purpose of the Submit Remote Command (SBMRMTCMD) command is to allow a client user or program to perform file management operations and file authorization activities on objects located on a server. A secondary purpose of this command is to allow a user to perform nonfile operations (such as creating a message queue) or to submit user-written commands to run on the server. The CMD parameter allows you to specify a character string of up to 2000 characters that represents a command to be run on the server.

You must have the proper authority on the server for the CL command being submitted and for the objects that the command is to operate on. If the client user has the correct authority to do so (as determined in a server user profile), the following actions are examples of what can be performed on remote files using the Submit Remote Command (SBMRMTCMD) command:

- Grant or revoke object authority to remote tables
- Verify tables or other objects
- Save or restore tables or other objects

Although the command can be used to do many things with tables or other objects on the remote system, using this command for some tasks is not as efficient as other methods on the system. For example, you can use this command to display the file descriptions or field attributes of remote files, or to dump files or other objects, but the output remains at the server. To display remote file descriptions and field attributes at the client, a better method is to use the Display File Description (DSPFD) and Display File Field Description (DSPFFD) commands with SYSTEM(*RMT) specified, and specify the names of the DDM files associated with the remote files.

Related concepts

Scheduling a system shutdown and restart

System values that control IPL

Related tasks

Setting up DDM files
The IBM i implementation of Distributed Relational Database Architecture (DRDA) support uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Related reference

Controlling DDM conversations
Normally, the DDM conversations associated with a client system job are kept active until one of the conditions described in this topic is met.

Display Command (DSPCMD) command

Display File Description (DSPFD) command

Display File Field Description (DSPFFD) command

Submit Remote Command (SBMRMTCMD) command

Displaying objects used by programs

You can use the **`Display Program References (DSPPGMREF)`** command to determine which tables, data areas, and other programs are used by a program or SQL package. This information is only available for SQL packages and compiled programs and can be displayed, printed, or written to a database output file.

When a program or package is created, the information about certain objects used in the program or package is stored. This information is then available for use with the **`Display Program References (DSPPGMREF)`** command. Information retrieved can include:

- The name of the program or package and its text description
- The name of the library or collection containing the program or package
- The number of objects referred to by the program package
- The qualified name of the system object
- The information retrieval dates
- The object type of the referenced object

For files and tables, the record contains the following additional fields:

- The name of the file or table in the program or package (possibly different from the system object name if an override operation was in effect when the program or package was created)

  Note: Any overrides apply only on the application requester (AR).
- The program or package use of the file or table (input, output, update, unspecified, or a combination of these four)
- The number of record formats referenced, if any
- The name of the record format used by the file or table and its record format level identifier
- The number of fields referenced for each format

Before the objects can be shown in a program, the user must have *USE authority for the program. Also, of the libraries specified by the library qualifier, only the libraries for which the user has read authority are searched for the programs.

The following table shows the objects for which the high-level languages and utilities save information.

Table 28. How high-level languages save information about objects

Language	Files	Programs	Data areas	See note
CL	Yes	Yes	Yes	1
COBOL/400 Language	Yes	Yes	No	2
PL/I	Yes	Yes	N/A	2
RPG/400 Language	Yes	No	Yes	3
Db2 for i SQL	Yes	N/A	N/A	4

Table 28. How high-level languages save information about objects (continued)

Language	Files	Programs	Data areas	See note

Notes:

1. All commands that refer to files, programs, or data areas specify in the command definition that the information should be stored when the command is compiled in a CL program. If a variable is used, the name of the variable is used as the object name (for example, &FILE). If an expression is used, the name of the object is stored as *EXPR. User-defined commands can also store the information for files, programs, or data areas specified on the commands. See the description of the FILE, PGM, and DTAARA parameters on the PARM or ELEM command statements.
2. The program name is stored only when a literal is used for the program name (this is a static call, for example, `CALL 'PGM1'`), not when a COBOL/400 identifier is used for the program name (this is a dynamic call, for example, `CALL PGM1`).
3. The use of the local data area is not stored.
4. Information about SQL packages.

The stored file information contains an entry (a number) for the type of use. In the database file output of the **`Display Program References (DSPPGMREF)`** command (built when using the OUTFILE parameter), this entry is a representation of one or more codes listed here. There can only be one entry per object, so combinations are used. For example, a file coded as a 7 would be used for input, output, and update.

Code	Meaning
1	Input
2	Output
3	Input and Output
4	Update
8	Unspecified

Related reference

Display Program References (DSPPGMREF) command

Element definition (ELEM)

Example: Displaying program reference

To see what objects are used by an application requester (AR) program, you can enter a command as follows.

```
DSPPGMREF PGM(SPIFFY/PARTS1) OBJTYPE(*PGM)
```

On the requester you can get a list of all the collections and tables used by a program, but you are not able to see on which relational database they are located. They might be located in multiple relational databases. The output from the command can go to a database file or to a displayed spooled file. The output looks like this:

```
File  . . . . . :   QPDSPPGM                         Page/Line   1/1
Control . . . . .                                    Columns     1 - 78
Find  . . . . . .

3/29/92               Display Program References
DSPPGMREF Command Input
Program . . . . . . . . . . . . . . . . . . . :   PARTS1
```

```
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Output  . . . . . . . . . . . . . . . . . . :   *
Include SQL packages  . . . . . . . . . . . :   *YES
Program . . . . . . . . . . . . . . . . . . :   PARTS1
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Text 'description'. . . . . . . . . . . . . :   Check inventory for parts
Number of objects referenced  . . . . . . . :   3
Object  . . . . . . . . . . . . . . . . . . :   PARTS1
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Object type . . . . . . . . . . . . . . . . :   *PGM
Object  . . . . . . . . . . . . . . . . . . :   QSQROUTE
Library . . . . . . . . . . . . . . . . . . :   *LIBL
Object type . . . . . . . . . . . . . . . . :   *PGM
Object  . . . . . . . . . . . . . . . . . . :   INVENT
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Object type . . . . . . . . . . . . . . . . :   *FILE
File name in program  . . . . . . . . . . . :
File usage  . . . . . . . . . . . . . . . . :   Input
```

To see what objects are used by a server SQL package, you can enter a command as follows:

```
DSPPGMREF PGM(SPIFFY/PARTS1) OBJTYPE(*SQLPKG)
```

The output from the command can go to a database file or to a displayed spooled file. The output looks like this:

```
File  . . . . . :   QPDSPPGM                         Page/Line   1/1
Control . . . . .                                    Columns     1 - 78
Find  . . . . . .

3/29/92               Display Program References
DSPPGMREF Command Input
Program . . . . . . . . . . . . . . . . . . :   PARTS1
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Output  . . . . . . . . . . . . . . . . . . :   *
Include SQL packages  . . . . . . . . . . . :   *YES
SQL package . . . . . . . . . . . . . . . . :   PARTS1
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Text 'description'. . . . . . . . . . . . . :   Check inventory for parts
Number of objects referenced  . . . . . . . :   1
Object  . . . . . . . . . . . . . . . . . . :   INVENT
Library . . . . . . . . . . . . . . . . . . :   SPIFFY
Object type . . . . . . . . . . . . . . . . :   *FILE
File name in program  . . . . . . . . . . . :
File usage  . . . . . . . . . . . . . . . . :   Input
```

Dropping a collection from a distributed relational database

Attempting to delete a collection that contains journal receivers might cause an inquiry message to be sent to the QSYSOPR message queue for the server job. The server and client jobs wait until this inquiry is answered.

The message that appears on the message queue is:

CPA7025

Receiver (*name*) in (*library*) never fully saved. (I C)

When the client job is waiting, it might appear as if the application is hung. Consider the following items when your client job has been waiting for a time longer than anticipated:

- Be aware that an inquiry message is sent to QSYSOPR message queue and needs an answer to proceed.
- Have the server reply to the message using its server reply list.

Note: When the application is in this apparent endless-loop state, the application requesting job will wait until the inquiry message on the system has been answered. This is because journal receivers cannot be moved to another library by using the Move Object (MOVOBJ) command. They also cannot be saved and restored to different libraries. All you can do is to create a new journal receiver in a different library, using the Create Journal Receiver (CRTJRNRCV) command, and attach it to the journal, using the Change Journal (CHGJRN) command. Any new journal receivers that are created by the system, using the Change Journal (CHGJRN) command with the JRNRCV(*GEN) parameter, are created in the new library. If, when the journal is saved, the attached receiver is in another library, then when the saved version of the journal is restored, the new journal receivers are also created in the other library.

Having the server reply to the message using its server reply list can be accomplished by changing the job that appears to be currently hung, or by changing the job description for all server jobs running on the system. However, you must first add an entry to the server reply list for message CPA7025 using the Add Reply List Entry (ADDRPYLE) command:

```
ADDRPYLE  SEQNBR(...)  MSGID(CPA7025) RPY(I)
```

To change the job description for the job that is currently running on the server, use the Submit Remote Command (SBMRMTCMD) command. The following example shows how the database administrator on one system in the Kansas City region changes the job description on the KC105 system (the system addressed by the TEST/KC105TST DDM file):

```
SBMRMTCMD  CMD('CHGJOB JOB(KC105ASJOB) INQMSGRPY(*SYSRPYL)')
   DDMFILE(TEST/KC105TST)
```

You can prevent this situation from happening on the server more permanently by using the Change Job Description (CHGJOBD) command so that any job that uses that job description uses the server reply list. The following example shows how this command is entered on the same server:

```
CHGJOBD JOBD(KC105ASJOB)  INQMSGRPY(*SYSRPYL)
```

This method should be used with caution. Adding CPA7025 to the server reply list affects all jobs which use the server reply list. Also changing the job description affects all jobs that use a particular job description. You might want to create a separate job description for server jobs.

Related concepts
Journal management
Managing work
Related reference
Add Reply List Entry (ADDRPYLE) command
Change Job Description (CHGJOBD) command
Change Journal (CHGJRN) command
Create Journal Receiver (CRTJRNRCV) command
Move Object (MOVOBJ) command
Submit Remote Command (SBMRMTCMD) command

Job accounting in a distributed relational database

The IBM i job accounting function gathers data so you can determine who is using the system and what system resources they are using.

Typical job accounting provides details on the jobs running on a system and resources used, such as use of the processing unit, printer, display stations, and database and communications functions. Job accounting is optional and must be set up on the system.

To set up resource accounting on the system, follow these steps:

1. Create a journal receiver by using the **Create Journal Receiver (CRTJRNRCV)** command.
2. Create the journal named QSYS/QACGJRN by using the **Create Journal (CRTJRN)** command. You must use the name QSYS/QACGJRN and you must have authority to add items to QSYS to create this journal. Specify the names of the journal receiver you created in the previous step on this command.
3. Change the accounting level system value QACGLVL using the **Work with System Values (WRKSYSVAL)** or **Change System Value (CHGSYSVAL)** command.

   The VALUE parameter on the **Change System Value (CHGSYSVAL)** command determines when job accounting journal entries are produced. A value of *NONE means the system does not produce any entries in the job accounting journal. A value of *JOB means the system produces a job (JB) journal entry. A value of *PRINT produces a direct print (DP) or spooled print (SP) journal entry for each file printed.

When a job is started, a job description is assigned to the job. The job description object contains a value for the accounting code (ACGCDE) parameter, which can be an accounting code or the default value *USRPRF. If *USRPRF is specified, the accounting code in the job's user profile is used.

You can add accounting codes to user profiles using the accounting code parameter ACGCDE on the **Create User Profile (CRTUSRPRF)** command or the **Change User Profile (CHGUSRPRF)** command. You can change accounting codes for specific job descriptions by specifying the desired accounting code for the ACGCDE parameter on the **Create Job Description (CRTJOBD)** command or the **Change Job Description (CHGJOBD)** command.

When a job accounting journal is set up, job accounting entries are placed in the journal receiver starting with the next job that enters the system after the **Change System Value (CHGSYSVAL)** command takes effect.

You can use the OUTFILE parameter on the **Display Journal (DSPJRN)** command to write the accounting entries to a database file that you can process.

Related concepts

Managing work

Managing the TCP/IP server
The DRDA and DDM TCP/IP server does not typically require any changes to your existing system configuration. At some time, you might want to change the way the system manages the server jobs to better meet your needs, to solve a problem, to improve the system performance, or to look at the jobs on the system.

Related reference

Accounting for a distributed relational database
You need to be able to account and charge for the use of distributed data.

Change Job Description (CHGJOBD) command

Change System Value (CHGSYSVAL) command

Change User Profile (CHGUSRPRF) command

Create Job Description (CRTJOBD) command

Create Journal Receiver (CRTJRNRCV) command

Create Journal (CRTJRN) command

Create User Profile (CRTUSRPRF) command

Display Journal (DSPJRN) command

Work with System Value (WRKSYSVAL) command

Managing the TCP/IP server

The DRDA and DDM TCP/IP server does not typically require any changes to your existing system configuration. At some time, you might want to change the way the system manages the server jobs to better meet your needs, to solve a problem, to improve the system performance, or to look at the jobs on the system.

To make such changes and meet your processing requirements, you need to know which objects affect which pieces of the system and how to change those objects. This topic collection describes how to manage the DRDA and DDM server jobs that communicate by using sockets over TCP. It describes the subsystem in which the system runs, the objects that affect the system, and how to manage those resources.

To fully understand how to manage your IBM i product, it is suggested that you carefully review the Work management topic collection before you continue with this topic collection. This topic collection describes, at a high level, some of the work management concepts that you need to understand to work with the server jobs, and how the concepts and objects relate to the system. This topic collection then shows you how the TCP/IP server can be managed and how they fit in with the rest of the system.

Related concepts

IBM i work management
All of the work on the IBM i operating system is submitted through the work management function. On the system, you can design specialized operating environments to handle different types of work to satisfy your system requirements.

Managing work

Related tasks

Setting up the TCP/IP server
If you own a Distributed Relational Database Architecture (DRDA) server that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Related reference

Job accounting in a distributed relational database
The IBM i job accounting function gathers data so you can determine who is using the system and what system resources they are using.

Change Job Description (CHGJOBD) command

Change System Value (CHGSYSVAL) command

Change User Profile (CHGUSRPRF) command

Create Job Description (CRTJOBD) command

Create Journal Receiver (CRTJRNRCV) command

Create Journal (CRTJRN) command

Create User Profile (CRTUSRPRF) command

Display Journal (DSPJRN) command

Work with System Value (WRKSYSVAL) command

DDM file creation using TCP/IP
You can create a DDM file that uses TCP/IP as the communication protocol for connecting with the remote system.

TCP/IP server terminology

The same software is used for both DDM and DRDA TCP/IP access to Db2 for i.

For brevity, the term *DDM server* is used rather than *DRDA and DDM server* in the following discussion. Sometimes, however, it might be referred to as the *TCP/IP server*, the *DRDA server*, or the *server* when the context makes the use of a qualifier unnecessary.

The DDM server consists of two or more jobs, one of which is what is called the DDM *listener*, because it listens for connection requests and dispatches work to the other jobs. The other job or jobs, as initially configured, are prestart jobs which service requests from the DRDA or DDM client after the initial connection is made. The set of all associated jobs, the listener and the server jobs, are collectively referred to as the *DDM server*.

The term *client* is used interchangeably with *DRDA application requester* (or AR) in the DRDA application environment. The term client will be used interchangeably with *DDM source system* in the DDM (distributed file management) application environment.

The term *server* is used interchangeably with *DRDA application server* (or AS) in the DRDA application environment. The term server will be used interchangeably with *DDM target system* in the DDM application environment. (Note that in some contexts, the IBM i product (the hardware) is also called a server.)

Establishing a connection over TCP/IP

To initiate an IBM i DRDA Application Server or DDM server job that uses TCP/IP communications support, the DRDA Application Requester or DDM client system, depending on the security configured, must connect to one of the following well-known port numbers: 446, 447, or 448.

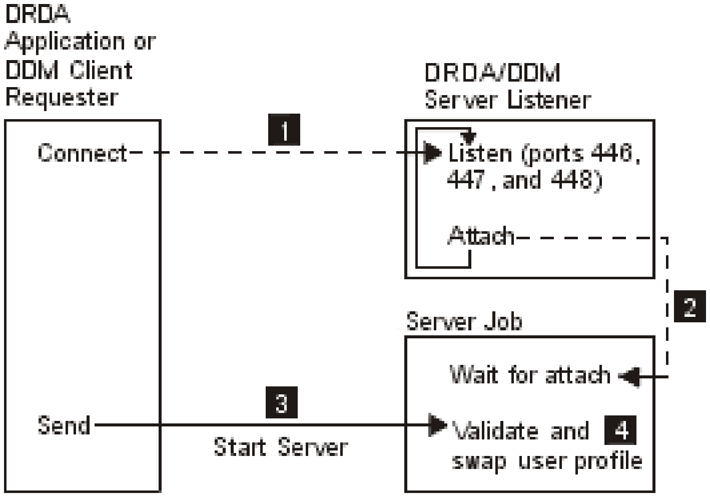


Figure 28. DRDA and DDM TCP/IP server

1. The DDM listener program must have been started (by using the Start TCP/IP Server (STRTCPSVR SERVER(*DDM)) to listen for and accept the client's connection request.

The DDM listener, on accepting this connection request, will issue an internal request to attach the client's connection to a DDM server job 2. This server job might be a prestarted job or, if the user has removed the QRWTSRVR prestart job entry from the QUSRSYS or user-defined subsystem (in which case prestart jobs are not used), a batch job that is submitted when the client connection request is processed. The server job will handle any further communications with the client.

The initial data exchange that occurs includes a request that identifies the user profile under which the server job is to run 3.

After the user profile and password (if it is sent with the user profile ID) have been validated, the server job will swap to this user profile as well as change the job to use the attributes, such as CCSID, defined for the user profile 4.

The functions of connecting to the listener program, attaching the client connection to a server job and exchanging data and validating the user profile and password are comparable to those performed when an APPC program start request is processed.

Related reference

Start TCP/IP Server (STRTCPSVR) command

The listener program

The DDM listener program runs in a batch job. There is a one-to-many relationship between it and the actual server jobs; there is one listener and potentially many DDM server jobs. The server jobs are normally prestart jobs. The listener job runs in the QSYSWRK subsystem.

The DDM listener allows client applications to establish TCP/IP connections with an associated server job by handling and routing inbound connection requests. After the client has established communications with the server job, there is no further association between the client and the listener for the duration of that connection.

The DDM listener must be active in order for DRDA application requesters and DDM client systems to establish connections with the DDM TCP/IP server. You can request that the DRDA listener be started automatically by either using the Change DDM TCP/IP Attributes (CHGDDMTCPA) command or through System i Navigator. From System i Navigator, navigate to the DDM settings: **Network** > **Servers** > **TCP/IP**. This will cause the listener to be started when TCP/IP is started. When starting the DRDA listener, both the QSYSWRK subsystem and TCP/IP must be active.

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Start TCP/IP Server (STRTCPSVR) CL command

The **Start TCP/IP Server (STRTCPSVR)** command, with a SERVER parameter value of *DDM or *ALL, is used to start the listener.

Related reference

Start TCP/IP Server (STRTCPSVR) command

Listener restrictions

Only one DDM listener can be active at one time. Requests to start the listener when it is already active will result in an informational message to the command issuer.

Note: The DDM server will not start if the QUSER password has expired. It is recommended that the password expiration interval be set to *NOMAX for the QUSER profile. With this value the password will not expire.

Examples: Starting TCP/IP Server

This topic contains some examples of the **Start TCP/IP Server** (STRTCPSVR) CL command.

Example: Starting all TCP/IP servers

```
STRTCPSVR SERVER(*ALL)
```

This command starts all of the TCP/IP servers, including the DDM server.

Example: Starting just the DDM TCP/IP server

```
STRTCPSVR *DDM
```

This command starts only the DDM TCP/IP server.

End TCP/IP Server (ENDTCPSVR) CL command

The **End TCP/IP Server (ENDTCPSVR)** command ends the DDM server.

If the DDM listener is ended, and there are associated server jobs that have active connections to client applications, the server jobs will remain active until communication with the client application is ended. Subsequent connection requests from the client application will fail, however, until the listener is started again.

Related reference

End TCP/IP Server (ENDTCPSVR) command

End TCP/IP server restrictions

If the **End TCP/IP Server (ENDTCPSVR)** command is used to end the DDM listener when it is not active, a diagnostic message will be issued. This same diagnostic message will not be sent if the listener is not active when an (ENDTCPSVR) SERVER(*ALL) command is issued.

Related reference

End TCP/IP Server (ENDTCPSVR) command

Example: Ending TCP/IP server

This topic contains some examples of ending TCP/IP servers.

Example: Ending all TCP/IP servers

```
ENDTCPSVR *ALL
```

This command ends all active TCP/IP servers.

Example: Ending just the DDM server

```
ENDTCPSVR SERVER(*DDM)
```

This command ends the DDM server.

Starting the listener in System i Navigator

The DDM listener can also be administered using System i Navigator, which is part of the IBM i Access Family licensed program.

This can be done by following the path **Network** > **Servers** > **TCP/IP directory**.

The server jobs

These topics discuss DRDA and DDM server jobs.

Subsystem descriptions and prestart job entries with DDM

A subsystem description defines how, where, and how much work enters a subsystem, and which resources the subsystem uses to perform the work. The following describes how the prestart job entries in the QUSRWRK (or QSYSWRK prior to OS/400 V5R2) subsystem description affect the DDM server.

A prestart job is a batch job that starts running before an application requester (AR) initiates communications with the server. Prestart jobs use prestart job entries in the subsystem description to determine which program, class, and storage pool to use when the jobs are started. Within a prestart job entry, you must specify attributes that the subsystem uses to create and manage a pool of prestart jobs.

Prestart jobs provide increased performance when initiating a connection to a server. Prestart job entries are defined within a subsystem. Prestart jobs become active when that subsystem is started, or they can be controlled with the **Start Prestart Jobs (STRPJ)** and **End Prestart Jobs (ENDPJ)** commands.

Related reference

End Prestart Jobs (ENDPJ) command

Start Prestart Jobs (STRPJ) command

Prestart jobs

Information about prestart jobs, such as the **Display Active Prestart Jobs (DSPACTPJ)** command, uses the term *program start request* exclusively to indicate requests made to start prestart jobs, even though the information might pertain to a prestart job that was started as a result of a TCP/IP connection request.

The following list contains the prestart job entry attributes with the initially configured value for the DDM TCP/IP server. They can be changed with the **Change Prestart Job Entry (CHGPJE)** command.

- Subsystem description. The subsystem that contains the prestart job entries is QUSRWRK in V5R2. In earlier releases, it was QSYSWRK.
- Program library and name. The program that is called when the prestart job is started is QSYS/QRWTSRVR.
- User profile. The user profile that the job runs under is QUSER. This is what the job shows as the user profile. When a request to connect to the server is received from a client, the prestart job function swaps to the user profile that is received in that request.
- Job name. The name of the job when it is started is QRWTSRVR.
- Job description. The job description used for the prestart job is *USRPRF. Note that the user profile is QUSER so this will be whatever QUSER's job description is. However, the attributes of the job are changed to correspond to the requesting user's job description after the user ID and password (if present) are verified.
- Start jobs. This indicates whether prestart jobs are to automatically start when the subsystem is started. These prestart job entries are shipped with a start jobs value of *YES. You can change these to *NO to prevent unnecessary jobs starting when a system IPL is performed.

Note: If the DDM server jobs are not running and the DDM listener job is batch, immediate DDM server jobs will still be run under the QSYSWRK subsystem.

- Initial number of jobs. As initially configured, the number of jobs that are started when the subsystem is started is 1. This value can be adjusted to suit your particular environment and needs.
- Threshold. The minimum number of available prestart jobs for a prestart job entry is set to 1. When this threshold is reached, additional prestart jobs are automatically started. This is used to maintain a certain number of jobs in the pool.
- Additional number of jobs. The number of additional prestart jobs that are started when the threshold is reached is initially configured at 2.
- Maximum number of jobs. The maximum number of prestart jobs that can be active for this entry is *NOMAX.
- Maximum number of uses. The maximum number of uses of the job is set to 200. This value indicates that the prestart job ends after 200 requests to start the server are processed. In certain situations, you might need to set the MAXUSE parameter to 1 in order for the TCP/IP server to function properly. When the server runs certain ILE stored procedures, pointers to destroyed objects might remain in the prestart job environment; subsequent uses of the prestart job might cause MCH3402 exceptions. IBM i minimizes this possibility.
- Wait for job. The *YES setting causes a client connection request to wait for an available server job if the maximum number of jobs is reached.
- Pool identifier. The subsystem pool identifier in which this prestart job runs is set to 1.
- Class. The name and library of the class the prestart jobs will run under is set to QSYS/QSYSCLS20.

When the start jobs value for the prestart job entry has been set to *YES, and the remaining values are as provided with their initial settings, the following events happen for each prestart job entry:

- When the subsystem is started, one prestart job is started.
- When the first client connection request is processed for the TCP/IP server, the initial job is used and the threshold is exceeded.
- Additional jobs are started for the server based on the number defined in the prestart job entry.
- The number of available jobs will not reach below 1.
- The subsystem periodically checks the number of prestart jobs in a pool that are unused and ends excess jobs. It always leaves at least the number of prestart jobs specified in the initial jobs parameter.

Related tasks

Configuring the server job subsystem
By default, the DDM TCP/IP server jobs run in the QUSRWRK subsystem. Using System i Navigator, you can configure DDM server jobs to run all or certain server jobs in alternate subsystems based on the client's IP address. Alternatively, the QSYS2.SET_SERVER_SBS_ROUTING procedure provides the ability to configure some servers to utilize alternate subsystems based on the user profile that is establishing the connection.

Related reference

Change Prestart Job Entry (CHGPJE) command

Display Active Prestart Jobs (DSPACTPJ) command

Monitoring prestart jobs
Prestart jobs can be monitored by using the **Display Active Prestart Jobs (DSPACTPJ)** command.

The **DSPACTPJ** command provides the following information:

- Current number of prestart jobs
- Average number of prestart jobs
- Peak number of prestart jobs
- Current number of prestart jobs in use

- Average number of prestart jobs in use
- Peak number of prestart jobs in use
- Current number of waiting connect requests
- Average number of waiting connect requests
- Peak number of waiting connect requests
- Average wait time
- Number of connect requests accepted
- Number of connect requests rejected

Related reference

Display Active Prestart Jobs (DSPACTPJ) command

Managing prestart jobs

The information presented for an active prestart job can be refreshed by pressing the F5 key while on the Display Active Prestart Jobs display.

Of particular interest is the information about program start requests. This information can indicate to you whether you need to change the available number of prestart jobs. If you have information indicating that program start requests are waiting for an available prestart job, you can change prestart jobs using the **Change Prestart Job Entry (CHGPJE)** command.

If the program start requests were not being acted on fast enough, you can do any combination of the following things:

- Increase the threshold.
- Increase the Initial number of jobs (INLJOBS) parameter value.
- Increase the Additional number of jobs (ADLJOBS) parameter value.

The key is to ensure that there is an available prestart job for every sent request that starts a server job.

Related reference

Change Prestart Job Entry (CHGPJE) command

Removing prestart job entries

If you do not want to use the prestart job function, remove prestart job entries from the subsystem description.

1. End the prestarted jobs using the **End Prestart Jobs (ENDPJ)** command.

   Prestarted jobs ended with the **ENDPJ** command will be started the next time the subsystem is started if start jobs *YES is specified in the prestart job entry. If you only end the prestart job and do not perform the next step, any requests to start the particular server will fail.
2. Remove the prestart job entries in the subsystem description using the **Remove Prestart Job Entry (RMVPJE)** command.

   The prestart job entries removed with the **RMVPJE** command are permanently removed from the subsystem description. After the entry is removed, new requests for the system are successful, but incur the performance overhead of job initiation.

Related reference

End Prestart Jobs (ENDPJ) command

Remove Prestart Job Entry (RMVPJE) command

Routing entries

An IBM i job is routed to a subsystem by using the routing entries in the subsystem description. The routing entry for the listener job in the QSYSWRK subsystem is present after IBM i is installed. This job is started under the QUSER user profile, and the QSYSNOMAX job queue is used.

Prior to V5R2, the server jobs ran in the QSYSWRK subsystem. In V5R2, the server jobs run by default in QUSRWRK. The characteristics of the server jobs are taken from their prestart job entry which also comes automatically configured with IBM i. If this entry is removed so that prestart jobs are not used for the servers, then the server jobs are started using the characteristics of their corresponding listener job.

The following list provides the initial configuration in the QSYSWRK subsystem for the listener job.

Subsystem
QSYSWRK

Job Queue
QSYSNOMAX

User
QUSER

Routing Data
QRWTLSTN

Job Name
QRWTLSTN

Class
QSYSCLS20

Configuring the server job subsystem

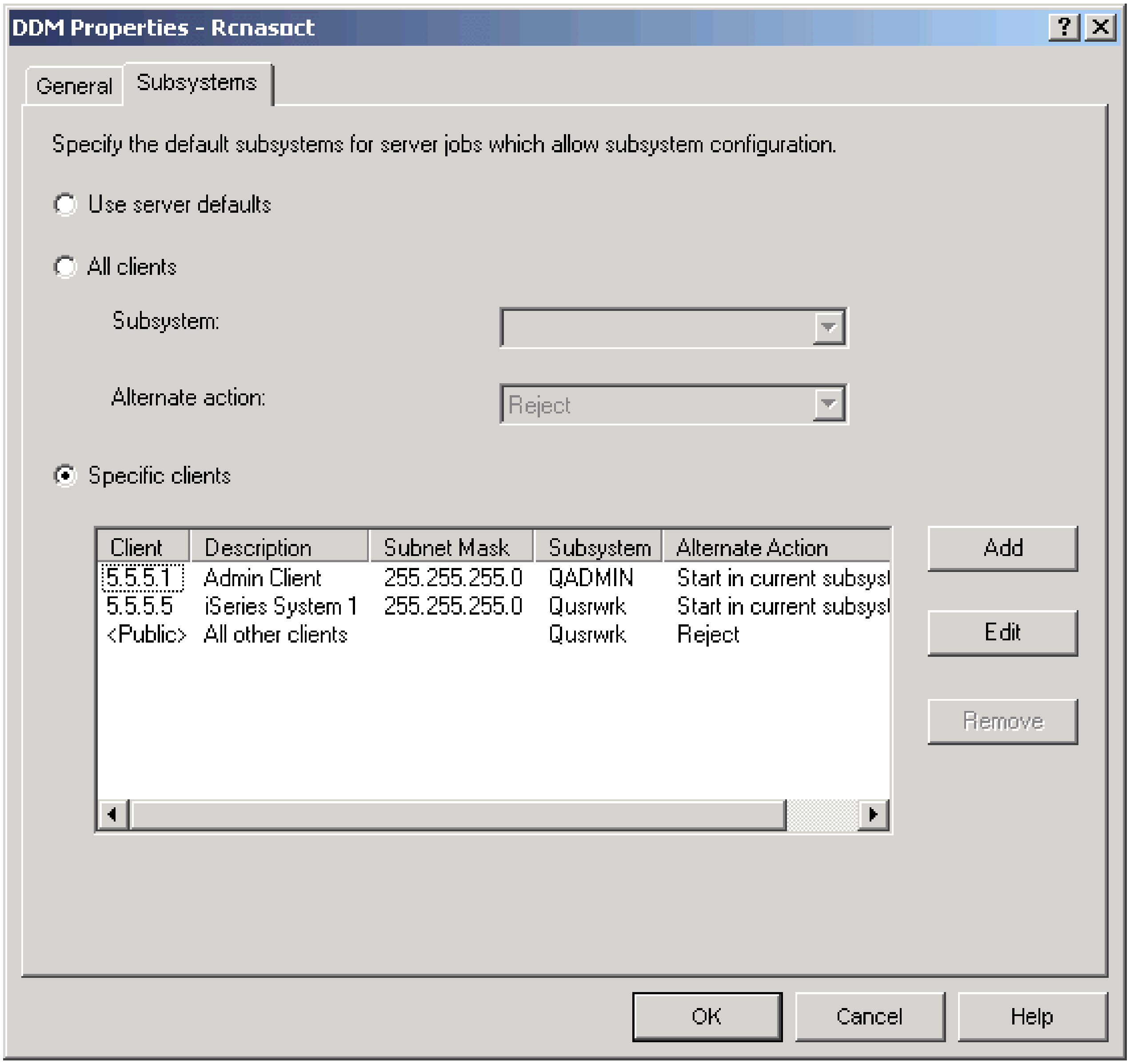
By default, the DDM TCP/IP server jobs run in the QUSRWRK subsystem. Using System i Navigator, you can configure DDM server jobs to run all or certain server jobs in alternate subsystems based on the client's IP address. Alternatively, the QSYS2.SET_SERVER_SBS_ROUTING procedure provides the ability to configure some servers to utilize alternate subsystems based on the user profile that is establishing the connection.

To set up the configuration based on the connecting user profile, see QSYS2.SET_SERVER_SBS_ROUTING procedure in the Database Performance and Query Optimization book.

To set up the configuration based on client's IP address:

1. Create a prestart job entry for each desired subsystem with the **`Add Prestart Job Entry (ADDPJE)`** command.
2. Start the prestart job entry you created with the **`Start Prestart Jobs (STRPJ)`** command.
3. From System i Navigator, expand **Network**.
4. Expand **Servers**.
5. Click **TCP/IP**.
6. Right-click **DDM** in the list of servers that are displayed in the right panel and select **Properties**.
7. On the **Subsystems** tab, add the specific client and the name of the subsystems.

In the following example, the administrator could connect and run in the QADMIN subsystem, while another server in the network could connect and run in QUSRWRK. All other clients would be rejected.



Specify the default subsystems for server jobs which allow subsystem configuration.

○ Use server defaults

○ All clients

Subsystem:

Alternate action: Reject

● Specific clients

Client	Description	Subnet Mask	Subsystem	Alternate Action
5.5.5.1	Admin Client	255.255.255.0	QADMIN	Start in current subsyst
5.5.5.5	iSeries System 1	255.255.255.0	Qusrwrk	Start in current subsyst
<Public>	All other clients		Qusrwrk	Reject

Related reference

Add Prestart Job Entry (ADDPJE) command

Start Prestart Jobs (STRPJ) command

Prestart jobs

Information about prestart jobs, such as the **`Display Active Prestart Jobs (DSPACTPJ)`** command, uses the term *program start request* exclusively to indicate requests made to start prestart jobs, even though the information might pertain to a prestart job that was started as a result of a TCP/IP connection request.

Identifying server jobs

Being able to identify a particular job is a prerequisite to investigating problems and gathering performance data.

If you look at the server jobs started on the system, you might find it difficult to relate a server job to a certain application requester job or to a particular PC client. System i Navigator provides support for these tasks that make the job much easier. These topics provide information about how to identify server jobs before you start debugging or investigating performance when you are not using System i Navigator.

IBM i job names

The job name used on the IBM i operating system consists of these parts.

- The simple job name
- User ID
- Job number (ascending order)

The DDM server jobs follow the following conventions:

- Job name is QRWTSRVR.
- User ID
  - Will always be QUSER, whether prestart jobs are used or not.
  - The job log will show which user is currently using the job.
- The job number is created by work management.

Displaying server jobs

These methods can be used to help identify server jobs.

One method is to use the **`Work with Active Jobs (WRKACTJOB)`** command. Another method is to use the **`Work with User Jobs (WRKUSRJOB)`** command. A third method is to display the history log to determine which job is being used by which client user.

Related reference

Work with Active Jobs (WRKACTJOB) command

Work with User Jobs (WRKUSRJOB) command

*Displaying active jobs using the **WRKACTJOB** command*

The **`Work with Active Jobs (WRKACTJOB)`** command shows all active jobs. All server jobs are displayed, as well as the listener job.

The following figures show a sample status using the (**WRKACTJOB**) command. Only jobs related to the server are shown in the figures. You must press F14 to see the available prestart jobs.

The following types of jobs are shown in the figures:

- 1 - Listener job
- 2 - Prestarted server jobs

```
                            Work with Active Jobs                     AS400597
                                                            10/16/09  10:55:40
CPU %:    41.7     Elapsed time:   04:37:55     Active jobs:    42

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...
                   Current
Opt  Subsystem/Job  User        Type  CPU %  Function        Status
__   QSYSWRK        QSYS        SBS      .0                   DEQW
          .
__ 1
       QRWTLSTN     QUSER       BCH      .0                   SELW
          .
          .
__   QUSRWRK        QSYS        SBS      .0                   DEQW
__ 2
       QRWTSRVR     QUSER       PJ       .0                   TIMW
__     QRWTSRVR     QUSER       PJ       .0                   TIMW
__     QRWTSRVR     QUSER       PJ       .0                   TIMW
__     QRWTSRVR     QUSER       PJ       .0                   TIMW
__     QRWTSRVR     QUSER       PJ       .0                   TIMW
          .                                                            More...
```

The following types of jobs are shown:

PJ
The prestarted server jobs.

SBS
The subsystem monitor jobs.

BCH
The listener job.

The **Work with Active Jobs (WRKACTJOB)** command shows all active jobs. All server jobs are displayed, as well as the listener job.

Related reference

Work with Active Jobs (WRKACTJOB) command

Displaying active user jobs using the ***WRKUSRJOB*** *command*

The **Work with User Jobs (WRKUSRJOB)** command USER(QUSER) STATUS(*ACTIVE) displays all active server jobs running under QUSER. This includes the DDM listener and all DDM server jobs. This command might be preferable, in that it will list fewer jobs for you to look through to find the DDM-related ones.

Related reference

Work with User Jobs (WRKUSRJOB) command

Displaying the history log

Each time a client user establishes a successful connection with a server job, that job is swapped to run under the profile of that client user.

To determine which job is associated with a particular client user, you can display the history log using the **Display Log (DSPLOG)** command. An example of the information provided is shown in the following figure.

```
                      Display History Log Contents
      .
      .
DDM job 036995/QUSER/QRWTSRVR servicing user MEL on 08/18/97 at 15:26:43.
      .
DDM job 036995/QUSER/QRWTSRVR servicing user REBECCA on 08/18/97 at 15:45:08.
      .
DDM job 036995/QUSER/QRWTSRVR servicing user NANCY on 08/18/97 at 15:56:21.
      .
DDM job 036995/QUSER/QRWTSRVR servicing user ROD on 08/18/97 at 16:02:59.
      .
DDM job 036995/QUSER/QRWTSRVR servicing user SMITH on 08/18/97 at 16:48:13.
      .
DDM job 036995/QUSER/QRWTSRVR servicing user DAVID on 08/18/97 at 17:10:27.
      .
      .
      .


Press Enter to continue.

F3=Exit   F10=Display all   F12=Cancel
```

The following example shows how you can filter out uninteresting entries by using the **Display Log (DSPLOG)** command with the MSGID parameter:

```
DSPLOG MSGID(CPI3E34)
```

You can also prevent these records from being written to the history log by setting the appropriate options in the QRWOPTIONS data area.

Related concepts

QRWOPTIONS data area

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related reference

Display Log (DSPLOG) command

Auditing the relational database directory

This topic discusses how to audit program access to the relational database directories.

Accesses to the relational database directory are recorded in the security auditing journal when either one of the items listed here is true.

- The value of the system QAUDLVL is *SYSMGT.
- The value of the user AUDLVL is *SYSMGT.

By using the *SYSMGT value, the system audits all accesses that were made with the following commands:

- **Add Relational Database Directory Entry (ADDRDBDIRE)** command
- **Change Relational Database Directory Entry (CHGRDBDIRE)** command
- **Display Relational Database Directory Entry (DSPRDBDIRE)** command
- **Remove Relational Database Directory Entry (RMVRDBDIRE)** command
- **Work with Relational Database Directory Entry (WRKRDBDIRE)** command

The relational database directory is a database file (QSYS/QADBXRDBD) that can be read directly without the directory entry commands.

Prior to V5R2, relational database (RDB) directory file QADBXRDBD in library QSYS was built with operational authority granted to *PUBLIC. Beginning in V5R2, that's no longer the case. Therefore, existing programs that access the RDB directory using this file might no longer run correctly. Unless you have *ALLOBJ special authority, you will have to access the logical file named QADBXRMTNM, which is built over QADBXRDBD. To audit direct accesses to this file, set auditing on with the **Change Object Auditing (CHGOBJAUD)** command.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Remove Relational Database Directory Entry (RMVRDBDIRE) command

Work with Relational Database Directory Entry (WRKRDBDIRE) command

Change Object Auditing (CHGOBJAUD) command

Operating considerations for DDM

This topic collection tells how the IBM i operating system, both as a client or server system, communicates with another IBM i to perform remote file processing. It also describes the differences when an IBM i is communicating with a system that is not an IBM i.

Note: Although this topic contains information about systems other than the IBM i, it does not contain all the information that the other server types using DDM might need to communicate with an IBM i. For complete information about how DDM is used with a particular remote system, refer to that system's documentation.

Related concepts

Planning and design for DDM

There are several requirements that must be met for distributed data management (DDM) to be used properly.

Accessing files with DDM

These topics describe the types of files supported by the IBM i operating system, when the DDM file and remote file must exist, and how to specify the names of remote files. Also included are examples and considerations for IBM i-to-IBM i and IBM i-to-System/36 file accessing.

Types of files supported by IBM i DDM

IBM i DDM supports all IBM i file types when the server system is another IBM i.

If the server system is not an IBM i, the corresponding file types might be known by different names on that server. The following table shows the IBM i equivalents of non-IBM i files and DDM architecture files:

IBM i types	Non-IBM i and DDM architecture types
Non-keyed physical file	Sequential (or direct) access file
Keyed physical file	Keyed access file
Logical file	Logical file

The following list describes the considerations that apply to the types of files supported by an IBM i.

- IBM i multiple-format logical files are not supported by DDM when the source or target system is neither an IBM i nor a System/38.
- For target physical (sequential or direct) files, if a record number is specified that is past the end of the file, the file is not extended and an error occurs.
- For target nondirect sequential files, the **Clear Physical File Member (CLRPFM)** command does not prepare a file member with deleted records.
- DDM files can be used as data files or source files by high-level language (HLL) programs. However, when a DDM file is used as a source file, the target system must be an IBM i or a System/38 and the remote file associated with the DDM file must be defined on the target system as a source file. That is, the remote file must have been created on the target IBM i or the target System/38 as FILETYPE(*SRC) by the **Create Physical File (CRTPF)** command or with FMTOPT(*CVTSRC) specified on the **Copy File (CPYF)** command.

Existence of DDM file and remote file

A file on a server system cannot be accessed for any type of operation (such as open, read, write, or display) unless a DDM file associated with the remote file already exists on the client system.

However, the remote file does not need to exist at the time that the DDM file is created or changed using the **Create DDM File (CRTDDMF)** command or the **Change DDM File (CHGDDMF)** command, because the remote file is not referred to until the DDM file is actually opened for access.

Rules for specifying server system file names for DDM

The rules for specifying the name of a DDM file (on the local IBM i) are the same as for any other file type on an IBM i. The rules, however, for specifying the name of a remote file depend on the type of server system.

A remote file name can be specified only on the RMTFILE parameter of the **Create DDM File (CRTDDMF)** and **Change DDM File (CHGDDMF)** commands. The following list is the maximum number of characters that can be used on the RMTFILE parameter to specify a remote file name:

- For the IBM i (database management): 33 characters. This maximum can occur when a full name is specified that includes a library qualifier and a member name. For example:

  ```
  LIBRARY123/FILE123456(MEMBER1234)
  ```

  The value DM can be added to the name to specify that this is a data management file. There can be one or more blanks between the name and DM. This is the default.

- For the IBM i (folder management services): 76 characters. This maximum can occur when a fully qualified path name (consisting of 76 characters) is specified. For example:

  ```
  /Path123/Path223/Path323/Path423/
     Path523/Path623/Path723/Path823/Path923/DOC1 FMS
  ```

  The value FMS specifies that this is a folder management object. There can be one or more blanks between the name and FMS.
- For System/38: 33 characters. This maximum can occur when a full name is specified that includes a library qualifier and a member name. For example:

  ```
  FILE123456.LIBRARY123(MEMBER1234)
  ```
- For System/36 and CICS: 8 characters. For example:

  ```
  FILE1234
  ```
- For other systems: 255 characters is the maximum length allowed by the DDM architecture. The actual maximum length and syntax are determined by the server system.

Target IBM i file names for DDM
As with local files, every IBM i remote file, library name, or member must begin with an alphabetic character (A through Z, $, #, or @) and can be followed by no more than 9 alphanumeric characters: A through Z, 0 through 9, $, #, @, _, or period (.). No name can exceed 10 characters. Blanks are not allowed.

The use of an extended name allows additional graphic characters to be included in quotation marks (”). The extended name also cannot exceed 10 characters, but quotation marks are included with the name, thereby limiting the number of graphic characters to 8. Lowercase letters remain lowercase letters. Examples of extended names are as follows:

```
”Test.Job”
”()/+=”
```

When an IBM i is the server system, the file name can be specified in various forms, as shown in the following examples.

library-name
Specifies the name of the library that contains the remote file. *LIBL causes the library list of the job on the server system to be searched for the specified file name. *CURLIB specifies the current library on the remote server.

remote-file-name
Specifies the name of a database file (physical, logical, or source file) on the IBM i target.

***NONSTD**
Specifies, for an IBM i target, that a member name is being included with the name of the remote file. The value *NONSTD *must* precede the full name, and the full name must be enclosed in single quotation marks and be in all uppercase.

Note: If you press F4 (Prompt) when on the Create DDM File or Change DDM File display, and specify the *NONSTD value with the remote file name abcde, the server converts abcde to 'ABCDE' (all uppercase) and the request is processed. However, if there is a slash or parenthesis in the remote file name, the system puts single quotation marks around the name but does not convert it to uppercase.

Therefore, if you are using the *NONSTD value for the remote file name and the target server requires uppercase file names, type the remote file name in uppercase characters even when using F4 (Prompt).

member-name
Specifies the name of the member in the remote file. The member name must be enclosed in parentheses and immediately follow the file name (with no space). If no member name is specified, then *FIRST is assumed and the first (or only) member in the file is accessed. This is the oldest (or only) member in the file.

*LAST is supported only on the **`Override with Database File (OVRDBF)`**, **`Clear Physical File Member (CLRPFM)`**, **`Initialize Physical File Member (INZPFM)`**, **`Reorganize Physical File Member (RGZPFM)`**, **`Open Database File (OPNDBF)`**, and **`Open Query File (OPNQRYF)`** commands. *LAST is the newest (or only) member in the file.

The examples here show valid IBM i remote file names:

```
   CUSTMAST
   PRODLIB/CUSTMAST
   *NONSTD 'CUSTMAST(MBR1)'
   *NONSTD '*LIBL/CUSTMAST(MBR2)'
   *NONSTD 'PRODLIB/CUSTMAST(MBR3) DM'
   *NONSTD 'PRODLIB/CUSTMAST(*FIRST)'
```

Target non-IBM i file names for DDM

For non-IBM i remote file names, the name must be in the form required by the server system.

If special characters are used in the remote file name, *NONSTD and single quotation marks must be used to specify the name, as shown in how to specify an IBM i member name. If the name string contains no more than 10 characters and no special characters, it can be entered without the *NONSTD value and the single quotation marks.

Using location-specific file names for commonly named files for DDM

When multiple systems are involved in a network, naming DDM files with location-specific file names can reduce confusion about which server system is being accessed for a file that has a commonly used name.

For example, for an inventory file that is named INVEN on multiple systems, using location-specific names such as NYCINVEN, STLINVEN, and DALINVEN for the DDM files on the local system to access files in New York City, St. Louis, and Dallas helps you to access the correct file.

Using an abbreviation or code that identifies the destination server system as part of the DDM file names makes it easier to remember where the desired remote file is located.

For non-IBM i remote files that have record formats, using the same name for the DDM file as for the record format can also be useful.

Examples: Accessing IBM i DDM remote files (IBM i-to-IBM i)

The examples show how access to a DDM file becomes an indirect reference (by using DDM) to the actual file on some other system. These examples are IBM i-to-IBM i examples.

Note: All of these examples assume that the DDM file on the local IBM i operating system is named DDMLIB/RMTCAR and that it is associated with a remote file named SALES/CAR on the system in Chicago.

Create a DDM file to access a remote file

```
CRTDDMF   FILE(DDMLIB/RMTCAR)  RMTFILE(SALES/CAR)
   RMTLOCNAME(CHICAGO)  TEXT('Chicago file SALES/CAR')
```

This command creates a DDM file named RMTCAR and stores it in the DDMLIB library on the local system. The remote file to be accessed is the CAR database file in the SALES library on the Chicago system. (The remote file is not accessed at the time the Create DDM File (CRTDDMF) command is used to create the DDM file. The existence of the file SALES/CAR is not checked when the DDM file is created.) Later, when the DDM file is accessed by a local program, the remote location CHICAGO is used by DDM to access the SALES/CAR file on the Chicago system.

Copy a local file to a remote file

```
CPYF   FROMFILE(QGPL/AUTO)  TOFILE(DDMLIB/RMTCAR)
```

This command copies the data from the AUTO file in the QGPL library on the local system into a remote file named SALES/CAR on the Chicago system, by using the DDM file DDMLIB/RMTCAR.

Allocate a remote file and member for use

```
ALCOBJ   OBJ((DDMLIB/RMTCAR  *FILE  *EXCL))
```

The Allocate Object (ALCOBJ) command is used to allocate (lock) both the DDM file (RMTCAR) on the client system and the first member of the remote file (as well as the file itself) on the server system. In effect, the command

```
ALCOBJ   OBJ((SALES/CAR  *FILE  *EXCL  *FIRST))
```

is run on the server system.

Override a local file with a DDM file

```
OVRDBF  FILE(FILEA)  TOFILE(DDMLIB/RMTCAR)
   POSITION(*RRN 3000)
```

This command overrides the database file FILEA with the DDM file RMTCAR, stored in the DDMLIB library. Both files are on the client system. The remote file that is identified in the DDM file (in this case, SALES/CAR on the Chicago system) is the file actually used by the client system program. When the remote file is opened, the first record to be accessed is record 3000.

Display records in a remote file

```
DSPPFM   FILE(DDMLIB/RMTCAR)
```

This command displays the records in the first member of the remote file SALES/CAR, which is associated with the DDM file DDMLIB/RMTCAR.

Display the object description of a DDM file

```
DSPOBJD   OBJ(DDMLIB/RMTCAR)  OBJTYPE(*FILE)
```

This command displays, on the local server, the object description of the RMTCAR DDM file. No reference is made by this command to the associated remote file on the Chicago system.

Display the file description of a DDM file

```
DSPFD  FILE(DDMLIB/RMTCAR) TYPE(*ATR) FILEATR(*DDM)
   SYSTEM(*LCL)
```

This command displays, on the client system, the file description of the DDM file named RMTCAR in the DDMLIB library. As indicated by the TYPE parameter, the attributes of the DDM file are displayed. Only the DDM file's attributes are displayed because FILEATR(*DDM) is specified.

Because SYSTEM(*LCL) is specified, the attributes of the DDM file are displayed and the remote system is not accessed. If SYSTEM(*RMT) is specified, the attributes of the associated remote file are displayed. If *RMT or *ALL is specified, the remote system is accessed to get the attributes of the remote file.

Delete a DDM file

```
DLTF  FILE(DDMLIB/RMTCAR)  SYSTEM(*LCL)
```

This command deletes the DDM file on the local system. Again, no reference is made to the associated SALES/CAR file on the Chicago system. If SYSTEM(*RMT) or SYSTEM(*FILETYPE) is specified, SALES/CAR on the Chicago system would be deleted.

Example: Accessing System/36 DDM remote files (IBM i-to-IBM i)

Of the command examples given in the previous topic (showing IBM i-to-IBM i examples), all except the first example can be coded the same way for accessing a file on a System/36.

That is, if the remote file name SALES/CAR is changed to CAR to meet the System/36 naming conventions, all the commands (except the first) can be used without change to access a remote System/36 file instead of an IBM i file.

The first example from the topic Examples: Accessing IBM i DDM remote files (IBM i-to-IBM i) is recorded here to access a remote System/36 file. Besides changing the remote file name, another parameter that should be coded is LVLCHK(*NO).

```
CRTDDMF   FILE(DDMLIB/RMTCAR)  RMTFILE(*NONSTD 'CAR')
   RMTLOCNAME(CHICAGO)  TEXT('Chicago file CAR on S/36')
   LVLCHK(*NO)
```

This command creates a DDM reference file named RMTCAR and stores it in the DDMLIB library on the local IBM i operating system. The remote file to be accessed is the CAR file on the System/36 named CHICAGO. LVLCHK(*NO) is specified to prevent level checking because the level identifiers created for the System/36 file do not match those in the program when it accesses the file.

Accessing members with DDM

Members are supported for database I/O operations only if the server system is a IBM i or a System/38 product.

Members can be locked before use, using the Allocate Object (ALCOBJ) command if the server system is a IBM i or a System/38 product.

The DDM file itself does not have members like a database file. However, if a member is identified on the client system (for example, using the Override with Database File (OVRDBF) command) and the server system is a IBM i or a System/38 product, that member name is used to identify a member in the server system's file. When the server system is neither a IBM i nor a System/38 product, and if the member name is specified as *FIRST, or in some cases *LAST, or the file name is the same as the member name, then the RMTFILE parameter values in the DDM file are sent without change. This allows file access on systems that do not support members.

In the following situations, an error message is sent to the requesting program and the function is not performed:

- The member name is other than *FIRST or in some cases *LAST
- The file name is different from the member name (for example, when the file is opened) and the server system does not support members

Example: Accessing DDM remote members (IBM i only)

These examples show how access to a DDM file becomes an indirect reference (by using DDM) to a member of a file on a remote IBM i operating system. These examples are IBM i-to-IBM i examples.

```
CRTDDMF  FILE(DDMLIB/RMTCAR)  RMTFILE(SALES/CAR)
   RMTLOCNAME(CHICAGO)
OVRDBF   FILE(FILE1)  TOFILE(DDMLIB/RMTCAR)  MBR(TEST1)
OVRDBF   FILE(FILE2)  TOFILE(DDMLIB/RMTCAR)
```

This example shows the creation of the same DDM file as in the previous examples. Then, **OVRDBF** commands are used to override two local files named FILE1 and FILE2 with the local DDM file RMTCAR. When an application program attempts to open the files, the DDM file DDMLIB/RMTCAR is opened twice instead. (FILE1 and FILE2 are not opened.)

After communications are established with the correct server system, the server system's TDDM opens the remote file SALES/CAR twice (two recursions) and opens two different (in this case) members in that file: member TEST1 and member *FIRST (the first member). This example requires only one DDM conversation and one target job because both open operations use the same DDM file and, therefore, the same location.

```
CLRPFM   FILE(DDMLIB/RMTCAR)  MBR(FRED)
```

This command clears, by using the DDM file named DDMLIB/RMTCAR, member FRED of the file SALES/CAR on the server system.

Example: DDM file that opens a specific member

A specific file member can be specified in the RMTFILE parameter, which is used only on the **Create DDM File (CRTDDMF)** and **Change DDM File (CHGDDMF)** commands, by using the *NONSTD value followed by the file, library, and member name.

This allows an application program to process a member other than the first member (*FIRST) without using file overrides. However, if the program requires redirection to more than one member, overrides should be used. Also, programs that already use overrides to specify members of local files should continue to do so, even if overrides to remote files are also used; otherwise, programs that worked locally would no longer do so. If the RMTFILE parameter contains a member name and an override with a different member name is in effect, the file open requests fails.

Note: If you press F4 (Prompt) on the Create DDM File or Change DDM File display, and specify the *NONSTD value with the remote file name abcde, the server converts abcde to 'ABCDE' (all uppercase) and the request is processed. However, if there is a slash or parenthesis in the remote file name, the system puts single quotation marks around the name but does not convert it to uppercase. Therefore, if you are using the *NONSTD value for the remote file name and the server system requires uppercase file names, type the remote file name in uppercase characters even when using F4 (Prompt).

```
CRTDDMF   FILE(DDMLIB/RMTCAR)  RMTFILE(*NONSTD
   'SALES/CAR(JULY)')  RMTLOCNAME(CHICAGO)
```

When a program opens the DDM file named RMTCAR on the client system DDMLIB library, the target IBM i operating system opens the member JULY in the file SALES/CAR.

Working with access methods for DDM

Access methods control what subsets of functions can be performed after a particular remote file is opened.

It is possible that an IBM i program, or a group of programs sharing a non-IBM i file, cannot do all the same operations using a file that is on the local IBM i operating system. For example, assume that an IBM i application program opens a keyed file with SHARE(*YES) and performs keyed I/O operations. It then calls another program that does relative record number operations using the same open data path (ODP) (because SHARE was specified). *Relative record numbers* specify the relationship between the location of a record and the beginning of a database file, member, or subfile. If the first program is redirected by an **Override with Database File (OVRDBF)** command to use a remote keyed file on a System/36 platform, this scheme no longer works. If a *keyed* access method is selected, record number operations fail. If a *record number* access method is selected, keyed operations fail.

When both client and server systems are IBM i products, access methods are not used. A potential problem exists when the server system is neither a IBM i nor a System/38 product. The combined-access access method (*COMBINED) is not supported by System/36, and probably not by any target other than a IBM i or System/38 platform.

Access intents

When a program opens a file, it must specify how it intends to work with the records in the file: read, add, update, delete, or a combination of these.

To successfully perform these operations, the job, user or both running the program must have the corresponding data authorities. The IBM i operating system does not check to make sure all data authorities exist when the file is opened, but it does check for each required data authority when the corresponding I/O operation is done using the file. The System/36 does check these data authorities at open time; therefore, a program may no longer work using a remote file on a System/36, even though the requester's data authorities to the remote file are the same as for a local file (which will work).

For example, assume that a program is used by two groups of users on the IBM i operating system to access the same local IBM i file. Group A has only *READ authority, while group B has *READ, *ADD, and *UPDATE. The program always opens the file for *READ, *ADD, and *UPDATE. But it has a *read only* logic path that is used when a member of group A calls the program. In this way, no authority exceptions are encountered, even though exceptions would be created if members of group A attempted to add or update records. Now, if this program is redirected to a remote System/36 file to which members of both user groups have the same data authorities as they had to the local IBM i file, the program might not work for members of group A. This is because the System/36 might reject requests to open the file when the requester does not have data authorities matching those specified in the access intent list accompanying the open request.

Key field updates

An IBM i program is allowed to change any part of a data record including key fields.

The exception to this is an ILE COBOL program because the ILE COBOL language does not allow key field changes. A System/36 program cannot change primary key fields in a record, regardless of the access method specified when the file is opened. Logical file key fields can be changed under some circumstances, but primary key fields can never be changed.

This means that an ILE RPG program, for example, that routinely changes key fields in a local keyed file might fail when it is redirected to a remote keyed file on a System/36 (or other system with similar restrictions). Several different errors might be returned by the DDM target, depending on the access method or access path being used when the key field change is attempted.

Deleted records

On the IBM i operating system, a record is marked as deleted by the system.

This is done either when an active record is deleted by an application or when a file is created with deleted records (for example, with the **Initialize Physical File Member (INZPFM)** command). A record that is added to a file or changed in a file is never marked as deleted, unless a subsequent delete operation is performed. On some other systems, like the System/36, a special data value in the record might be used to indicate deleted status. For example, if a record contains all hex FFs, it might be considered deleted.

This means that an IBM i application normally used to add or change records in a local file might encounter errors when attempting these operations with a remote file on a server that is neither a IBM i nor a System/38 platform. If the application happens to supply a record that is considered deleted by the target DDM server, the target might reject the add-or-change request.

Blocked record processing

If SEQONLY is used to block records sent to a remote system, the records are not sent until the block is full. If a source job is canceled before a block is sent, the records in the block are lost. If blocking is used, the user should make sure that a force end of data operation or closing the file is done before canceling the source job.

Variable-length records

IBM i DDM supports variable-length record files as defined in the DDM architecture.

You can use IBM i DDM to open variable-length record files on server systems that are not IBM i or S/38 platforms. (Initially you can open variable-length record files if you are not opening the file for updating.) For subsequent read operations, variable-length records are padded with blanks to the maximum record length of the file. Trailing blanks are removed on write operations.

In addition, the actual record length (maximum record length of file minus the number of padded blanks) is appended to the end of each record. For write operations, the actual record length is used to determine the length of the variable-length record to send to the server system. No counting of trailing blanks is necessary to determine the actual length of record data.

A variable-length record file can also be created on the IBM i server system as a result of a create file request.

Other DDM-related functions involving remote files

Besides accessing remote files for data record I/O operations, other operations related to remote files can be performed. These are briefly described in these topics.

Performing file management functions on remote systems

IBM i DDM supports creating, deleting, or renaming of files on a remote system.

The **`Submit Remote Command (SBMRMTCMD)`** command can be used to submit these types of file management commands, or other CL commands, to the server system so they can run on that server. The **`Submit Network Job (SBMNETJOB)`** command or display station pass-through can also be used, without the need for DDM.

Note: The CL commands in "Target IBM i-required file management commands" on page 171, "Member-related commands with DDM" on page 172, and "Source file commands" on page 174 do not need to be used with the **`SBMRMTCMD`** command; they can run directly on the server system by specifying a DDM file name on the CL command itself.

Locking files and members for DDM

Putting object locks on DDM files and their associated remote files requires special consideration.

`Allocate Object (ALCOBJ)` and `Deallocate Object (DLCOBJ)` commands

The **`ALCOBJ`** command locks DDM files on the client system and the associated remote files on the server systems.

When the target is an IBM i or a System/38, resulting locks on the remote files are the same as if the files were local files. When the target is neither an IBM i nor a System/38, equivalent locks are obtained, although the server system might promote the lock to a stronger lock condition than was specified on the **`ALCOBJ`** command.

Note: On server systems that are neither IBM i nor System/38, remote files are locked with the specified lock condition; remote members are locked with a minimum specified lock condition only on IBM i and System/38 server systems. (IBM i or System/38 remote files are locked with shared-read locks.)

`Work with Job (WRKJOB)` and `Work with Object Locks (WRKOBJLCK)` commands

For both the **`WRKOBJLCK`** command and menu option 12 (Work with locks, if active) of the **`WRKJOB`** command, only the locks held for the local DDM files are shown, not locks held for the remote files (or for their members).

If locked, DDM files are always locked as shared read (*SHRRD), regardless of the lock conditions used for their associated remote files or members.

Controlling DDM conversations

Normally, the DDM conversations associated with a client system job are kept active until one of the conditions described in this topic is met.

1. All the DDM files and remote files used in the conversation are closed and unlocked (deallocated).
2. No other DDM-related functions like the use of the Submit Remote Command (SBMRMTCMD) command or the Display File Description (DSPFD) command (to display remote file information) are being performed.
3. No DDM-related function has been interrupted (by a break program, for example) while running.
4. The ENDCMTCTL command was issued (if commitment control was used with a DDM file).
5. No distributed relational database architecture-related functions are being performed.
6. The activation group, in which the DDM conversation was started, ends. A DDM conversation is not dropped when the activation group ends under the following conditions:
   - The DDM conversation is scoped to the job level.
   - The commitment control of the activation group is scoped to the job level, and a unit of work is outstanding. The conversation remains until the next job level commit or rollback, or until the job ends.
7. The job or routing step ends.

If *1*, *2*, and *3* are true and the source job or activation group has not ended, the conversation is considered to be unused, that is, the conversation is kept active but no requests are being processed.

DDM conversations can be active and unused because the default value of the DDMCNV job attribute is *KEEP. This is desirable for the usual situation of a client system program accessing a remote file for multiple I/O operations. These operations are handled one at a time, as shown in Figure 14 on page 28 and explained in the text following it.

If multiple DDM requests are to be made in a job and the DDM files are being continually opened and closed in the job, *KEEP should be used to keep an unused DDM conversation active. (However, as long as one DDM file remains open or locked, *KEEP has no effect.)

For source jobs that access remote files but do not access data records in them, it might be desirable, depending on the frequency of the file accesses, to automatically drop each DDM conversation at the completion of each file-related source job request. Whether the conversation in the source job is kept active or automatically dropped during the time a conversation is unused is determined by the DDMCNV job attribute value (*KEEP or *DROP).

Regardless of the value of the DDMCNV job attribute, conversations are dropped when one of the following things occurs:

- The job ends.
- The activation group ends. A DDM conversation is not dropped when the activation group ends under the following conditions:
  - The DDM conversation is scoped to the job level.
  - The commitment control of the activation group is scoped to the job level, and a unit of work is outstanding. The conversation remains until the next job level commit or rollback, or until the job ends.
- The job initiates a Reroute Job (RRTJOB) command.

Unused conversations within an active job can also be dropped by the Reclaim DDM Conversations (RCLDDMCNV) or Reclaim Resources (RCLRSC) command. Errors, such as communications line failures, can also cause conversations to drop.

*Display DDMCNV values (**WRKJOB** command)*
To display the current value (*KEEP or *DROP) of the DDMCNV job attribute for your source job, you can use menu option 2 (Work with definition attributes) on the **Work with Job (WRKJOB)** Command display. You can also find out the value within a CL program by using the **Retrieve Job Attributes (RTVJOBA)** command.

*Change DDMCNV values (**CHGJOB**) command*
To control whether the system is to automatically reclaim (or drop) DDM conversations in a source job whenever they become unused, the system default *KEEP can be changed to *DROP by using a **Change Job (CHGJOB)** command. If the value is left as *KEEP, the **Reclaim DDM Conversations (RCLDDMCNV)** or **Reclaim Resources (RCLRSC)** command can be used at any time to drop all DDM conversations (within that job only) that currently do not have any active users.

*Reclaim DDM resources (**RCLRSC** and **RCLDDMCNV** commands)*
When an IBM i user wants to ensure that the resources for all APPC conversations (including DDM conversations) that are no longer active are returned to the system, the Reclaim Resources (RCLRSC) command can be used.

The Reclaim Distributed Data Management Conversations (RCLDDMCNV) command applies to the DDM conversations for the job on the client in which the command is entered. There is an associated server job for the DDM conversation used by the client job. The server job ends automatically when the associated DDM conversation ends. For TCP/IP conversations that end, the server job is normally a prestart job and is usually recycled rather than ended. The command allows you to reclaim unused DDM conversations without closing all open files or doing any of the other functions performed by the Reclaim Resources (RCLRSC) command.

Although the RCLDDMCNV command applies to all DDM conversations used by a job, using it does not mean that all of them will be reclaimed. A conversation is reclaimed only if it is not being actively used. If commitment control is used, a COMMIT or ROLLBACK operation might have to be done before a DDM conversation can be reclaimed.

Displaying DDM remote file information

The CL commands **`Display File Description (DSPFD)`** and **`Display File Field Description (DSPFFD)`** can be used by an IBM i client system user to display the attributes of one or more DDM files on the client system, or to display the attributes of one or more remote files on a server system.

Displaying DDM remote file records

The **`Display Physical File Member (DSPPFM)`** command can be used to display a remote file on a server system.

For performance reasons, however, whenever possible, you should use display station pass-through to sign on the remote system, and display the file directly. When display station pass-through is used, only the display images are transmitted over the communications line. When DDM is used to access the remote file, each record is transmitted separately over the line, which requires many more transmissions.

If pass-through cannot be used (for example, if the remote file is not on IBM i, System/38, or System/36, or if pass-through is not configured on your system), direct record positioning rather than relative positioning should be used whenever possible. For example, if record number 100 is being displayed and you want to see record number 200 next, that record is accessed faster if you enter 200 in the control field instead of +100. The results are the same, unless the file contains deleted records.

Coded character set identifier with DDM

Support for the national language of any country requires the proper handling of a set of characters.

A cross-system support for the management of character information is provided with the Character Data Representation Architecture (CDRA). CDRA defines the coded character set identifier (CCSID) values to identify the code points used to represent characters, and to convert these codes (character data), as needed to preserve their meanings.

Keep these considerations in mind when you are using CCSIDs with DDM:

- Data is converted to the process CCSID of the source job if both the client and server systems support CCSIDs.
- Data is not converted if one system is the IBM i operating system that supports CCSIDs and the other system is any other system that does not support CCSIDs.
- A file created on an IBM i server system by any client system that does not support CCSIDs is always created with CCSID 65535.
- The SBMRMTCMD (Submit Remote Command) command can be used to change the file CCSID on an IBM i server system by specifying the CHGPF (Change Physical File) command and the CCSID parameter.

Use of object distribution

Although DDM file names can be specified on the **`Send Network File (SNDNETF)`** and **`Receive Network File (RCVNETF)`** commands, these commands should be run, whenever possible, on the system where the data actually exists. Therefore, if both systems are IBM i operating systems and both are part of a SNADS network, *object distribution* can be used instead of DDM to transfer the data between them.

- The **SNDNETF** command should run directly on the system that contains the data being sent. If necessary, the **`Submit Remote Command (SBMRMTCMD)`** or **`Submit Network Job (SBMNETJOB)`** command can be used to submit the **SNDNETF** command to the system where the data exists.

  Note: Another way to use the **SNDNETF** command without using DDM is to run it on the server system using display station pass-through.

- The **RCVNETF** command must be run on the system where the data has been sent. If necessary, a DDM file can be referred to on the **RCVNETF** command to place the data on another system. However, if possible, you should arrange to have the data sent to the system where the data is to be used, to avoid using a DDM file.

For both sending and receiving operations, the file types of the data files must match and can only be a save file or a physical database file. If DDM is being used, however, the file being transferred cannot be a save file.

Use of object distribution with DDM

You can also use *both* SNADS (on IBM i operating systems) and DDM (on IBM i and non-IBM i) to transfer files between IBM i operating systems and systems that are not part of a SNADS network but have DDM installed.

Although a System/36 might have SNADS, it cannot be used for IBM i object distribution.

For example, if an IBM i DDM file refers to a file on a System/36, the IBM i operating system can use the **SNDNETF** command to send the file to another IBM i using object distribution. Similarly, if a file has been sent to an IBM i operating system, the **RCVNETF** command can be used to receive the file onto a System/36 using DDM.

Canceling distributed data management work

Whether you are testing an application, handling a user problem, or monitoring a particular device, there are times when you want to end work that is being done on a system.

When you are using an interactive job, you normally end the job by signing off the system. There are other ways that you can cancel or discontinue jobs on the system. They depend on what kind of a job it is and what system it is on.

End Job (ENDJOB) command

The **End Job (ENDJOB)** command ends any job.

The job can be active, on a job queue, or already ended. You can end a job immediately or by specifying a time interval so that end of job processing can occur.

Ending a source job ends the job on both the source and the target. If the application is under commitment control, all uncommitted changes are rolled back.

End Request (ENDRQS) command

The **End Request (ENDRQS)** command cancels a local or source operation (request) that is currently stopped at a breakpoint.

This means the command cancels an application requester operation or request. You can cancel a request by entering **ENDRQS** on a command line or you can select option 2 from the System Request menu.

If it cannot be processed immediately because a server function that cannot be interrupted is currently running, the command waits until interruption is allowed.

When a request is ended, an escape message is sent to the request processing program that is currently called at the request level being canceled. Request processing programs can monitor for the escape message so that cleanup processing can be done when a request is canceled. The static storage and open files are reclaimed for any program that was called by the request processing program. None of the programs called by the request processing program are notified of the cancellation, so they have no opportunity to stop processing.

⚠ **Attention:** Using the **ENDRQS** command on a source job may produce unpredictable results and can cause the loss of the connection to the target.

System/36 client and server considerations for DDM

Before an IBM i operating system can access files on a System/36 platform, Level 1.0 of the DDM architecture must be installed on the System/36. These topics contain information that applies when an IBM i is the client or server system communicating with a System/36.

DDM-related differences between IBM i and System/36 files

Because of differences between the types of files supported by IBM i and System/36, several items need to be considered when DDM is used between these two systems.

Generally, when a System/36 file is created locally (by the BLDFILE utility, for example), the System/36 user specifies such things as the type of file (S = sequential, D = direct, or I = indexed), whether records or blocks are to be allocated, how many of them are to be allocated, and how many additional times this amount can be added to the file to extend it.

Also, you can specify whether the file is to be *delete-capable* (DFILE) or not (NDFILE). In files specified as *not delete-capable*, records can be added or changed in the file, but not deleted.

Once these attributes have been specified, System/36 then creates the file and fills it with the appropriate hexadecimal characters. If a System/36 user specifies the file as:

- A *sequential* file, the entire file space is filled with hex 00 characters and the end-of-file (EOF) pointer is set to the beginning of the initial extent. If you attempt to read an empty sequential file, an EOF condition is received.
- A *direct* file that is *delete-capable*, the entire file space is filled with hex FF characters (deleted records) and the EOF pointer is set to the end of the initial extent. If you attempt to read an empty direct file that is delete-capable, a record-not-found condition is received.
- A *direct* file that is *not delete-capable*, the entire file space is filled with hex 40 characters (blank or null records) and the EOF pointer is set to the end of the initial extent. If you attempt to read an empty direct file that is not delete-capable, a blank record is returned for every record in the file until the end of the file is reached.
- An *indexed* file, it is prepared in the same manner as sequential files.

Typically, once a delete-capable file has been in use, it contains a relatively continuous set of active records with only a few deleted records, possibly an end of data marker, and then a continuous set of deleted records to the end of the file (EOF) space. This means that, unless the file is reorganized, a user can *un*delete (recover) a deleted record.

Of the three types of System/36 files, System/36 indexed files differ little from IBM i-supported logical files. If an IBM i source program is to use DDM to access the other types of files on a System/36, the application programmer should first consider the items remaining in this topic collection that relate to System/36.

System/36 client to IBM i server considerations for DDM

When System/36 is using DDM to communicate as a client system to access files on an IBM i server system, the information in this topic applies and should be considered.

- When System/36 creates a direct file on the IBM i operating system, the IBM i creates a nonkeyed physical file with the maximum number of records, and prepares them as deleted records. The DDM architecture command **`Clear File (CLRFIL)`**, when issued from a non-IBM i client system, clears and prepares the file; the CL command **`Clear Physical File Member (CLRPFM)`**, when issued by a local or remote IBM i system, does not prepare the file.
- System/36 supports a maximum of three key definitions for logical files and one key definition for keyed physical files.
- Nondelete-capable direct files cannot be created using DDM on the IBM i. In addition, IBM i does not support nondelete-capable files for all file organizations.

IBM i client to System/36 server considerations for DDM

When the IBM i operating system is using DDM to communicate as a client system to access files on a System/36 server system, the information in this topic applies and should be considered.

- Some file operations that are not rejected by an IBM i server system might be rejected by a target System/36. Examples are:
  - A delete record operation is rejected if the System/36 file is not a delete-capable file. To the IBM i source user, the rejection might appear to occur for unknown reasons.
  - Change operation that attempts to change the key in the primary index of a System/36 file is always rejected.
- In the System/36 environment, when System/36 users try to copy a delete-capable file to a file that is not delete-capable with the NOREORG parameter, a warning message is issued stating that deleted records might be copied. The user can choose option 0 (Continue) to continue the process. By selecting this option, the file is copied and any deleted records in the input file become active records in the output file. The IBM i operating system rejects the copy request if the user specifies COMPRESS(*NO).
- If data is copied to a target System/36 file that is a direct file and is not delete-capable, default values for all Copy File (CPYF) command parameters except FROMMBR, TOMBR, and MBROPT must be specified.
- The IBM i operating system does not support the overwriting of data on the Delete File (DLTF) command. If an IBM i user accessing a System/36 wants to overwrite the data, an application program must be written on IBM i, or the user must access the target System/36 and perform the overwrite operation.
- Depending on how a System/36 file is originally created, the maximum number of records it can contain is approximately eight million. This number can be significantly smaller if the file is not extendable or if sufficient storage space is not available to extend the file to add more records.
- System/36 supports a maximum of three key definitions for logical files and one key definition for keyed physical files.
- System/36 file support does not allow a file with active logical files to be cleared. When some IBM i programs (like ILE COBOL programs) open a file for output only, a command to clear the file is issued. A target System/36 rejects any such command to clear the file if a logical file exists over the file to be cleared.
- System/36 file support automatically skips deleted records. If an IBM i source user wants to change the records in a System/36 base file over which at least one logical file has been built, the file must be opened in I/O mode, specifying direct organization and random access by record number. Then each record can be read by record number and changed. If a deleted record is found, a record-not-found indication is returned, and the record might be written rather than rewritten for that position (write operation rather than a change operation).
- System/36 file support also handles file extensions differently, depending on the file type and the language being used. However, an IBM i user cannot extend any type of System/36 file unless the access method used to access the file is similar to the method used when the file was created.

  If an IBM i user is accessing a System/36 file with an access method similar to the one used to create the file, the file can be extended during its use in the following manner:

  - If the file was created as a sequential file, the IBM i user should take the following actions:
    - ILE COBOL programming language: open the file using the EXTEND option.
    - PL/I PRPQ: open the file using the UPDATE option. Perform a read operation using the POSITION option of LAST, and then perform the write operations.

  (ILE RPG programming language handles any needed file extensions automatically.)
- If the file was created as a direct file, the IBM i user should, if the IBM i language is:
  - ILE COBOL programming language: open the file using the I-O option, position the end of file pointer to the end of the file (using, for example, READ LAST), and perform a write operation.

  - PL/I PRPQ: open the file using the UPDATE option, position the end of file (EOF) pointer to the end of the file (using, for example, READ LAST), and perform a write operation.

    (ILE RPG programming language handles any needed file extensions automatically.)
  - If the file was created as an indexed file, the file is extended each time a write operation is performed for a record having a key that does not already exist in the file.
- The IBM i user can access sequential System/36 files using either sequential or direct (by relative record number) methods, but significant differences occur when EOF or end of data occurs. If a System/36 sequential file is being processed using relative record number access and is opened for either input/output or output only, then, on reaching the end of active records (EOF), you cannot add new records in the available free space beyond the end of data. You will have to close and reopen the file to extend it. To extend the file, you can either reopen it as a sequential file or open a logical file that uses this file as the base file.
- Because the normal access method used for a System/36 file can be changed by IBM i parameters to values other than *RMTFILE, it is possible that DDM might attempt to access the System/36 file in ways that the System/36 might not support. Normally, the default value (*RMTFILE) on the ACCMTH parameter gives the user the needed method of access. The use of access methods not normally expected (such as direct or sequential access to indexed files, or sequential access to direct files) requires the use of an ACCMTH parameter explicitly for the access.

  The normal access method used for a System/36 file can be changed on the IBM i operating system in the following ways:
  - By the ACCMTH parameter of the DDM file commands Create DDM File (CRTDDMF) and Change DDM File (CHGDDMF)
  - By the SEQONLY parameter of the Override with Database File (OVRDBF) command
  - By using the OVRDBF command to override one DDM file with another DDM file that has a different ACCMTH value in effect
- The IBM i user can access a System/36 file using a member name if the member name is *FIRST, or in some cases *LAST, or if the file name is the same as the member name.
- Target System/36 DDM cannot support creating logical files with duplicate (nonunique) keys, because the System/36 local data management key sort sends messages to the server system console with options 1 or 3 when duplicate keys are detected. This forces the server system operator either to change the file attributes to allow duplicate keys or to cancel the target data manager.

  Note: Never cancel the target data manager using a SYSLOG HALT.

Override considerations to System/36 for DDM

When a file override is issued on the IBM i to get records in a logical file on a System/36, the results might be different than expected because of the difference in how each system deals with keyed files.

The IBM i operating system uses access paths and logical files, which produce a single view of a file. A System/36 logical file can be considered a list of keys and relative record numbers.

When the IBM i operating system accesses a System/36 logical file:

- If you specify a relative record number, you receive the record from the underlying System/36 base file that corresponds to that record number. Then if you request to read the next record, you receive the next sequential record from the base file.
- If you specify a key, you receive the record that corresponds to the first occurrence of that key in the index file. If you request to read the next record, you receive the record that matches the next entry in the index file.

The following example shows the various results for records being retrieved from a System/36 logical file by an IBM i program. The example assumes that:

- File S36FILEA is the base file and S36FILEB is the logical file that is built over the base file.
- Both files have DDM files named S36FILEA and S36FILEB that point to corresponding remote files on the target System/36.

- The key field is numeric and it always contains the record number.
- The records in the base file (S36FILEA) are in ascending sequence by key, and the records in the logical file (S36FILEB) are in descending sequence with the same key.
- To create the results shown in the following table, the POSITION parameter value is shown to vary, and no NBRRCDS parameter is specified on either command (which means the total records read is dependent only on the POSITION parameter value).

```
OVRDBF   FILE(S36FILEA)  TOFILE(S36FILEB)
   POSITION(*RRN ... or  *KEY ...)
CPYF   FROMFILE(S36FILEA)  TOFILE(ISERIESFILEB)
CRTFILE(*YES)  FMTOPT(*NOCHK)
```

Depending on the values specified on the **Override with Database File (OVRDBF)** command for the POSITION parameter, the following are the resulting records that are copied into the file ISERIESFILEB when it is created on the IBM i client system:

POSITION parameter (See note)	Resulting records retrieved
*RRN 1	299 records, 1 through 299
*KEY 1	1 record, first record only
*RRN 299	1 record, last record only
*KEY 299	299 records, 299 through 1
*RRN 150	150 records, 150 through 299
*KEY 150	150 records, 150 through 1
Note: This column assumes only one key field for *KEY values and uses the remote file name as the default value for the record format name.	

Personal computer client to IBM i server considerations for DDM
IBM i Access Family uses DDM to allow a personal computer to communicate as a client system to access objects on an IBM i target.

IBM i Access Family uses Level 3.0 of the DDM architecture stream file access support to access folder management services (FMS) folders and documents. The following considerations apply to IBM i Access Family use of the IBM i DDM target support for the DDM architecture, Level 3.0. Other client systems that send Level 2.0 or Level 3.0 DDM architecture requests for stream files and directories might be able to use this information to help in connecting to the IBM i operating system by using DDM.

- A FMS must follow the file or directory name to access folder management services (FMS) folders and documents. There can be one or more blanks between the end of the name and the FMS.
- A leading slash (/) signifies the name is fully qualified. If there is no leading slash, any current directory in use is added to the front of the name given.
- The total length of a fully qualified document name is 76 characters. This includes any current directory that may be in use. This does not include the trailing FMS, which is used for typing purposes.
- A / FMS signifies the root folder for a directory name.
- To reduce the number of messages logged to the job log, some errors occurring on the IBM i target during open, get, put, and close document operations are not logged to the job log. See Table 29 on page 266 for an illustration of these return codes.

Table 29. IBM i return codes

Description	DDM reply	Function
Data stream (DS) in use	STRIUSRM	GET
Data stream (DS) in use	STRIUSRM	PUT

Table 29. IBM i return codes (continued)

Description	DDM reply	Function
Delete document SHDONL(TRUE) specified, but shadow does not exist	EXSCNDRM	DELFIL
Document in use	FILIUSRM	OPEN
Document is read only	ACCINTRM	OPEN
Document not found	EXSCNDRM	DELFIL
Document not found	FILNFNRM	OPEN
End of data	SUBSTRRM	GET
File already open for the declare name	OPNCNFRM	OPEN
File not open	FILNOPRM	GET, PUT, LOCK, UNLOCK
Folder in use	DRCIUSRM	OPEN
Folder not found	DRCNFNRM	OPEN
Substring not valid	SUBSTRRM	UNLOCK
Unlocking a region that is not locked	EXSCNDRM	UNLOCK

- To provide better performance, the IBM i target handles the closing document in a manner such that when the document is closing, a command completion reply message (CMDCMPRM) is returned to the client system before the document is actually closed. If the document is damaged during the closing time, the user never receives this reply message unless he views the job log. When the user opens the file again, the updated data might not be there.
- The IBM i operating system does not support wait on the locking data stream function. The user on the client system must handle the wait function.

Data availability and protection

In a distributed relational database environment, data availability involves not only protecting data on an individual system in the network, but also ensuring that users have access to the data across the network.

The IBM i operating system provides the following array of functions to ensure that data on systems in a distributed relational database network is available for use:

- Save/restore
- Journal management and access path journaling
- Commitment control
- Auxiliary storage pools
- Checksum protection
- Mirrored protection and the uninterruptible power supply

While the system operator for each system is typically responsible for backup and recovery of that system's data, you should also consider aspects of network redundancy as well as data redundancy. When you are planning your strategy, ensure the optimum availability of data across your network. The more critical certain data is to your enterprise, the more ways you should have to access that data.

Recovery support for a distributed relational database

Failures that can occur on a computer system are a system failure (when the entire system is not operating); a loss of the site because of fire, flood, or similar catastrophe; or the damage or loss of an object. For a distributed relational database, a failure on one system in the network prevents users across the entire network from accessing the relational database on that system.

If the relational database is critical to daily business activities at other locations, enterprise operations across the entire network can be disrupted for the duration of one system's recovery time. Clearly, planning for data protection and recovery after a failure is particularly important in a distributed relational database.

Each system in a distributed relational database is responsible for backing up and recovering its own data. Each system in the network also handles recovery procedures after an abnormal system end. However, backup and recovery procedures can be done by the distributed relational database administrator using display station pass-through for those systems with an inexperienced operator or no operator at all.

The most common type of loss is the loss of an object or group of objects. An object can be lost or damaged because of several factors, including power failure, hardware failures, system program errors, application program errors, or operator errors. The IBM i operating system provides several methods for protecting the system programs, application programs, and data from being permanently lost. Depending on the type of failure and the level of protection chosen, most of the programs and data can be protected, and the recovery time can be significantly reduced.

You can use the following methods to protect your data and programs:

Writing data to auxiliary storage

The Force-Write Ratio (FRCRATIO) parameter on the Create File command can be used to force data to be written to auxiliary storage. A force-write ratio of one causes every add, update, and delete request to be written to auxiliary storage immediately for the table in question. However, choosing this option can reduce system performance. Therefore, saving your tables and journaling tables should be considered the primary methods for protecting the database.

Physical protection

Making sure your system is protected from sudden power loss is an important part of ensuring that your server is available to a client. An uninterruptible power supply, which can be ordered separately, protects the system from loss because of power failure, power interruptions, or drops in voltage by supplying power to the system devices until power can be restored. Normally, an uninterruptible power supply does not provide power to all workstations. With the IBM i product, the uninterruptible power supply allows the system to:

- Continue operations during brief power interruptions or momentary drops in voltage.
- End operations normally by closing files and maintaining object integrity.

Data recovery after disk failures for distributed relational databases

Recovery is not possible for recently entered data if a disk failure occurs and all objects are not saved on tape or disk immediately before the failure. After previously saved objects are restored, the system is operational, but the database is not current.

Auxiliary storage pools (ASPs), checksum protection, and mirrored protection are IBM i disk recovery functions that provide methods to recover recently entered data after a disk-related failure. These functions use additional system resources, but provide a high level of protection for systems in a distributed relational database. Because some systems might be more critical as servers than others, the distributed relational database administrator should review how these disk data protection methods can be best used by individual systems within the network.

Related concepts

Backup and recovery

Auxiliary storage pools

An *auxiliary storage pool* (ASP) is one or more physical disk units assigned to the same storage area. ASPs allow you to isolate certain types of objects on specified physical disk units.

The system ASP isolates system programs and the temporary objects that are created as a result of processing by system programs. User ASPs can be used to isolate some objects such as libraries, SQL objects, journals, journal receivers, applications, and data. The IBM i product supports up to 32 basic user ASPs, and 223 independent user ASPs. Isolating libraries or objects in a user ASP protects them from disk failures in other ASPs and reduces recovery time.

In addition to reduced recovery time and isolation of objects, placing objects in an ASP can improve performance. If a journal receiver is isolated in a user ASP, the disks associated with that ASP are dedicated to that receiver. In an environment that requires many read and write operations to the database files, this can reduce arm contention on the disks in that ASP, and can improve journaling performance.

Checksum protection in a distributed relational database

Checksum protection guards against data loss on any disk in an auxiliary storage pool (ASP).

The checksum software maintains a coded copy of ASP data in special checksum data areas within that ASP. Any changes made to permanent objects in a checksum-protected ASP are automatically maintained in the checksum data of the checksum set. If any single disk unit in a checksum set is lost, the system reconstructs the contents of the lost device using the checksum and the data on the remaining functional units of the set. In this way, if any one of the units fails, its contents can be recovered. This reconstructed data reflects the most up-to-date information that was on the disk at the time of the failure. Checksum protection can affect system performance significantly. In a distributed relational database, this might be a concern.

Mirrored protection for a distributed relational database

Mirrored protection increases the availability of a system by duplicating different disk-related hardware components, such as a disk controller, a disk I/O processor, or a bus. The system can remain available after a failure, and service for the failed hardware components can be scheduled at a convenient time.

Different levels of mirrored protection provide different levels of system availability. For example, if only the disk units on a system are mirrored, all disk units have disk-unit-level protection, so the system is protected against the failure of a single disk unit. In this situation, if a controller, I/O processor, or bus failure occurs, the system cannot run until the failing part is repaired or replaced. All mirrored units on the system must have identical disk-unit-level protection and reside in the same ASP. The units in an ASP are automatically paired by the system when mirrored protection is started.

Journal management for distributed relational databases

Journal management can be used as a part of the backup and recovery strategy for relational databases and indexes.

IBM i journal support provides an audit trail and forward and backward recovery. Forward recovery can be used to take an older version of a table and apply changes logged in the journal to the table. Backward recovery can be used to remove changes logged in the journal from the table.

When a collection is created, a journal and an object called a journal receiver are created in the collection. Improved performance is gained when the journal receiver is on a different ASP from the tables. Placing the collection on a user ASP places the tables and journal and journal receivers all in the same user ASP. There is no gain in performance there. Creating a new journal receiver in a different ASP (used just for this journal's journal receivers) and attaching it with the **`Change Journal (CHGJRN)`** command will get the next system-generated journal receivers all in the other user ASP, and then the user will see improved performance.

When a table is created, it is automatically journaled to the journal SQL created in the collection. You are then responsible for using the journal functions to manage the journal, journal receivers, and the journaling of tables to the journal. For example, if a table is moved into a collection, no automatic change to the journaling status occurs. If a table is restored, the normal journal rules apply. That is, if a table is journaled when it is saved, it is journaled to the same journal when it is restored on that system. If the table is not journaled at the time of the save, it is not journaled at restore time. You can stop journaling on any table by using the journal functions, but doing so prevents SQL operations from running under commitment control. SQL operations can still be performed if you have specified COMMIT(*NONE), but this does not provide the same level of integrity that journaling and commitment control provide.

With journaling active, when changes are made to the database, the changes are journaled in a journal receiver before the changes are made to the database. The journal receiver always has the latest database information. All activity is journaled for a database table regardless of how the change was made.

Journal receiver entries record activity for a specific row (added, changed, or deleted), and for a table (opened, table or member saved, and so on). Each entry includes additional control information identifying the source of the activity, the user, job, program, time, and date.

The system journals some file-level changes, including moving a table and renaming a table. The system also journals member-level changes, such as initializing a physical file member, and system-level changes, such as initial program load (IPL). You can add entries to a journal receiver to identify significant events (such as the checkpoint at which information about the status of the job and the system can be journaled so that the job step can be restarted later) or to help in the recovery of applications.

For changes that affect a single row, row images are included following the control information. The image of the row after a change is made is always included. Optionally, the row image before the change is made can also be included. You control whether to journal both before and after row images or just after row images by specifying the IMAGES parameter on the **`Start Journal Physical File (STRJRNPF)`** command.

All journaled database files are automatically synchronized with the journal when the system is started (IPL time) or while varying on an independent ASP. If the system ended abnormally, or the independent ASP varied off abnormally, some database changes might be in the journal, but not yet reflected in the database itself. If that is the case, the system automatically updates the database from the journal to bring the tables up to date.

Journaling can make saving database tables easier and faster. For example, instead of saving entire tables every day, you can save the journal receivers that contain the changes to the tables. You can still save the entire tables on a regular basis. This method can reduce the amount of time it takes to perform your daily save operations.

The **`Display Journal (DSPJRN)`** command can be used to convert journal receiver entries to a database file. Such a file can be used for activity reports, audit trails, security, and program debugging.

Related concepts

Journal management

Related reference

Change Journal (CHGJRN) command

Display Journal (DSPJRN) command

Start Journal Physical File (STRJRNPF) command

Index recovery

An index describes the order in which rows are read from a table. When indexes are recorded in the journal, the system can recover the index to avoid spending a significant amount of time rebuilding indexes during the IPL that follows an abnormal system end or while varying on an independent ASP after it was varied off abnormally.

When you journal tables, images of changes to the rows in the table are written to the journal. These row images are used to recover the table if the system, or independent ASP, ends abnormally. However, after an abnormal end, the system might find that indexes built over the table are not synchronized with the data in the table. If an access path and its data are not synchronized, the system must rebuild the index to ensure that the two are synchronized and usable.

When indexes are journaled, the system records images of the index in the journal to provide known synchronization points between the index and its data. By having that information in the journal, the system can recover both the data and the index, and ensure that the two are synchronized. In such cases, the lengthy time to rebuild the indexes can be avoided.

The IBM i operating system provides several functions to assist with index recovery. All indexes on the system have a maintenance option that specifies when the index is maintained. SQL indexes are created with an attribute of *IMMED maintenance.

In the event of a power failure or abnormal server failure, indexes that are in the process of change might need to be rebuilt to make sure they agree with the data. All indexes on the server have a recovery option that specifies when the index should be rebuilt if necessary. All SQL indexes with an attribute of UNIQUE are created with a recovery attribute of *IPL, which means these indexes are rebuilt before the IBM i licensed program has been started. All other SQL indexes are created with the *AFTIPL recovery attribute, which means they are rebuilt after the operating system has been started or after the independent ASP has varied on. During an IPL or vary on of an independent ASP, you can see a display showing indexes that need to be rebuilt and their recovery options, and you can override these recovery options.

SQL indexes are not journaled automatically. You can use the **`Start Journal Access Path (STRJRNAP)`** command to journal any index created by SQL operations. The system save and restore functions allow you to save indexes when a table is saved by using ACCPTH(*YES) on the **`Save Object (SAVOBJ)`** or **`Save Library (SAVLIB)`** command. If you must restore a table, there is no need to rebuild the indexes. Any indexes not previously saved and restored are automatically and asynchronously rebuilt by the database.

Before journaling indexes, you must start journaling for the tables associated with the index. In addition, you must use the same journal for the index and its associated table.

Index journaling is designed to minimize additional output operations. For example, the system writes the journal data for the changed row and the changed index in the same output operation. However, you should seriously consider isolating your journal receivers in user ASPs when you start journaling your indexes. Placing journal receivers in their own user ASP provides the best journal management performance, while helping to protect them from a disk failure.

Related reference

Start Journal Access Path (STRJRNAP) command

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Designing tables to reduce index rebuilding time

Table design can also help reduce index recovery time.

You can divide a large master table into a history table and a transaction table. The transaction table is then used for adding new data and the history table is used for inquiry only. Each day, you can merge the transaction data into the history table and then clear the transaction file for the next day's data. With this design, the time to rebuild indexes can be shortened, because if the system abnormally ends during the day, the index to the smaller transaction table might need to be rebuilt. However, because the index to the large history table is read-only for most of the day, it might not be out of synchronization with its data, and might not have to be rebuilt.

Consider the trade-off between using a table design to reduce index rebuilding time and using system-supplied functions like access path journaling. The table design described in the previous paragraph might require a more complex application design. After evaluating your situation, you can decide whether to use system-supplied functions like access path journaling rather than design more complex applications.

System-managed access-path protection

System-managed access-path protection (SMAPP) provides automatic protection for access paths.

Using the SMAPP support, you do not have to use the journaling commands, such as the **`Start Journal Access Path (STRJRNAP)`** command, to get the benefits of access path journaling. SMAPP support recovers access paths after an abnormal system end rather than rebuilding them while restarting the system or varying on an independent ASP.

The SMAPP support is turned on with the shipped system.

The system determines which access paths to protect based on target access path recovery times provided by the user or by using a system-provided default time. The target access path recovery times can be specified as a system-wide value or on an ASP basis. Access paths that are being journaled to a user-defined journal are not eligible for SMAPP protection because they are already protected.

Related concepts

System-managed access-path protection

Related reference

Start Journal Access Path (STRJRNAP) command

Transaction recovery through commitment control

Commitment control is an extension of the IBM i journal management function. The system can identify and process a group of relational database changes as a single unit of work (transaction).

An SQL COMMIT statement guarantees that the group of operations is completed. An SQL ROLLBACK statement guarantees that the group of operations is backed out. The only SQL statements that cannot be committed or rolled back are:

- DROP COLLECTION
- GRANT or REVOKE if an authority holder exists for the specified object

Under commitment control, tables and rows used during a transaction are locked from other jobs. This ensures that other jobs do not use the data until the transaction is complete. At the end of the transaction, the program issues an SQL COMMIT or ROLLBACK statement, freeing the rows. If the system or job ends abnormally before the commit operation is performed, all changes for that job since the last time a commit or rollback operation occurred are rolled back. Any affected rows that are still locked are then unlocked. The lock levels are as follows:

***NONE**

Commitment control is not used. Uncommitted changes in other jobs can be seen.

***CHG**

Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the unit of work (transaction) is completed. Uncommitted changes in other jobs can be seen.

***CS**

Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the unit of work (transaction) is completed. A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***ALL**

Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows read, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

Table 30 on page 273 shows the record lock duration for each of these lock level values.

If you request COMMIT (*CHG), COMMIT (*CS), or COMMIT (*ALL) when the program is precompiled or when interactive SQL is started, then SQL sets up the commitment control environment by implicitly calling the **`Start Commitment Control (STRCMTCTL)`** command. The LCKLVL parameter specified when SQL starts commitment control is the lock level specified on the COMMIT parameter on the CRTSQL*xxx* commands. NFYOBJ(*NONE) is specified when SQL starts commitment control. To specify a different NFYOBJ parameter, issue a STRCMTCTL command before starting SQL.

Note: When running with commitment control, the tables referred to in the application program by data manipulation language statements must be journaled. The tables do not have to be journaled at precompile time, but they must be journaled when you run the application.

If a remote relational database is accessing data on the system and requesting commit-level repeatable read (*RR), the tables are locked until the query is closed. If the cursor is read only, the table is locked (*SHRNUP). If the cursor is in update mode, the table is locked (*EXCLRD).

The journal created in the SQL collection is normally the journal used for logging all changes to SQL tables. You can, however, use the system journal functions to journal SQL tables to a different journal.

Commitment control can handle up to 500 000 000 distinct row changes in a unit of work. If COMMIT(*ALL) is specified, all rows read are also included in the 500 000 000 limit. (If a row is changed or read more than once in a unit of work, it is only counted once toward the 500 000 000 limit.) Maintaining a large number of locks adversely affects system performance and does not allow concurrent users to access rows locked in the unit of work until the unit of work is completed. It is, therefore, more efficient to keep the number of rows that are processed in a unit of work small.

The HOLD value on COMMIT and ROLLBACK statements allows you to keep the cursor open and start another unit of work without issuing an OPEN statement again. If non-IBM i connections are not released for a program and SQL is still in the call stack, the HOLD value is not available. If ALWBLK(*ALLREAD) and either COMMIT(*CHG) or COMMIT(*CS) are specified when the program is precompiled, all read-only cursors allow blocking of rows and a ROLLBACK HOLD statement does not roll the cursor position back.

If there are locked rows (records) pending from running a SQL precompiled program or an interactive SQL session, a COMMIT or ROLLBACK statement can be issued from the server Command Entry display. Otherwise, an implicit ROLLBACK operation occurs when the job is ended.

You can use the **`Work with Commitment Definitions (WRKCMTDFN)`** command to monitor the status of commitment definitions and to free locks and held resources that are involved with commitment control activities across systems.

Table 30. Record lock duration

SQL statement	COMMIT parameter	Duration of record locks	Lock type
SELECT INTO	*NONE *CHG *CS *ALL (See note 2)	No locks No locks Row locked when read and released From read until ROLLBACK or COMMIT	 READ READ
FETCH (read-only cursor)	*NONE *CHG *CS *ALL (See note 2)	No locks No locks From read until the next FETCH From read until ROLLBACK or COMMIT	 READ READ
FETCH (update or delete capable cursor) See note 1	*NONE	When record not updated or deleted from read until next FETCH When record is updated or deleted from read until UPDATE or DELETE	UPDATE
	*CHG	When record not updated or deleted from read until next FETCH When record is updated or deleted from read until UPDATE or DELETE	UPDATE
	*CS	When record not updated or deleted from read until next FETCH When record is updated or deleted from read until UPDATE or DELETE	UPDATE
	*ALL	From read until ROLLBACK or COMMIT	UPDATE[3]

INSERT (target table)	*NONE
	*CHG
	*CS
	*ALL
		No locks
		From insert until ROLLBACK or COMMIT
		From insert until ROLLBACK or COMMIT
		From insert until ROLLBACK or COMMIT
			
			UPDATE
			UPDATE
			UPDATE[4]

Table 30. Record lock duration (continued)

SQL statement	COMMIT parameter	Duration of record locks	Lock type
INSERT (tables in subselect)	*NONE *CHG *CS *ALL	No locks No locks Each record locked while being read From read until ROLLBACK or COMMIT	 READ READ
UPDATE (non-cursor)	*NONE *CHG *CS *ALL	Each record locked while being updated From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT	UPDATE UPDATE UPDATE UPDATE
DELETE (non-cursor)	*NONE *CHG *CS *ALL	Each record locked while being deleted From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT	UPDATE UPDATE UPDATE UPDATE
UPDATE (with cursor)	*NONE *CHG *CS *ALL	Lock released when record updated From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT	UPDATE UPDATE UPDATE UPDATE
DELETE (with cursor)	*NONE *CHG *CS *ALL	Lock released when record deleted From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT From read until ROLLBACK or COMMIT	UPDATE UPDATE UPDATE UPDATE
Subqueries (update or delete capable cursor or UPDATE or DELETE non-cursor)	*NONE *CHG *CS *ALL (see note 2)	From read until next FETCH From read until next FETCH From read until next FETCH From read until ROLLBACK or COMMIT	READ READ READ READ
Subqueries (read-only cursor or SELECT INTO)	*NONE *CHG *CS *ALL	No locks No locks Each record locked while being read From read until ROLLBACK or COMMIT	 READ READ

Table 30. Record lock duration (continued)

SQL statement	COMMIT parameter	Duration of record locks	Lock type

Notes:

1. A cursor is open with UPDATE or DELETE capabilities if the result table is not read-only (see description of DECLARE CURSOR) and if one of the following items is true:
   - The cursor is defined with a FOR UPDATE clause.
   - The cursor is defined without a FOR UPDATE, FOR FETCH ONLY, or ORDER BY clause and the program contains at least one of the following items:
     - Cursor UPDATE referring to the same cursor-name
     - Cursor DELETE referring to the same cursor-name
     - An EXECUTE or EXECUTE IMMEDIATE statement with ALWBLK(*READ) or ALWBLK(*NONE) specified on the CRTSQL*xxx* command
2. A table or view can be locked exclusively in order to satisfy COMMIT(*ALL). If a subselect is processed that includes a group by or union, or if the processing of the query requires the use of a temporary result, an exclusive lock is acquired to protect you from seeing uncommitted changes.
3. If the row is not updated or deleted, the lock is reduced to *READ.
4. An UPDATE lock on rows of the target table and a READ lock on the rows of the subselect table.
5. A table or view can be locked exclusively in order to satisfy repeatable read. Row locking is still done under repeatable read. The locks acquired and their duration are identical to *ALL.

Related concepts

Troubleshooting transactions and commitment control

Related tasks

Working with commitment definitions in a distributed relational database

Use the **Work with Commitment Definitions (WRKCMTDFN)** command if you want to work with the commitment definitions on the system.

Related reference

DECLARE CURSOR

Start Commitment Control (STRCMTCTL) command

Work with Commitment Definitions (WRKCMTDFN) command

Save and restore processing for a distributed relational database

Saving and restoring data and programs allows recovery from a program or system failure, exchange of information between systems, or storage of objects or data offline. A comprehensive backup policy at each system in the distributed relational database network ensures that a system can be restored and quickly made available to network users in the event of a problem.

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Save Changed Object (SAVCHGOBJ) command

Save Save File Data (SAVSAVFDTA) command

Save System (SAVSYS) command

Restore Library (RSTLIB) command

Restore Object (RSTOBJ) command

Restore User Profiles (RSTUSRPRF) command

Restore Authority (RSTAUT) command

Restore Configuration (RSTCFG) command

Save and restore processing for a distributed relational database

Saving and restoring data and programs allows recovery from a program or system failure, exchange of information between systems, or storage of objects or data offline. A comprehensive backup policy at each system in the distributed relational database network ensures that a system can be restored and quickly made available to network users in the event of a problem.

Saving the system on external media, such as tape, protects system programs and data from disasters, such as fire or flood. However, information can also be saved to a disk file called a save file. A *save file* is a disk-resident file used to store data until it is used in input and output operations or for transmission to another IBM i operating system over communication lines. Using a save file allows unattended save operations because an operator does not need to load tapes. In a distributed relational database, save files can be sent to another system as a protection method.

When information is restored, the information is written from tape or a save file into auxiliary storage where it can be accessed by system users.

The IBM i operating system has a full set of commands to save and restore your database tables and SQL objects:

- The **Save Library (SAVLIB)** command saves one or more collections
- The **Save Object (SAVOBJ)** command saves one or more objects such as SQL tables, views and indexes
- The **Save Changed Object (SAVCHGOBJ)** command saves any objects that have changed since either the last time the collection was saved or from a specified date
- The **Save Save File Data (SAVSAVFDTA)** command saves the contents of a save file
- The **Save System (SAVSYS)** command saves the operating system, security information, device configurations, and system values
- The **Restore Library (RSTLIB)** command restores a collection
- The **Restore Object (RSTOBJ)** command restores one or more objects such as SQL tables, views and indexes
- The **Restore User Profiles (RSTUSRPRF)**, **Restore Authority (RSTAUT)** and **Restore Configuration (RSTCFG)** commands restore user profiles, authorities, and configurations saved by a **Save System (SAVSYS)** command

Related concepts

Troubleshooting transactions and commitment control

Related tasks

Working with commitment definitions in a distributed relational database
Use the **Work with Commitment Definitions (WRKCMTDFN)** command if you want to work with the commitment definitions on the system.

Related reference

Transaction recovery through commitment control
Commitment control is an extension of the IBM i journal management function. The system can identify and process a group of relational database changes as a single unit of work (transaction).

DECLARE CURSOR

Start Commitment Control (STRCMTCTL) command

Work with Commitment Definitions (WRKCMTDFN) command

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Save Changed Object (SAVCHGOBJ) command

Save Save File Data (SAVSAVFDTA) command

Save System (SAVSYS) command

Restore Library (RSTLIB) command

Restore Object (RSTOBJ) command

Restore User Profiles (RSTUSRPRF) command

Restore Authority (RSTAUT) command

Restore Configuration (RSTCFG) command

Saving and restoring indexes in the distributed relational database environment

Restoring an SQL index can be faster than rebuilding it. Although times vary depending on a number of factors, rebuilding a database index takes approximately one minute for every 10 000 rows.

After restoring the index, the table might need to be brought up to date by applying the latest journal changes (depending on whether journaling is active). Even with this additional recovery time, you might find it faster to restore indexes rather than to rebuild them.

The system ensures the integrity of an index before you can use it. If the system determines that the index is unusable, the system attempts to recover it. You can control when an index will be recovered. If the system ends abnormally, during the next IPL the system automatically lists those tables requiring index or view recovery. You can decide whether to rebuild the index or to attempt to recover it at one of the following times:

- During the IPL
- After the IPL
- When the table is first used

Related concepts

Backup and recovery

Saving and restoring security information in the distributed relational database environment

You can use a variety of CL commands to save and restore security information.

If you make frequent changes to your system security environment by updating user profiles and updating authorities for users in the distributed relational database network, you can save security information to media or a save file without a complete **Save System (SAVSYS)** command, a long-running process that uses a dedicated system. With the **Save Security Data (SAVSECDTA)** command, you can save security data in a shorter time without using a dedicated system. Data saved using the **SAVSECDTA** command can be restored using the **Restore User Profiles (RSTUSRPRF)** or **Restore Authority (RSTAUT)** command.

Included in the security information that the **SAVSECDTA** and **RSTUSRPRF** commands can save and restore are the server authentication entries that the DRDA TCP/IP support uses to store and retrieve remote system user ID and password information.

Related reference

Save System (SAVSYS) command

Save Security Data (SAVSECDTA) command

Restore User Profiles (RSTUSRPRF) command

Restore Authority (RSTAUT) command

Saving and restoring SQL packages in the distributed relational database environment

When an application program that refers to a relational database on a remote system is precompiled and bound, an SQL package is created on the server to contain the control structures necessary to process any SQL statements in the application.

An SQL package is an IBM i object, so it can be saved to media or a save file using the Save Object (SAVOBJ) command and restored using the Restore Object (RSTOBJ) command.

An SQL package must be restored to a collection that has the same name as the collection from which it was saved, and an SQL package cannot be renamed.

Related reference

Restore Object (RSTOBJ) command

Save Object (SAVOBJ) command

Saving and restoring relational database directories

The relational database directory is not an IBM i object. Instead, it is made up of files that are opened by the system at initial program load (IPL) time.

Because of this, the **Save Object (SAVOBJ)** command cannot be used to directly save these files. You can save the relational database directory by creating an output file from the relational database directory data. This output file can then be used to add entries to the directory again if it is damaged.

When entries have been added and you want to save the relational database directory, specify the OUTFILE parameter on the **Display Relational Database Directory Entry (DSPRDBDIRE)** command to send the results of the command to an output file. The output file can be saved to tape or to a save file and restored to the system. If your relational database directory is damaged or your system needs to be recovered, you can restore the output file that contains relational database entry data using a control language (CL) program. The CL program reads data from the restored output file and creates the CL commands that add entries to a new relational database directory.

For example, the relational database directory for the Spiffy Corporation MP000 system is sent to an output file named RDBDIRM as follows:

```
DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBDIRM)
```

The sample CL program that follows reads the contents of the output file RDBDIRM and re-creates the relational database directory using the **Add Relational Database Directory Entry (ADDRDBDIRE)** command. Note that the old directory entries are removed before the new entries are made.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
/******************************************************************/
/* -  Restore RDB Entries from output file created with:        - */
/* -   DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBDIRM)             - */
/* -    FROM A V4R2 OR LATER LEVEL OF OS/400 or IBM i                  - */
/******************************************************************/
PGM        PARM(&ACTIVATE)
DCL       VAR(&ACTIVATE) TYPE(*CHAR) LEN(7)

/* Declare Entry Types Variables to Compare with &RWTYPE          */
 DCL &LOCAL    *CHAR 1
 DCL &SNA      *CHAR 1
 DCL &IP       *CHAR 1
 DCL &ARD      *CHAR 1
 DCL &ARDSNA   *CHAR 1
 DCL &ARDIP    *CHAR 1
 DCL &RWTYPE   *CHAR 1
 DCL &RWRDB    *CHAR 18
 DCL &RWRLOC   *CHAR 8
 DCL &RWTEXT   *CHAR 50
 DCL &RWDEV    *CHAR 10
 DCL &RWLLOC   *CHAR 8
 DCL &RWNTID   *CHAR 8
 DCL &RWMODE   *CHAR 8
 DCL &RWTPN    *CHAR 8
 DCL &RWSLOC   *CHAR 254
 DCL &RWPORT   *CHAR 14
 DCL &RWDPGM   *CHAR 10
 DCL &RWDLIB   *CHAR 10

 DCLF     FILE(RDBSAV/RDBDIRM) /* SEE PROLOG CONCERNING THIS                       */
   IF COND(&ACTIVATE = SAVE) THEN(GOTO CMBLBL(SAVE))
   IF COND(&ACTIVATE = RESTORE) THEN(GOTO CMDLBL(RESTORE))
 SAVE:
 CRTLIB RDBSAV
 DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBSAV/RDBDIRM)
 GOTO CMDLBL(END)

 RESTORE:
/* Initialize Entry Type Variables to Assigned Values                      */
 CHGVAR &LOCAL   '0'               /* Local RDB (one per system) */
 CHGVAR &SNA     '1'               /* APPC entry (no ARD pgm)    */
 CHGVAR &IP      '2'               /* TCP/IP entry (no ARD pgm)  */
 CHGVAR &ARD     '3'               /* ARD pgm w/o comm parms     */
 CHGVAR &ARDSNA  '4'               /* ARD pgm with APPC parms    */
 CHGVAR &ARDIP   '5'                /* ARD pgm with TCP/IP parms  */

 RMVRDBDIRE    RDB(*ALL)            /* Clear out directory        */

NEXTENT:                            /* Start of processing loop   */
 RCVF                               /* Get a directory entry      */
 MONMSG     MSGID(CPF0864) EXEC(DO) /* End of file processing     */
  QSYS/RCVMSG PGMQ(*SAME (*)) MSGTYPE(*EXCP) RMV(*YES) MSGQ(*PGMQ)
  GOTO CMDLBL(LASTENT)
 ENDDO

/*                Process entry based on type code                */
 IF (&RWTYPE = &LOCAL) THEN(     +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) )

 ELSE IF (&RWTYPE = &SNA) THEN(  +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
                   DEV(&RWDEV) LCLLOCNAME(&RWLLOC) +
                   RMTNETID(&RWNTID) MODE(&RWMODE) TNSPGM(&RWTPN) )

 ELSE IF (&RWTYPE = &IP) THEN(  +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWSLOC *IP) +
                   TEXT(&RWTEXT) PORT(&RWPORT) )

 ELSE IF (&RWTYPE = &ARD) THEN(  +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
                   ARDPGM(&RWDLIB/&RWDPGM) )

 ELSE IF (&RWTYPE = &ARDSNA) THEN(  +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
                   DEV(&RWDEV) LCLLOCNAME(&RWLLOC) +
                   RMTNETID(&RWNTID) MODE(&RWMODE) TNSPGM(&RWTPN) +
                   ARDPGM(&RWDLIB/&RWDPGM) )

 ELSE IF (&RWTYPE = &ARDIP) THEN(  +
   QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWSLOC *IP) +
                    TEXT(&RWTEXT) PORT(&RWPORT) +
                    ARDPGM(&RWDLIB/&RWDPGM) )

 GOTO CMDLBL(NEXTENT)

LASTENT:
 RETURN
 DLTLIB RDBSAV
 END


ENDPGM
```

There is an alternate method of restoring the directory, for the case when no output file of the type described previously is available. This method is to extract the object from a saved system, restore it to some other library, and then manually enter the entries in it with the **Add Relational Database Directory Entry (ADDRDBDIRE)** command.

The files that make up the relational database directory are saved when a **Save System (SAVSYS)** command is run. The physical file that contains the relational database directory can be restored from the save media to your library with the following **Restore Object (RSTOBJ)** command:

```
RSTOBJ     OBJ(QADBXRDBD)  SAVLIB(QSYS)
           DEV(TAP01)  OBJTYPE(*FILE)
           LABEL(Qpppppppvrmxx0003)
           RSTLIB(your lib)
```

In this example, the relational database directory is restored from tape. The characters *ppppppp* in the LABEL parameter represent the product number of IBM i (for example, 5722SS1 for Version 5 Release 3). The *vrm* in the LABEL parameter is the version, release, and modification level of IBM i. The *xx* in the LABEL parameter refers the last 2 digits of the current system language value. For example, 2924 is for the English language; therefore, the value of *xx* is 24.

After you restore this file to your library, you can use the information in the file to manually re-create the relational database directory.

Related reference

Example: Setting up a relational database directory

The Spiffy Corporation network example illustrates how the relational database directory is set up and used on systems in a distributed relational database network.

Add Relational Database Directory Entry (ADDRDBDIRE) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Restore Object (RSTOBJ) command

Save Object (SAVOBJ) command

Save System (SAVSYS) command

Network redundancy considerations for a distributed relational database

Network redundancy provides different ways for users on the distributed relational database network to access a relational database on the network.

If there is only one communications path from a client to a server, when the communications line is down, users on the client do not have access to the server relational database. For this reason, network redundancy considerations are important to the distributed relational database administrator for the Spiffy Corporation. For example, consider service booking or customer parts purchasing issues for a dealership. When a customer is waiting for service or to purchase a part, the service clerk needs access to all authorized tables of enterprise information to schedule work or sell parts.

If the local system is down, no work can be done. If the local system is running but a request to a remote system is needed to process work and the remote system is down, the request cannot be handled. In the Spiffy Corporation example, this might mean that a dealership cannot request parts information from a regional inventory center. Also, if a server that handles many client jobs is down, none of the clients can complete their requests. In the case of the Spiffy Corporation network, if a regional center is down, none of the servers it supports can order parts.

Providing the region's dealerships with access to regional inventory data is important to the Spiffy Corporation distributed relational database administrator. Providing paths through the network to data can be addressed in several ways. The original network configuration for the Spiffy Corporation linked the end node dealerships to their respective network node regional centers.

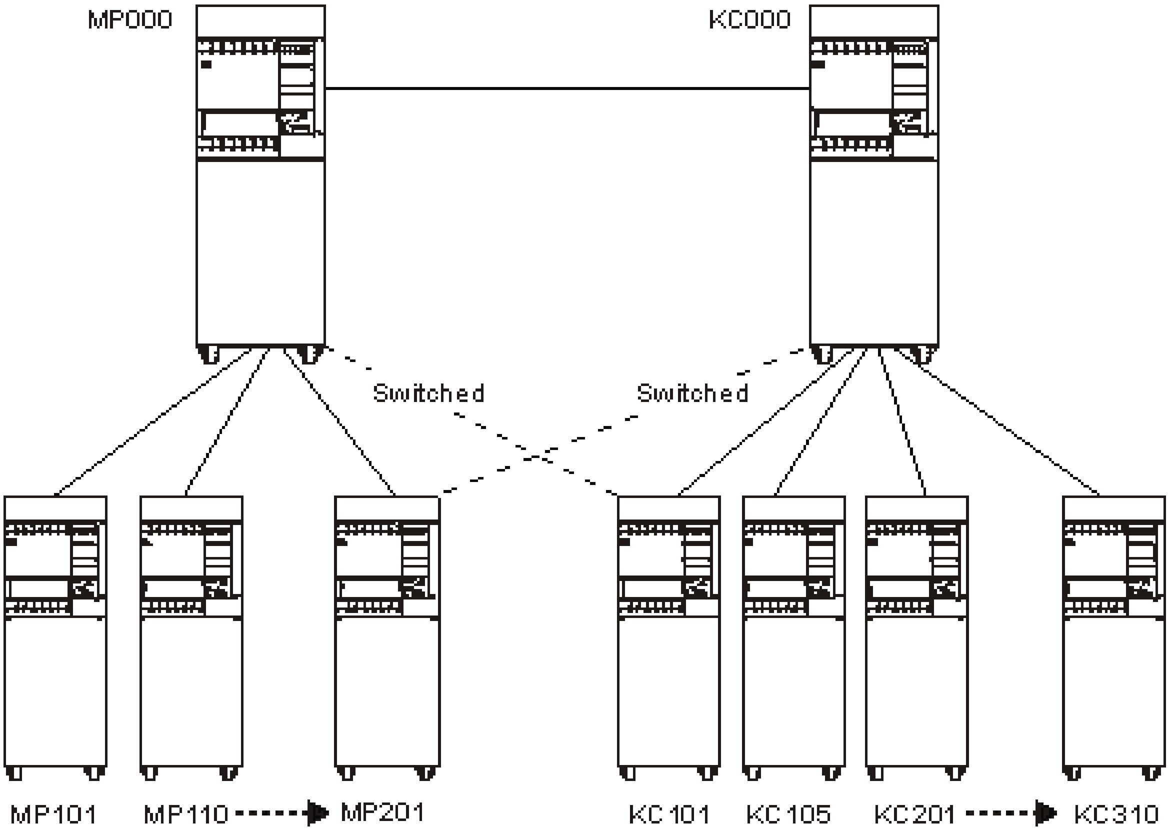


Figure 29. Alternative network paths

An alternative for some dealerships might be a switched-line connection to a different regional center. If the local regional center is unavailable to the network, access to another server allows the requesting dealership to obtain information that is needed to do their work. In the first figure, some clients served by the MP000 system establish links to the KC000 system, which can be used whenever the MP000 system is unavailable. The Vary Configuration (VRYCFG) or Work with Configuration Status (WRKCFGSTS) command can be used by a system operator or distributed relational database administrator to vary the line on when needed and vary the line off when the primary server is available.

Another alternative might be if one of the larger area dealerships also acted as a server for other dealerships. As shown in the second figure, an end node is only a server to other end nodes through its network node. In the first figure, if the link to Minneapolis is down, none of the dealerships can query another (end node) for inventory. The configuration illustrated above can be changed so that one of the dealerships is configured as an APPN network node, and lines to that dealership from other area dealerships are set up.

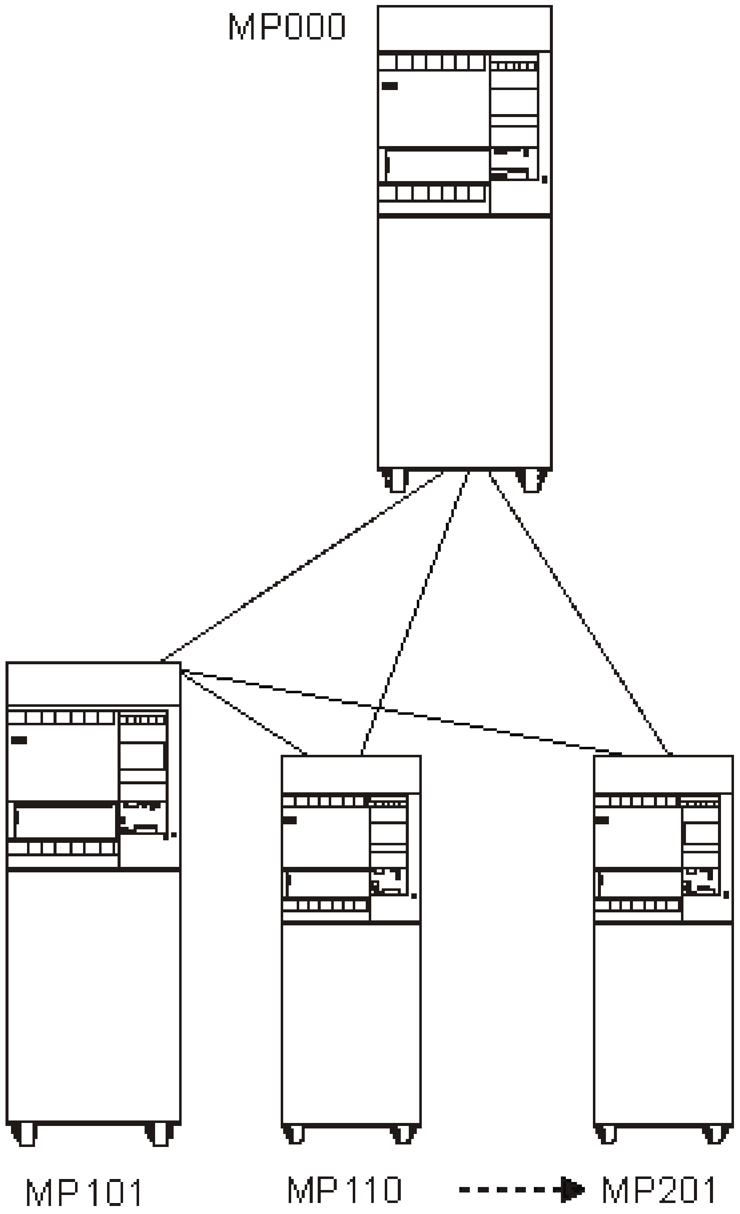


Figure 30. Alternate server

Related reference

Vary Configuration (VRYCFG) command

Work with Configuration Status (WRKCFGSTS) command

Data redundancy in your distributed relational database network

Data redundancy in a distributed relational database also provides different ways for users on the distributed relational database network to access a database on the network.

The considerations a distributed relational database administrator examines to create a data redundancy strategy are more complex than ensuring communications paths are available to the data.

Tables can be replicated across systems in the network, or a snapshot of data can be used to provide data availability. DB2 DataPropagator can provide this capability.

The following figure shows that a copy of the MP000 system distributed relational database can be stored on the KC000 system, and a copy of the KC000 system distributed relational database can be stored on the MP000 system. The client from one region can link to the other server to query or to update a replicated copy of their relational database.

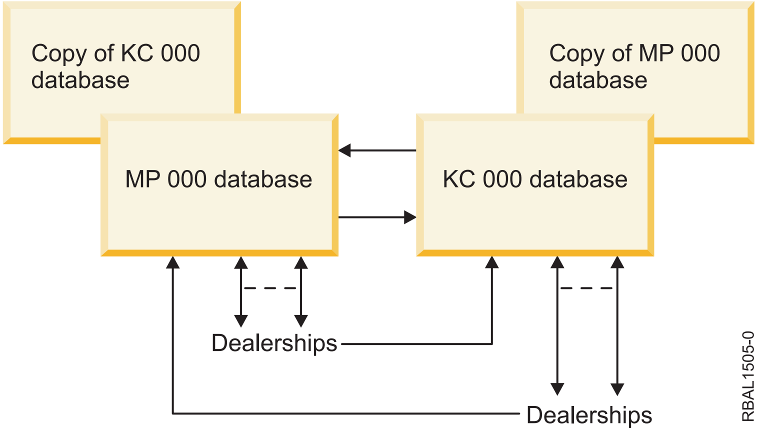


Figure 31. Data redundancy example

The administrator must decide what is the most efficient, effective strategy to allow distributed relational database processing. Alternative strategies might include these scenarios.

One alternative might be that when MP000 is unavailable, its clients connect to the KC000 system to query a read-only snapshot of the MP000 distributed relational database so service work can be scheduled.

DB2 DataPropagator can provide a read-only copy of the tables to a remote system on a regular basis. For the Spiffy Corporation, this might be at the end or the beginning of each business day. In this example, the MP000 database snapshot provides a 24-hour-old, last-point-in-time picture for dealerships to use for scheduling only. When the MP000 system is back on line, its clients query the MP000 distributed relational database to completely process inventory requests or other work queried on the read-only copy.

Another alternative might be that Spiffy Corporation wants dealership users to be able to update a replicated table at another server when their regional server is unavailable.

For example, a client that normally connects to the MP000 database can connect to a replicated MP000 database on the KC000 system to process work. When the MP000 system is available again, the MP000 relational database can be updated by applying journal entries from activity that originated in its replicated tables at the KC000 location. When these journal entries have been applied to the original MP000 tables, distributed relational database users can access the MP000 as a server again.

Journal management processes on each regional system update all relational databases. The amount of journal management copy activity in this situation should be examined because of potential adverse performance effects on these systems.

DRDA and DDM performance

No matter what kind of application programs you are running on a system, performance can always be a concern. For a distributed relational database, network, system, and application performance are all crucial.

System performance can be affected by the size and organization of main and auxiliary storage. There can also be performance gains if you know the strengths and weaknesses of SQL programs.

Related concepts

Troubleshooting DRDA and DDM

When a problem occurs accessing a distributed relational database, determine the nature of the problem and whether it is a problem with the application or a problem with the local or remote system.

Improving distributed relational database performance through the network

You can improve the performance of your network in various ways.

Among them are the following ways:

- Line speed
- Pacing
- Frame size
- RU sizing
- Connection type (nonswitched versus switched)

  Note: Unprotected conversations are used for DRDA connections under one of the following circumstances:

  - When the connection is performed from a program using remote unit of work (RUW) connection management
  - If the program that is making the connection is not running under commitment control
  - If the database to which the connection is made does not support two-phase commit for the protocol that is being used

  If the characteristics of the data are such that the transaction only affects one database management system, establishing the connection from a program using RUW connection management or from a program running without commitment control can avoid the overhead associated with two-phase commit flows.

  Additionally, when conversations are kept active with DDMCNV(*KEEP) and those conversations are protected conversations, two-phase commit flows are sent regardless of whether the conversation was used for DRDA or DDM processing during the unit of work. Therefore, when you run with DDMCNV(*KEEP), it is better to run with unprotected conversations if possible. If running with protected conversations, you should run with DDMCNV(*DROP) and use the RELEASE statement to end the connection and the conversation at the next commit when the conversation will not be used in future units of work.

Related concepts

Communications Management PDF

APPC, APPN, and HPR

TCP/IP setup

Improving distributed relational database performance through the system

Achieving efficient system performance requires a proper balance among system resources. Overusing any resource adversely affects performance. This topic describes the commands that are available to help you observe the performance of your system.

You can use the Performance Tools licensed program to help analyze the system performance. In addition, there are some system commands available to help you observe the performance of your system:

- **Work with System Status (WRKSYSSTS)** command
- **Work with Disk Status (WRKDSKSTS)** command
- **Work with Active Jobs (WRKACTJOB)** command

In using them, you should observe system performance during typical levels of activity. For example, statistics gathered when no jobs are running on the system are of little value in assessing system performance. To observe the system performance, complete the following steps:

1. Enter the **WRKSYSSTS**, **WRKDSKSTS**, or **WRKACTJOB** command.

2. Allow the system to collect data for a minimum of 5 minutes.
3. Press F5 (Refresh) to refresh the display and present the performance data.
4. Tune your system based on the new data.

Press F10 (Restart) to restart the elapsed time counter.

Use both the **Work with System Status (WRKSYSSTS)** and the **Work with Active Jobs (WRKACTJOB)** commands when observing the performance of your system. With each observation period, you should examine and evaluate the measures of server performance against the goals you have set.

Some of the typical measures include:

- Interactive throughput and response time, available from the **WRKACTJOB** display.
- Batch throughput. Observe the AuxIO and CPU% values for active batch jobs.
- Spool throughput. Observe the AuxIO and CPU% values for active writers.

Each time you make tuning adjustments, you should measure and compare all of your main performance measures. Make and evaluate adjustments one at a time.

Related concepts

Work management

Related tasks

Working with active jobs in a distributed relational database
Use the **Work with Active Jobs (WRKACTJOB)** command if you want to monitor the jobs running for several users, or if you are looking for a job and you do not know the job name or the user ID.

Related reference

Work with System Status (WRKSYSSTS) command

Work with Disk Status (WRKDSKSTS) command

Work with Active Jobs (WRKACTJOB) command

Performance considerations for DRDA

Distributed relational database performance is affected by the overall design of the database. The location of distributed data, the level of commitment control you use, and the design of your SQL indexes all affect performance.

Related concepts

Planning and design
To prepare for a distributed relational database, you must understand both the needs of the business and relational database technology. To prepare for the use of DDM, you need to meet several requirements including communication requirements, security requirements, and modification requirements.

Factors that affect blocking for DRDA

A very important performance factor is whether blocking occurs when data is transferred between the application requester (AR) and the application server (AS). A group of rows transmitted as a block of data requires much less communications overhead than the same data sent one row at a time.

One way to control blocking when connected to another IBM i platform is to use the SQL multiple-row INSERT and multiple-row FETCH statements. The multiple-row FETCH forces the blocking of the number of rows specified in the FOR n ROWS clause, unless a hard error or end of data is encountered. The following discussion gives rules for determining if blocking will occur for single-row FETCHs.

DB2 for IBM i to DB2 for IBM i blocking occurs under these conditions. These conditions do not apply to the use of the multiple-row FETCH statement. Any condition listed under each of the following cases is sufficient to prevent blocking.

- The cursor is scroll-sensitive.
- The cursor contains LOBs that can be overridden as locators.
- The result set is not an array result set.

- The DRDA AR does not force nonblocking.

Refer to the cursor attributes of your database interface for more information.

Related reference

Resolving loop, wait, or performance problems
If a request takes longer than expected to complete, check the application requester (AR) first.

Factors that affect the size of DRDA query blocks
DRDA query blocks are controlled in one of these ways.

- If the application requester (AR) specifies a query row set of size *x*, the application server (AS) returns a query block that contains *x* rows. If less than *x* rows are left in the result set, the remaining rows are returned.
- If the AR does not specify a query row set, the AS returns as many full rows in the query block as will fit in the query block size that is specified by the DRDA AR. You can configure the query row set size and query block size in either the client interface or the client database management system.

Performance considerations for DDM

These topics provide information to help you improve performance when using DDM and also provide some information about when to use something other than DDM to accomplish some functions.

- When a DDM file is specified on the CPYF command, optimal performance is obtained if the following items are all true:
  - The from-file is a logical or physical file and the to-file is a physical file.
  - FMTOPT is *NONE, *NOCHK, or not specified.
  - INCHAR, INCREL, ERRLVL, RCDDMT (*ALL), PRINT(*COPIED), PRINT(*EXCLD), SRCSEQ, TOKEY, SRCOPT, or FROMKEY parameter is not specified.
  - The from-file is not overridden with the POS keyword, other than *NONE or *START.
  - The to-file is not overridden with INHWRT(*YES).
- The **`Open Query File (OPNQRYF)`** command uses System/38 extensions to the DDM architecture. The System/38 DDM architecture extensions minimize DDM system processing time. These extensions are not used when:
  - The client system is neither a System/38 platform nor an IBM i operating system
  - The server system is neither a System/38 nor an IBM i operating system
- You can greatly reduce the amount of data transferred between systems if you use query functions such as the IBM i command OPNQRYF OPTIMIZE(*YES). However, for user-written applications, the amount of data exchanged between the systems is larger than that used to communicate using DDM with systems that are not running IBM i. The additional data provides IBM i extended DDM functions and also reduces client system DDM processing overhead. Using normal read, write, update, add, and delete operations as examples, consider the following items:
  - Standard DDM architecture DDM overhead data includes such information as a file identification, operation code, and simple result information. A user program read-by-key operation uses approximately 40 characters of DDM information in addition to the length of the key data. Data returned from the remote system uses approximately 32 characters of DDM information plus the length of the data file record.
  - System/38 DDM architecture extensions cause additional data overhead such as record format identification and a major portion of the I/O feedback area information. A user program read-by-key operation uses approximately 60 characters of DDM information in addition to the length of the key data. Data returned from the remote system uses approximately 80 characters of DDM information plus the length of the data file record. Normally, the additional length in data streams is not noticeable. However, as line activity increases, line utilization might peak sooner when using these extended data streams as opposed to standard DDM data streams.

- The target DDM job priority is controlled by the job class specified by the associated subsystem description routing entry. The following routing entry is normally the one used for all target (program start request) jobs:

  ```
  ADDRTGE ... PGM(*RTGDTA) ... CMPVAL(PGMEVOKE 29)
  ```

  The subsystems QBASE and QCMN, which are shipped with the IBM i operating system, have this routing entry.

  To have target DDM jobs in a subsystem run at a different priority than other APPC target jobs in the same subsystem, insert the following routing entry with the appropriate sequence number:

  ```
  ADDRTGE SBSD(xxx) SEQNBR(nnn) CMPVAL(QCNTEDDM 37)
          PGM(*RTGDTA) CLS(uuu)
  ```

  The class *uuu* is used to determine target job priority.
- To display records in a remote file, display station pass-through should be used whenever possible. Otherwise, direct record positioning should be used with the **Display Physical File Member (DSPPFM)** command.
- If a DDM user exit security program is a CL program and it creates an IBM i exception and an inquiry message that requires the server system operator to respond, both the user exit program and the client system job must wait for the response. Consider using the default system reply list by specifying INQMSGRPY(*SYSRPYL) for the TDDM job's description specified on the **Add Communications Entry (ADDCMNE)** command for that APPC remote location information.
- The WAIT and WAITFILE parameters, used on commands like **Allocate Object (ALCOBJ)** or **Receive Message (RCVMSG)**, have no effect on the client system program. These parameters function the same as they do when local files are accessed. The wait time values specified on commands run on the client system do not take effect on the client system; they affect only the server system and only if it is an IBM i operating system or a System/38 platform.

  Notes:

  1. The WAITFILE parameter of the create or change IBM i-Intersystems Communications Function (ICF) file command determines how long the APPC support will wait for session resources to become available when doing an acquire operation or a start function. The WAITFILE value is not used for sessions where the connection to the adjacent system is over a switched connection. An example is an SDLC switched line, an X.25 SVC line, an Ethernet line, or a token-ring connection. Using a switched connection, acquire operations and start functions do not time out.
  2. Because APPN sessions might cross multiple systems and lines to reach the remote system, the WAITFILE timer should be adjusted to allow more time in these cases. You should not specify *IMMED for the WAITFILE parameter if your application is running in a network configured to use APPN functions. The value you specify for this parameter is dependent on the size and type of the network.
- As for any LU session type 6.2 data exchange, certain SNA parameters can affect performance. These parameters include the path information unit size (MAXFRAME), the request/response unit size (MAXLENRU), SNA pacing (INPACING, OUTPACING), and for X.25, packet size and window size. In general, the larger the value used, the better the realized performance.
- **SNA path information unit size**

  The path information unit (PIU) is the size of the actual data transmission block between two systems. The MAXFRAME parameter on the Create Controller Description (APPC) (CRTCTLAPPC) or Create Controller Description (SNA Host) (CRTCTLHOST) command specifies the path information unit size the local system attempts to use. During session establishment, both systems determine which size is used, and it is always the smaller value. Other remote systems might have different PIU size considerations.
- **SNA response/request unit size**

  The response/request unit (RU) size (CRTMODD MAXLENRU) controls the amount of system buffering before fitting that data into the path information unit that is actually transmitted. In APPC, the transmit and receive RU lengths are negotiated during session establishment. Again, the negotiation results in the smallest value being used. Other remote systems have different RU size considerations.

- **SNA pacing values**

  The pacing value determines how many request/response units (RUs) can be received or sent before a response is required indicating buffer storage is available for more transmissions. During session establishment, both systems determine which size is used, and it is always the smaller value.

  In cases where both batch and interactive processing occur at the same time on the same communications line, IBM i job priority might be used to favor interactive processing over batch processing. In addition, reducing the value of pacing for a batch application and raising it for an interactive application might be necessary to provide a level of line activity priority for the interactive application.

  On the IBM i operating system, different pacing values can be specified through the creation of different MODES (Create Mode Description [CRTMODD] command) to the different applications. Other remote systems have different SNA pacing value considerations.

- **X.25 packet**

  An X.25 packet smaller than the MAXFRAME value adds data transmission time over a non-X.25 data link. In general, for X.25, the longer the MAXFRAME and the actual amount of data being transmitted, the greater this difference is. In the case of DDM, which adds DDM control information to the normal file record data, the packet size has an additional effect on the difference between local and remote file processing and between non-X.25 and X.25 data links.

  In cases of many deblocked DDM operations, the number of packets required to transmit data might become so high that packet processing overhead within the X.25 adapter affects performance significantly. Use the largest X.25 packet window size supported by the network and communicating products to maximize performance.

  When X.25 must be used to access remote files, successive transmission of many small deblocked records, such as less than 80 character records, might cause the X.25 adapter to expend a disproportionate amount of time processing X.25 packet characters as opposed to transmission of user data.

  In general, the overhead in processing X.25 packets results in less throughput than the use of a conventional line when identical line speeds are used and data transfer is in only one direction. When data is transferred at the same time in both directions, the advantages of X.25 duplex support is realized. On the System/38, the overall processing effect is minimal, because the overhead in processing the packets is done within the Integrated X.25 Adapter.

In general, the processing of remote files by using DDM is transparent to an application program or utility function, such as that provided by the Copy File (CPYF) command. However, additional time is required when accessing remote files by using a communications line. The performance difference between local file and remote file processing is proportional to the number of accesses to remote files, the data record length, and the line speed during a unit of performance measurement.

An additional difference between local and remote file processing is that the input or output operation to a local file might not result in an immediate physical disk operation because the system transfers blocks of data from the disk and writes blocks of data to the disk. There are times, then, that the user program accesses data within main storage and the physical I/O occurs at a different time. Therefore, to minimize the difference between local file and remote file performance, it is essential that knowledge of an application design and the amount and type of accesses to files be considered when determining which files are to be accessed remotely using DDM.

The additional time for each remote access is comprised of:

- Additional system processing to convert local system file interfaces to the DDM architecture interfaces
- Amount of data transmitted over the communications line
- Amount of remote system processing of the file operations
- Speed of the communications line

The communications line time accounts for most of the additional time, though the actual time is dependent on line speed and the amount of line activity during the DDM function.

As is true in non-DDM cases, local and remote system job priorities have the most significant effect on performance. On the IBM i operating system, the PRIORITY and TIME SLICE values of the class being used control job priority. The SDDM runs under the source job, and the TDDM runs under the class assigned to the APPC routing entry of the server system's subsystem. In applications that access multiple files, the best results are achieved when the most heavily accessed files are on the same system as the program that is running and the less heavily accessed files are on a remote system. Key considerations regarding the placement of files and application programs follow:

- The system having primary responsibility for file maintenance needs to be identified. In all cases of multiple systems applications, the best performance results if only one system is responsible for file maintenance. If an application program maintains the file through exclusive (nonshared) processing, best performance can be realized when the application program resides on the system with the file.

  In some cases, transmitting the file back to the local system might require:

  - An APPC program.
  - A program using remote DDM files.
  - The Copy File (CPYF) command by using DDM.
  - Object distribution SNDNETF and RCVNETF operations. In interactive applications, display station pass-through should be considered when the amount of display data transferred is significantly less than the amount of database file data that would be sent by using DDM.
- In cases where file placement requires movement of application processing to a remote system for best performance results, use of the Submit Remote Command (SBMRMTCMD) command should be considered. This works best in a batch processing input stream where each program waits for the preceding program to complete. The use of the SBMRMTCMD command is valid only when the source and server systems are IBM i or Systems/38 products. For example, assume that program A accesses local files. Program A would run on a local system. Program B accesses remote files. You can use the SBMRMTCMD command to run program B on the remote system.
- In cases where file maintenance is shared across systems, the best performance can be obtained if the file is placed on the system with the largest percentage of file update, add, and delete operations.

  In certain cases, a pair of source and target APPC programs can provide performance improvements over DDM. For example, assume 10 records are to be retrieved from the remote system. When DDM is used and record blocking cannot be used (for example, user program random input operation, sequential input for change, or use of the OVRDBF SEQONLY[*NO] command), assume 10 data transmissions are sent and 10 are received, for a total of 20 transmissions. User-written APPC programs can build additional intelligence into the data stream such that request for the data and receipt of the data can be done in two data transmissions instead of 20, one request for all records of customer 00010 and one response containing 10 records for customer 00010.

Consider two sample application processing techniques, batch file processing and interactive file processing.

Related concepts

Planning and design for DDM
There are several requirements that must be met for distributed data management (DDM) to be used properly.

Batch file processing with DDM

Consider these items when using batch file processing with DDM.

- When an application opens a local file for *sequential input only* or *output add*, the system uses blocking techniques to achieve maximum throughput. To ensure blocking is used for a remote file accessed by using DDM, do not use random record processing operations in the program but specify OVRDBF SEQONLY(*YES) against the DDM files opened by the program.
- Use of read and read-next operations in the high-level language (HLL) program to access the file maximizes the effect of the SEQONLY(*YES) specification.

- The use of random processing operations, such as chain operations of ILE RPG or start operations of ILE COBOL programming language, causes DDM to send deblocked operations across the communications line even if the application processes the file data sequentially. This results in significant differences between local and remote file processing.
- When simple physical file transfer is desired (all records transferred and no application processing of the data), use of DDM by using the **Copy File (CPYF)** command, or a user-written program using DDM with the **Override Database File (OVRDBF)** command SEQONLY(*YES number-of-records) specified, transfers the data more quickly than a user-written APPC program. The **Copy File** command and the DDM SEQONLY(*YES) support require less calls and returns between the program and APPC data management modules than does a standard ILE RPG or ILE COBOL APPC program.
- For ILE RPG or ILE COBOL sequential input-only applications, SEQONLY(*YES) should be specified with no *number of records* to achieve best throughput. For ILE RPG or ILE COBOL sequential output-only applications to keyed files, a large *number-of-records* value should be used.
- The **Send Network File (SNDNETF)** command can be considered as an alternative to DDM or user-written APPC programs when transferring all records within a file to a remote IBM i operating system. The **SNDNETF** command requires SNADS to be configured on the source and target IBM i operating system. If one or more intermediate systems are between the source and target IBM i operating systems, SNADS provides intermediate node routing of the data when correctly configured.
- Use of the **SNDNETF** command by using SNADS offers the advantages of transmitting one copy of the data to multiple users on one or more server systems through a multiple node network, and the time scheduled transmission of that data by using the SNADS distribution queue parameter.

  However, in addition to requiring SNADS to use the **SNDNETF** command, the server system user must also run the **Receive Network File (RCVNETF)** command to make the file usable on the server system. Use of DDM would not require this additional server system processing.

  In general, the file transmission times by using SNADS (user program DDM sequential file processing, the DDM **Copy File** command, and a user-written APPC program between two IBM i operating systems) are within 10% of each other. However, the use of the **SNDNETF** and **RCVNETF** commands to make a copy of the data usable on the server system does add total processing time over the other methods of file transfer.
- Because the **SNDNETF** command can transmit objects within a save file, the amount of data that is actually sent by using this technique might be less than that sent using the other techniques. If the database file data sent contains a significant number of duplicate character strings, use of the **Save Object (SAVOBJ)** command parameter DTACPR(*YES) (data compression) can significantly reduce the amount of data that is actually sent by using a SNADS distribution. However, if there are few duplicate character strings, there is little change in the amount of data sent.
- The IBM i file transfer subroutines might also be used to transfer an entire file between IBM i operating systems and an IBM i and a System/36 platform. These subroutines might be called from high-level language programs, and in some cases throughput is achieved similar to that by using DDM.

Interactive file processing with DDM

Consider these items when using interactive file processing with DDM.

- The greater the number of random file operations per unit of performance measurement, the greater the difference between local and remote file processing because each operation has to be sent separately across the communications line. DDM cannot anticipate the next operation.

  Using a simple inquiry application that produces display output, by using workstation subfile support (as an example), consider an application that does two random record retrievals per Enter key versus one that does 15 random record retrievals. The operator might barely notice a delay in response time when two records are retrieved. However, there would be a noticeable difference between local and remote response time when 15 records are retrieved randomly from the remote system.
- Use of display station pass-through should be considered when the amount of data transferred back to the local (client) system per unit of performance measurement significantly exceeds the amount of data presented on the display. Test results have shown that the total elapsed time between a single deblocked DDM get record operation and an equivalent user-written APPC operation is very close, with

APPC being slightly quicker. The DDM operation does require more processing seconds than the direct APPC interface.

Also, because each DDM operation always requires an operation result response from the remote system to ensure data integrity, user-designed partner APPC programs can offer an advantage for update, add, and delete operations by not validating the result of the operation until a later time.

- Be aware that additional time is needed when accessing files on other systems, particularly the time required for communications over the line. This should be considered when determining whether the file should be a local or remote file, especially if it is to be used often.

DDM conversation length considerations
Consider these items regarding the length of conversations when using DDM.

- Within a source job, if it is likely that a DDM conversation will be used by more than one program or DDM file, *KEEP is the value that should be specified for the DDMCNV job attribute. This saves the time and resources needed to start a target job (TDDM) each time a DDM file is accessed for the same location and mode combination within the source job.
- There is significant system and communications line overhead when a target DDM manager is started. The processing includes the APPC program start request, system type identification, and file open processing. However, if it is not necessary to keep the conversation active, *DROP should be specified for DDMCNV. When the local DDM file is closed, the session being used is released for use by other jobs using DDM or other APPC functions, such as SNADS and display station pass-through, to the same remote system.
- When the client and server systems are IBM i operating systems or System/38 platforms, the file input and output requests made by an application program use a form of DDM support that minimizes the amount of time needed to code and decode each request. This is accomplished by System/38 extensions to the DDM architecture.

  When the client and server systems are neither an IBM i operating system nor a System/38, then System/38 extensions to the DDM architecture are not used.

Examples: Application programming

These examples show how to use a distributed relational database for functional specification tasks and how to code for DDM-related tasks.

DRDA examples

This example application for distributed relational database use is written in RPG/400, COBOL/400, Java, and ILE C programming languages.

Example: Business requirement for distributed relational database

The application for the distributed relational database in this example is parts stock management in an automobile dealer or distributor network.

This program checks the level of stock for each part in the local part stock table. If this is below the re-order point, the program then checks on the central tables to see whether there are any existing orders outstanding and what quantity has been shipped against each order.

If the net quantity (local stock, plus orders, minus shipments) is still below the re-order point, an order is placed for the part by inserting rows in the appropriate tables on the central system. A report is printed on the local system.

Technical notes

Commitment control

This program uses the concept of local and remote logical units of work (LUW). Because this program uses remote unit of work, it is necessary to close the current LUW on one system (COMMIT) before beginning a new unit of work on another system.

Cursor repositioning

When an LUW is committed and the application connects to another database, all cursors are closed. This application requires the cursor reading the part stock file to be re-opened at the next part number. To achieve this, the cursor is defined to begin where the part number is greater than the current value of part number, and to be ordered by part number.

Note: This technique will not work if there are duplicate rows for the same part number.

Example: Program definitions

Here are the example program definitions for the parts stock management in an automobile dealer or distributor network.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
/******************************************************************************/
/* Local Database                                                             */
/******************************************************************************/
CREATE COLLECTION DRDA

CREATE TABLE DRDA/PART_STOCK (PART_NUM   CHAR(5)   NOT NULL,
                              PART_UM    CHAR(2)   NOT NULL,
                              PART_QUANT INTEGER   NOT NULL WITH
DEFAULT,
                              PART_ROP   INTEGER   NOT
NULL,
                              PART_EOQ   INTEGER   NOT
NULL,
                              PART_BIN   CHAR(6)   NOT NULL WITH
DEFAULT
                             )

CREATE UNIQUE INDEX DRDA/PART_STOCI ON DRDA/PART_STOCK (PART_NUM
ASC)

/******************************************************************************/
/* Remote Database                                                            */
/******************************************************************************/
CREATE COLLECTION DRDA

CREATE TABLE DRDA/PART_ORDER (ORDER_NUM  SMALLINT  NOT NULL,
                              ORIGIN_LOC CHAR(4)   NOT NULL,
                              ORDER_TYPE CHAR(1)   NOT NULL,
                              ORDER_STAT CHAR(1)   NOT NULL,
                              NUM_ALLOC  SMALLINT  NOT NULL WITH DEFAULT,
                              URG_REASON CHAR(1)   NOT NULL WITH DEFAULT,
                              CREAT_TIME TIMESTAMP NOT NULL,
                              ALLOC_TIME TIMESTAMP,
                              CLOSE_TIME TIMESTAMP,
                              REV_REASON CHAR(1)
                             )

CREATE UNIQUE INDEX DRDA/PART_ORDEI ON DRDA/PART_ORDER (ORDER_NUM
ASC)

CREATE TABLE DRDA/PART_ORDLN (ORDER_NUM  SMALLINT  NOT NULL,
                              ORDER_LINE SMALLINT  NOT NULL,
                              PART_NUM   CHAR(5)   NOT NULL,
                              QUANT_REQ  INTEGER   NOT
NULL,
                              LINE_STAT  CHAR(1)   NOT
NULL
                             )

CREATE UNIQUE INDEX PART_ORDLI ON DRDA/PART_ORDLN (ORDER_NUM ASC,
                                                   ORDER_LINE
ASC)

CREATE TABLE DRDA/SHIPMENTLN (SHIP_NUM   SMALLINT  NOT NULL,
                              SHIP_LINE  SMALLINT  NOT NULL,
                              ORDER_LOC  CHAR(4)   NOT NULL,
                              ORDER_NUM  SMALLINT  NOT NULL,
                              ORDER_LINE SMALLINT  NOT NULL,
                              PART_NUM   CHAR(5)   NOT NULL,
                              QUANT_SHIP INTEGER   NOT
NULL,
                              QUANT_RECV INTEGER   NOT NULL WITH
DEFAULT
                             )

CREATE UNIQUE INDEX SHIPMENTLI ON DRDA/SHIPMENTLN (SHIP_NUM
ASC,
                                                   SHIP_LINE ASC)
```

Figure 32. Creating a collection and tables

```
/******************************************************************************/
/* Local Database                                                             */
/******************************************************************************/
/******************************************************************************/
/* PART_STOCK                                                                 */
/******************************************************************************/
INSERT INTO DRDA/PART_STOCK VALUES('14020','EA',038,050,100,' ')

INSERT INTO DRDA/PART_STOCK VALUES('14030','EA',043,050,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('14040','EA',030,020,030,' ')

INSERT INTO DRDA/PART_STOCK VALUES('14050','EA',010,005,015,' ')

INSERT INTO DRDA/PART_STOCK VALUES('14060','EA',110,045,090,' ')

INSERT INTO DRDA/PART_STOCK VALUES('14070','EA',130,080,160,' ')

INSERT INTO DRDA/PART_STOCK VALUES('18020','EA',013,025,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('18030','EA',015,005,010,' ')

INSERT INTO DRDA/PART_STOCK VALUES('21010','EA',029,030,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('24010','EA',025,020,040,' ')

INSERT INTO DRDA/PART_STOCK VALUES('24080','EA',054,050,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('24090','EA',030,025,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('24100','EA',020,015,030,' ')

INSERT INTO DRDA/PART_STOCK VALUES('24110','EA',052,050,080,' ')

INSERT INTO DRDA/PART_STOCK VALUES('25010','EA',511,300,600,' ')

INSERT INTO DRDA/PART_STOCK VALUES('36010','EA',013,005,010,' ')

INSERT INTO DRDA/PART_STOCK VALUES('36020','EA',110,030,060,' ')

INSERT INTO DRDA/PART_STOCK VALUES('37010','EA',415,100,200,' ')

INSERT INTO DRDA/PART_STOCK VALUES('37020','EA',010,020,040,' ')

INSERT INTO DRDA/PART_STOCK VALUES('37030','EA',154,055,060,' ')

INSERT INTO DRDA/PART_STOCK VALUES('37040','EA',223,120,120,' ')

INSERT INTO DRDA/PART_STOCK VALUES('43010','EA',110,020,040,' ')

INSERT INTO DRDA/PART_STOCK VALUES('43020','EA',067,050,050,' ')

INSERT INTO DRDA/PART_STOCK VALUES('48010','EA',032,030,060,' ')
```

Figure 33. Inserting data into the tables

```
/******************************************************************************/
/* Remote Database                                                            */
/******************************************************************************/
/******************************************************************************/
/* PART_ORDER TABLE                                                           */
/******************************************************************************/
INSERT INTO DRDA/PART_ORDER VALUES(1,'DB2B','U','O',0,' ','1991-03-12-17.00.00',
                              NULL,NULL,NULL)

INSERT INTO DRDA/PART_ORDER VALUES(2,'SQLA','U','O',0,' ','1991-03-12-17.01.00',
                              NULL,NULL,NULL)

INSERT INTO DRDA/PART_ORDER VALUES(3,'SQLA','U','O',0,' ','1991-03-12-17.02.00',
                              NULL,NULL,NULL)

INSERT INTO DRDA/PART_ORDER VALUES(4,'SQLA','U','O',0,' ','1991-03-12-17.03.00',
                              NULL,NULL,NULL)

INSERT INTO DRDA/PART_ORDER VALUES(5,'DB2B','U','O',0,' ','1991-03-12-17.04.00',
                              NULL,NULL,NULL)


/******************************************************************************/
/*  PART_ORDLN TABLE                                                          */
/******************************************************************************/
INSERT INTO DRDA/PART_ORDLN VALUES(1,1,'24110',005,'O')

INSERT INTO DRDA/PART_ORDLN VALUES(1,2,'24100',021,'O')
```

DRDA example: RPG program

This example program is written in the RPG programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
100
****************************************************************                        00/00/00
200       *
*                        00/00/00
300       *   DESCRIPTIVE NAME = D-DB SAMPLE APPLICATION
*                        00/00/00
400       *                       REORDER POINT PROCESSING
*                        00/00/00
500       *                       IBM i
*                        00/00/00
600       *
*                        00/00/00
700       *   FUNCTION =  THIS MODULE PROCESSES THE PART_STOCK TABLE AND
*                        00/00/00
800       *               FOR EACH PART BELOW THE ROP (REORDER POINT)
*                        00/00/00
900       *               CREATES A SUPPLY ORDER AND PRINTS A REPORT.
*                        00/00/00
1000      *
*                        00/00/00
1100      *
*                        00/00/00
1200      *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*                        00/00/00
1300      *
*                        00/00/00
1400      *              LOCADB         LOCAL DB NAME
*                        00/00/00
1500      *              REMODB         REMOTE DB NAME
*                        00/00/00
1600      *
*                        00/00/00
1700      *    TABLES =  PART-STOCK       - LOCAL
*                        00/00/00
1800      *             PART_ORDER       - REMOTE
*                        00/00/00
1900      *             PART_ORDLN       - REMOTE
*                        00/00/00
2000      *             SHIPMENTLN       - REMOTE
*                        00/00/00
2100      *
*                        00/00/00
2200      *   INDICATORS =   *IN89       - '0' ORDER HEADER  NOT  DONE
*                        00/00/00
2300      *                                '1' ORDER HEADER  IS   DONE
*                        00/00/00
2400      *                  *IN99       - '1' ABNORMAL END (SQLCOD<0)
*                        00/00/00
2500      *
*                        00/00/00
2600      *   TO BE COMPILED WITH COMMIT(*CHG) RDB(remotedbname)
*                        00/00/00
2700      *
*                        00/00/00
2800      *   INVOKE BY : CALL DDBPT6RG PARM(localdbname remotedbname)
*                        00/00/00
2900      *
*                        00/00/00
3000      *   CURSORS WILL BE CLOSED IMPLICITLY (BY CONNECT) BECAUSE
*                        00/00/00
3100      *   THERE IS NO REASON TO DO IT EXPLICITLY
*                        00/00/00
3200      *
*                        00/00/00
3300
****************************************************************                        00/00/00
3400
*                                                                                       00/00/00
3500      FQPRINT  O   F      33     OF
PRINTER                                              00/00/00
3600
F*                                                                                      00/00/00
3700
I*                                                                                      00/00/00
3800      IMISC
DS                                                                         00/00/00
3900      I                                    B   1
20SHORTB                                   00/00/00
4000      I                                    B   3
60LONGB                                    00/00/00
4100      I                                    B   7
80INDNUL                                   00/00/00
4200      I                                        9  13
PRTTBL                                   00/00/00
4300      I                                      14  29
```

DRDA example: COBOL program

This example program is written in the COBOL programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
100        IDENTIFICATION DIVISION.
200       *------------------------
300        PROGRAM-ID. DDBPT6CB.
00/00/00
400       ****************************************************************
00/00/00
500       *   MODULE NAME = DDBPT6CB
00/00/00
600       *
700       *   DESCRIPTIVE NAME = D-DB SAMPLE APPLICATION
800       *                      REORDER POINT PROCESSING
900       *                      IBM i
00/00/00
1000      *                      COBOL
1100      *
1200      *   FUNCTION =  THIS MODULE PROCESSES THE PART_STOCK TABLE AND
1300      *               FOR EACH PART BELOW THE ROP (REORDER POINT)
1400      *               CHECKS THE EXISTING ORDERS AND SHIPMENTS,
00/00/00
1500      *               CREATES A SUPPLY ORDER AND PRINTS A REPORT.
00/00/00
1600      *
1700      *      DEPENDENCIES = NONE
00/00/00
1800      *
1900      *      INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
2000      *
2100      *              LOCAL-DB       LOCAL DB NAME
00/00/00
2200      *              REMOTE-DB      REMOTE DB NAME
00/00/00
2300      *
2400      *   TABLES =  PART-STOCK       - LOCAL
00/00/00
2500      *             PART_ORDER       - REMOTE
00/00/00
2600      *             PART_ORDLN       - REMOTE
00/00/00
2700      *             SHIPMENTLN       - REMOTE
00/00/00
2800      *
00/00/00
2900      *   CRTSQLCBL  SPECIAL PARAMETERS
00/00/00
3000      *    PGM(DDBPT6CB) RDB(remotedbname) OPTION(*APOST *APOSTSQL)
00/00/00
3100      *
00/00/00
3200      *   INVOKE BY : CALL DDBPT6CB PARM(localdbname remotedbname)
00/00/00
3300      *
00/00/00
3400      ****************************************************************
00/00/00
3500        ENVIRONMENT DIVISION.
3600       *---------------------
3700        INPUT-OUTPUT SECTION.
3800        FILE-CONTROL.
3900            SELECT RELAT  ASSIGN TO PRINTER-
QPRINT.                                          00/00/00
4000        DATA DIVISION.
4100       *--------------
4200        FILE SECTION.
4300
*-------------                                                                        00/00/00
4400        FD  RELAT
4500            RECORD CONTAINS 33 CHARACTERS
4600            LABEL RECORDS ARE OMITTED
4700            DATA RECORD IS REPREC.
4800        01  REPREC          PIC  X(33).
4900        WORKING-STORAGE SECTION.
5000
*------------------------                                                             00/00/00
5100       *    PRINT LINE
DEFINITIONS                                                          00/00/00
5200        01  LINE0           PIC  X(33) VALUE  SPACES.
5300        01  LINE1           PIC  X(33) VALUE
```

Figure 36. COBOL program example

```
5400              '***** ROP PROCESSING REPORT *****'.
5500        01  LINE2.
5600          05  FILLER        PIC  X(18) VALUE  '   ORDER NUMBER = '.
5700          05  MASK0         PIC  ZZZ9
```

DRDA example: C program using embedded SQL

This example program is written in the C programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
/******************************************************************************/
/*   PROGRAM NAME: SAMPEMBC                                                   */
/*                                                                            */
/*   DESCRIPTIVE NAME: Sample embedded C application using DRDA               */
/*                                                                            */
/*   FUNCTION: This module processes the PART_STOCK table and                 */
/*             for each part below the ROP (REORDER POINT)                    */
/*             creates a supply order.                                        */
/*                                                                            */
/*   LOCAL TABLES:  PART_STOCK                                                */
/*                                                                            */
/*   REMOTE TABLES: PART_ORDER, PART_ORDLN, SHIPMENTLN                        */
/*                                                                            */
/*   COMPILE OPTIONS:                                                         */
/*   CRTSQLCI   OBJ(SAMPEMBC) COMMIT(*CHG) RDB(rdbname) OBJTYPE(*PGM)         */
/*              RDBCNNMTH(*RUW)                                               */
/*                                                                            */
/*   INVOKED BY:                                                              */
/*   CALL PGM(SAMPEMBC) PARM('lcldbname' 'rmtdbname')                         */
/******************************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

EXEC SQL INCLUDE SQLCA;

/******************************************************************************/
/* SQL Variables                                                              */
/******************************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
  char  loc[4] = "SQLA";            /* dealer's database name                 */
  char  remote_db[18] = "              ";
                                    /* sample remote database                 */
  char  local_db[18] = "               ";
                                    /* sample local  database                 */
  char  part_table[5] = " ";        /* part number in table part_stock        */
  long  quant_table;                /* quantity in stock, tbl part_stock      */
  long  rop_table;                  /* reorder point    , tbl part_stock      */
  long  eoq_table;                  /* reorder quantity , tbl part_stock      */
  short next_num;                   /* next order nbr,table part_order        */
  short ord_table;                  /* order nbr.       , tbl order_line      */
  short orl_table;                  /* order line       , tbl order_line      */
  long  qty_table;                  /* ordered quantity , tbl order_line      */
  long  line_count = 0;             /* total number of order lines            */
  short ind_null;                   /* null indicator for qty_table           */
  short contl = 0;                  /* continuation line, tbl order_line      */
EXEC SQL END DECLARE SECTION;

/******************************************************************************/
/* Other Variables                                                            */
/******************************************************************************/
char first_time, what_time;
long  qty_rec = 0, qty_req = 0;

EXEC SQL WHENEVER SQLERROR GOTO error_tag;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
```

Figure 37. C program example using embedded SQL

```
/******************************************************************************/
/* Function Declarations                                                      */
/******************************************************************************/
/******************************************************************************/
/* Function For Declaring Cursors                                             */
/******************************************************************************/
declare_cursors() {

  /* SQL Cursor declaration and reposition for local UW                       */
  EXEC SQL DECLARE NEXT_PART CURSOR FOR
                             SELECT PART_NUM, PART_QUANT, PART_ROP, PART_EOQ
                             FROM DRDA/
PART_STOCK
                             WHERE PART_ROP > PART_QUANT AND
                                   PART_NUM > :part_table
                             ORDER BY PART_NUM;

  /* SQL Cursor declaration and connect for RUW                               */
  EXEC SQL DECLARE NEXT_OLINE CURSOR FOR
                              SELECT A.ORDER_NUM, ORDER_LINE, QUANT_REQ
                              FROM DRDA/PART_ORDLN A,
                                   DRDA/PART_ORDER
B
                              WHERE PART_NUM  = :part_table AND
                                    LINE_STAT  <> 'C' AND
```

DRDA example: Java program

This example program is written in the Java programming language.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
/******************************************************************************/
/*   PROGRAM NAME: SampJava                                                   */
/*                                                                            */
/*   DESCRIPTIVE NAME: Sample Java application using DRDA                     */
/*                                                                            */
/*   FUNCTION: This module processes the PART_STOCK table and                 */
/*             for each part below the ROP (REORDER POINT)                    */
/*             creates a supply order.                                        */
/*                                                                            */
/*   LOCAL TABLES:  PART_STOCK                                                */
/*                                                                            */
/*   REMOTE TABLES: PART_ORDER, PART_ORDLN, SHIPMENTLN                        */
/*                                                                            */
/*   COMPILE OPTIONS:                                                         */
/*   javac SampJava.java                                                      */
/*                                                                            */
/*   INVOKED BY:                                                              */
/*   java SampJava lcldbname rmtdbname                                        */
/******************************************************************************/
import java.sql.*;

public class SampJava {
  private static String JDBCDriver = "com.ibm.db2.jcc.DB2Driver";
  private static String part_table = " ";/* part number in table part_stock   */
  private static long line_count = 0;/* total number of order lines           */
  private static long eoq_table = 0;/* reorder quantity , tbl part_stock      */
  private static long quant_table = 0;/* quantity in stock, tbl part_stock    */
  private static long rop_table = 0;/* reorder point    , tbl part_stock      */
  private static int contl = 0;     /* continuation line, tbl order_line      */
  private static short next_num = 0;/* next order nbr,table part_order        */

  /****************************************************************************/
  /* Method For Reseting Environment                                          */
  /****************************************************************************/
  private static void resetTables(Connection rmtConn) throws SQLException
{

    Statement stmt1 = rmtConn.createStatement();

    /* Clean up for rerunability in test environment                          */
    stmt1.executeUpdate("DELETE FROM DRDA.PART_ORDLN WHERE ORDER_NUM IN " +
                        "       (SELECT ORDER_NUM FROM DRDA.PART_ORDER " +
                        "                         WHERE ORDER_TYPE = 'R')");
    stmt1.executeUpdate("DELETE FROM DRDA.PART_ORDER WHERE ORDER_TYPE = 'R'");
    stmt1.close();
    rmtConn.commit();

  } /* function delete_for_rerun                                              */
```

```
  /****************************************************************************/
  /* Method For Calculating Order Quantity                                    */
  /****************************************************************************/
  private static void calculateOrderQuantity(Connection lclConn, Connection rmtConn, String
loc)
throws SQLException {
    PreparedStatement prpStmt1;
    PreparedStatement prpStmt2;
    ResultSet rsltSet1;
    ResultSet rsltSet2;
    short ord_table = 0;            /* order nbr.       , tbl order_line      */
    short orl_table = 0;            /* order line       , tbl order_line      */

    prpStmt1 = lclConn.prepareStatement("SELECT PART_NUM, PART_QUANT, PART_ROP, PART_EOQ " +
                                        "       FROM DRDA.PART_STOCK WHERE PART_ROP >
PART_QUANT AND " +
                                        "       PART_NUM > ? ORDER BY PART_NUM");
    prpStmt1.setString(1,part_table);
    rsltSet1 = prpStmt1.executeQuery();
    if (rsltSet1.next() == false) {
      System.out.println("--------------------------------");
      System.out.println("NUMBER OF LINES CREATED = " + line_count);
      System.out.println("--------------------------------");
      System.out.println("*****  END OF PROGRAM  *********");
      rop_table = 0;                /* no (more) orders to process            */
    }
    else {
      /* available qty = Stock qty + qty in order - qty received              */
      part_table = rsltSet1.getString(1);
      quant_table = rsltSet1.getLong(2);
      rop_table = rsltSet1.getLong(3);
      eoq_table = rsltSet1.getLong(4);
      long qty_rec = 0;
```

Example: Program output

Here is the program output for the parts stock management example.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
*****  ROP PROCESSING  *********
ORDER NUMBER = 6
--------------------------------
LINE     PART       QTY
NBR      NBR        REQUESTED
--------------------------------
1        14020      100
2        14030      50
3        18020      50
4        21010      50
5        37020      40
--------------------------------
NUMBER OF LINES CREATED = 5
--------------------------------
*****  END OF PROGRAM  *********
```

Figure 38. Example: Program output

DDM examples

The examples in this topic collection are based on representative application programs that might be used for processing data both on the local IBM i platforms and on one or more remote systems.

The first example is a simple inquiry application, and the second example is an order entry application. The third example accesses multiple files on multiple IBM i platforms. The fourth example accesses multiple IBM i platforms and a System/36.

The coding for each of these examples and tasks has one or two parts:

- Coding, shown in pseudocode form, not related to DDM but used to build the programming environment. The examples show you the task steps needed, independent of the language you use for your applications. You can write or adapt your programs in your language with the necessary coding to perform these or similar tasks.
- Coding, mostly done in CL, related to communicating with the other systems using DDM in the network.

Communications setup for DDM examples and tasks

These topics describe the network in which DDM is used for these task examples.

The network contains a central system in Philadelphia (a IBM i product), two remote IBM i platforms in Toronto and New York City, a System/38 in Chicago, and a System/36 in Dallas. The Advanced Program-to-Program Communication (APPC) network for these systems was configured with the values shown in the following figure.

In this set of task examples, the System/36 has Release 5 of DDM installed and DDM with the compatible PTF installed. The System/38 has Release 8 of CPF installed with the DDM licensed program and the compatible program temporary fix (PTF) change applied to the system.

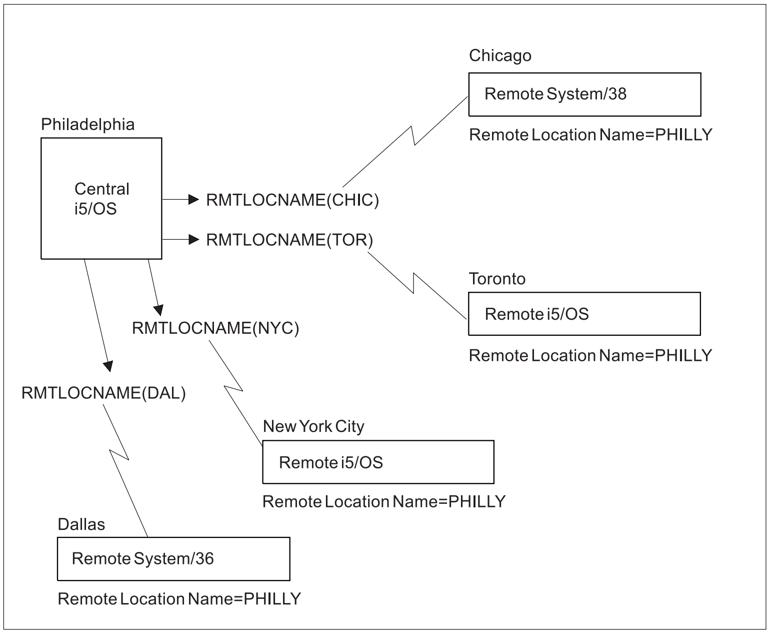


Figure 39. DDM network used in ORDERENT application tasks

DDM example 1: Simple inquiry application

This first example shows how multiple locations in a customer's business can use their own primary files to process the same inquiry application on their systems.

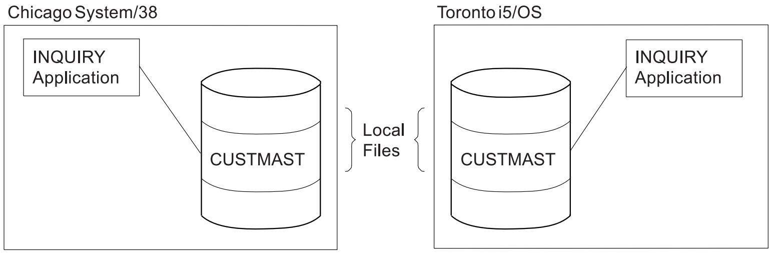


Figure 40. Two non-DDM servers doing local inquiries

Without DDM, the two locations shown here (Chicago and Toronto) have their own primary file (CUSTMAST), both with different and duplicate levels of information.

The following program (in pseudocode form) is run at each location to access its own primary file named CUSTMAST.

```
       Open CUSTMAST
LOOP:  Prompt for CUSTNO
       If function 1, go to END
       Get customer record
       Display
       Go to LOOP
END:   Close CUSTMAST
       RETURN
```

Using DDM, the CUSTMAST files are consolidated into one file at a centralized location (Philadelphia, in these examples), and then the local files in Chicago and Toronto can be deleted. The inquiry program used at each remote location and at the central location to access that file is identical to the program used previously.

To perform remote inquiries without changing the program, each of the remote locations need only create a DDM file and use an override command:

```
CRTDDMF   FILE(INQ)  RMTFILE(CUSTMAST)  RMTLOCNAME(PHILLY)

OVRDBF  FILE(CUSTMAST) TOFILE(INQ)
```

The DDM file points to the Philadelphia system as the server system and to the CUSTMAST file as the remote file. The same values for this command can be used at each remote location if they also have a remote location named PHILLY.

Because CUSTMAST is the file name used in the program, the Override with Database File (OVRDBF) command must be used to override the nonexistent CUSTMAST file with the DDM file INQ. (If the CUSTMAST file still exists on the local system, the override is needed to access the central system's primary file; without it, the local file is accessed.)

The figure below shows the same two systems accessing the centralized CUSTMAST file by using their DDM files, each named INQ.

An alternative to this approach is to leave the CUSTMAST files on the Chicago and Toronto systems and use them for nonessential inquiries, such as name and address, and use the central CUSTMAST file in Philadelphia for any changes. The CUSTMAST files on the Chicago and Toronto systems can be changed periodically to the current level of the primary file on the Philadelphia system.

This alternative method is used in the ORDERENT application example.

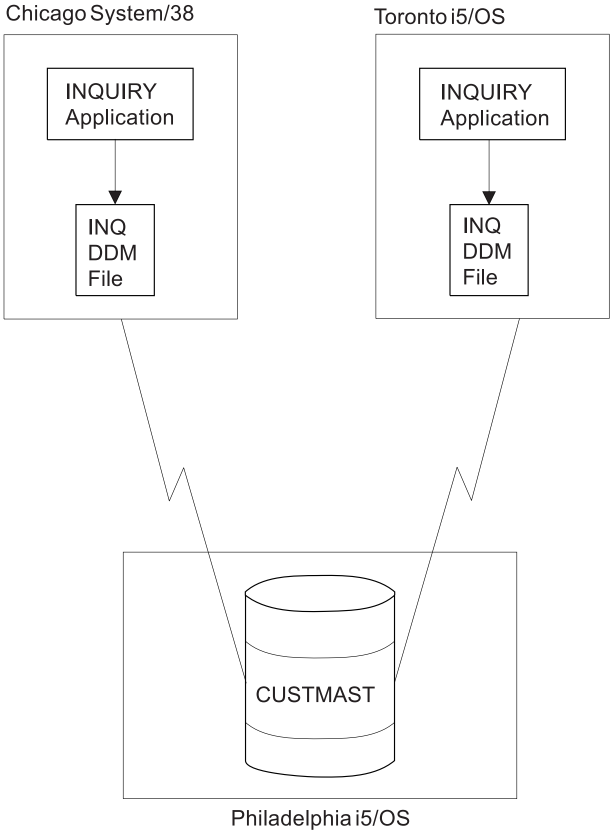


Figure 41. Two DDM servers doing remote inquiries

DDM example 2: ORDERENT application

This second example shows how multiple locations in a customer's business can process the same order entry application using DDM.

The first task in this example shows how to use DDM to put copies of the same application program on remote systems with one primary file at a central location. The second task in this example shows how to use DDM to copy a file to a remote system.

DDM example 2: Central system ORDERENT files

At the central site of Philadelphia, the four files shown in the figure are being used by the ORDERENT application program.

At the central system, the CUSTMAST file is a physical file that is the primary file of customer data for all locations. The CUSTMST2 file is a logical file that is based on the CUSTMAST physical file. Using a logical file at the central system provides at least two advantages:

- The same program, ORDERENT, can be used *without* change by the central system and by each of the remote systems.
- The data can be accessed through a separate file and cannot keep a customer's primary record locked for the duration of the order.

The four files at the central site are used as follows:

- The CUSTMAST file contains all the data about all its customers. After a customer order is completed, the CUSTMAST file is changed with all the new information provided by that order.
- The CUSTMST2 file, which is a logical file at the central system, is used at the beginning of a customer order. When an operator enters a customer number, the program reads the customer data from the CUSTMST2 logical file, but the data actually comes from the primary file, CUSTMAST.
- The INVEN file contains the current quantities of all items available for sale to customers. When the operator enters an item number and quantity ordered, the corresponding primary item in the INVEN file is changed.
- The DETAIL file is a list of all the individual items ordered; it contains a record for each item and quantity ordered by customers.

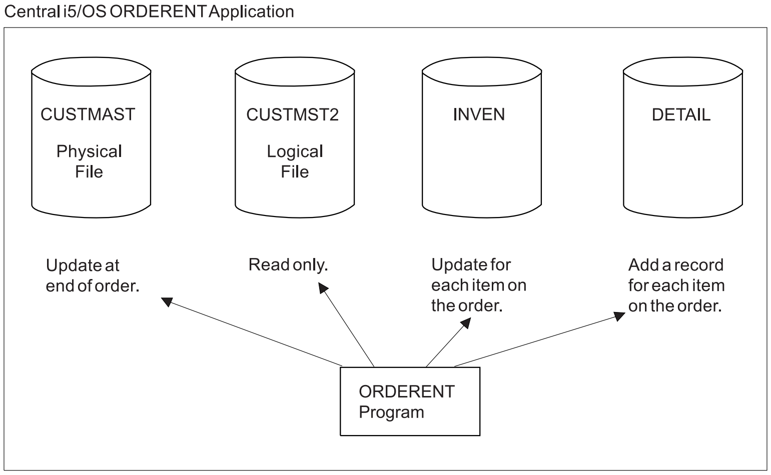

Central i5/OS ORDERENT Application
CUSTMAST
Physical File
CUSTMST2
Logical File
INVEN
DETAIL
Update at end of order.
Read only.
Update for each item on the order.
Add a record for each item on the order.
ORDERENT Program

Figure 42. Files used by central system ORDERENT program

DDM example 2: Description of ORDERENT program

Initially, the ORDERENT program exists only in library PGMLIB on the central system (in Philadelphia).

This program does the following items:

- When an order entry operator enters a customer number, ORDERENT reads the customer number, then reads the first member of file CUSTMST2 in the PGMLIB library to find the customer name, address, and other information. The retrieved information is displayed to the operator, and the program asks for an item number and quantity desired.

- When the operator enters an item number and quantity desired and presses the Enter key, the program changes the corresponding primary item in the first member of the INVEN file, and it adds a record to the DETAIL file for each item and quantity entered. The program continues asking for another item number and quantity until the operator ends the program.
- When the operator ends the program, the file CUSTMAST is changed with the information for the entire order. (See the pseudocode of ORDERENT for details.)

For the following examples, it is assumed that all users on the remote systems who need to access CUSTMAST in Philadelphia already have authority to do so, and that those who do not need authority do not have it. In these examples, the IBM i platform in Chicago does not have a compiler.

If we want this program to be used at all the remote locations that also stock a physical inventory, the program needs to be sent to each of the remote systems. We can assume that each of the remote systems has its own inventory and primary files INVEN, DETAIL, and CUSTMST2 (which is a copy of CUSTMAST). How the program can be sent to a remote system is described in "DDM example 2: Transferring a program to a server system" on page 308.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
   Pseudocode for ORDERENT Program
   .
   .
   .
DECLARE CUSTMAST CHANGE
      * Declare file CUSTMAST and allow changing.
DECLARE CUSTMST2 READ
      * Declare file CUSTMST2 as read only.
DECLARE INVEN CHANGE
      * Declare inventory file INVEN and allow changing.
DECLARE DETAIL OUTPUT
      * Declare file DETAIL as output only.
   .
   .
   .
Open CUSTMAST, CUSTMST2, INVEN, and DETAIL files
      * Begin program.
   Show order entry display asking for CUSTNO.
         * Order entry operator enters CUSTNO.
   If function key, go to End.
   Read CUSTNO from display.
      For CUSTNO, return NAME, ADDR, and other
      information from CUSTMST2 file.
   Show NAME, ADDR, and other information on display.
   LOOP: Display 'Item Number ___     Quantity Desired ____'.
         * Order entry operator enters item number and quantity.
      Read ITEMNO and Quantity Desired from display.
      If ITEMNO = 0 then go to LOOPEND.
         Change INVEN with ITEMNO and Quantity Desired.
         Write an item record to the DETAIL file.
      Go to LOOP.
   LOOPEND:  For CUSTNO, change CUSTMAST using
             information in file INVEN.
   End
      * Program has ended.
Close CUSTMAST, CUSTMST2, INVEN, and DETAIL files.
RETURN
```

DDM example 2: Remote system ORDERENT files

The ORDERENT program remains the same at all locations, but the CUSTMST2 file is now a *copy* of the central system's customer primary file CUSTMAST.

By using CUSTMST2 whenever possible for data that does not change often, the amount of communications time needed to process each order entry request can be minimized. The remote ORDERENT program reads the local CUSTMST2 file at the beginning of each order, and then, using DDM, updates the CUSTMAST file on the central system only when an order has been completed.

The other two files, INVEN and DETAIL, have the same functions on each remote system as on the central system.

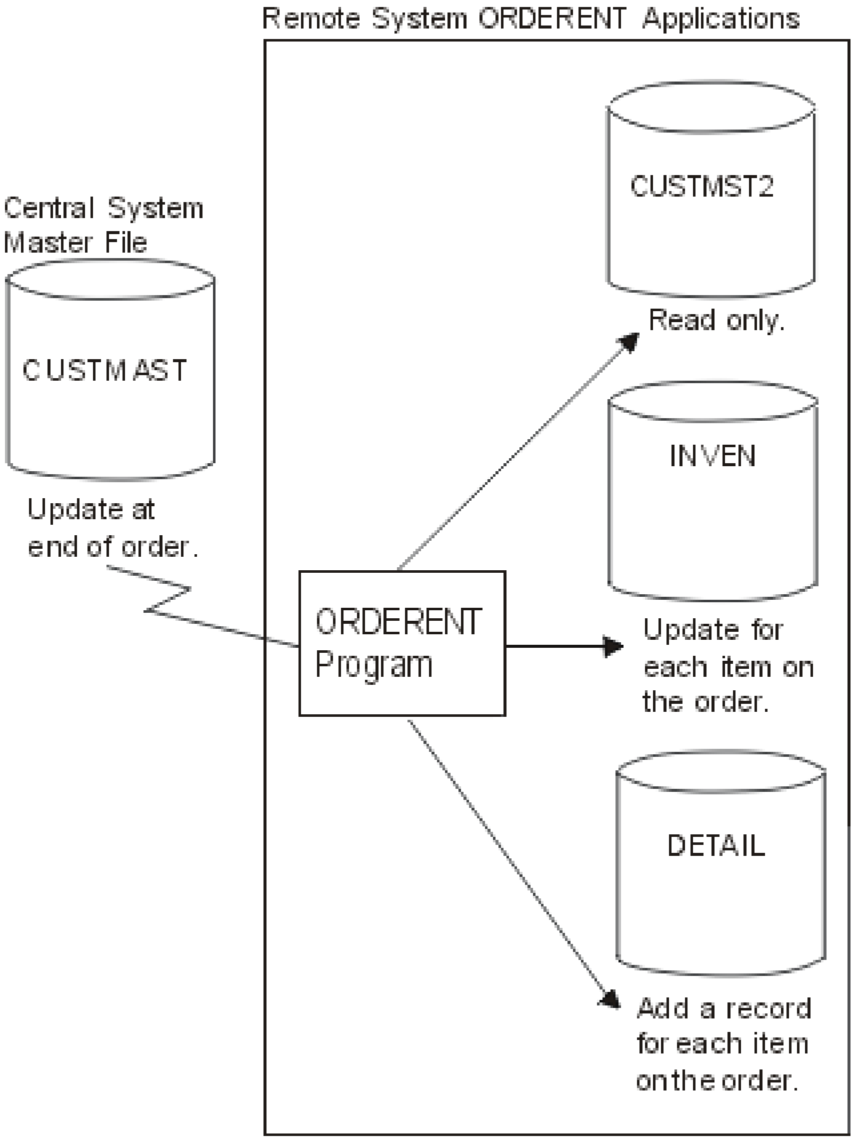

Remote System ORDERENT Applications
Central System Master File
CUSTMAST
Update at end of order.
ORDERENT Program
CUSTMST2
Read only.
INVEN
Update for each item on the order.
DETAIL
Add a record for each item onthe order.

Figure 43. Files used by remote ORDERENT programs

The CUSTMAST file is changed by all locations and contains the most current information for each customer (for constantly changing data such as the customer's account balance). The CUSTMST2 file, which is used for reading data that changes only occasionally (such as name and address), should be changed periodically (once a week, for example), by recopying the CUSTMAST file into it. Task 2 of this example explains one way to do this.

DDM example 2: Transferring a program to a server system

In this task, the central system in the DDM network, located in Philadelphia, sends a program named ORDERENT to a remote System/38 in Chicago.

The program ORDERENT is transferred from the Philadelphia system to the user in Chicago whose user ID is ANDERSON CHICAGO, and then the program is set up so that ORDERENT in Chicago changes the CUSTMAST file in library PGMLIB on the central system in Philadelphia. The read-only function is performed against the local file (in Chicago) and the change is done in the remote file (in Philadelphia).

For this task, two methods are shown for transferring the ORDERENT program in Philadelphia to the remote system in Chicago. Basically, the same sets of commands are used in both methods, except that the second group of commands used in the pass-through method are embedded in **Submit Remote Command (SBMRMTCMD)** commands used in the SBMRMTCMD command method.

- The first method uses pass-through and object distribution, allowing the operator on the client system to set up both systems without involving the server system operator or using the **SBMRMTCMD** command. This method can be used only for IBM i or System/38.
- The second method uses the **SBMRMTCMD** command because, in this task, the server system is a System/38. (The **SBMRMTCMD** command can be used when the server system is an IBM i or a System/38.)

DDM example 2: Pass-through method

One set of commands is entered on the client system, a pass-through session is started with the server system, and a second set of commands is entered on the client system and run on the server system.

The following commands are issued on the client system in Philadelphia:

```
CRTSAVF  FILE(TRANSFER)
SAVOBJ   OBJ(ORDERENT)  LIB(PGMLIB)  SAVF(TRANSFER)
   UPDHIST(*NO)  DTACPR(*YES)
SNDNETF  FILE(TRANSFER)  TOUSRID(ANDERSON CHICAGO)
```

Next, a pass-through session is started between the Philadelphia and Chicago systems with the Begin Pass-Through (BGNPASTHR) command. The session is used at the client system to enter the following commands, which are run on the server system:

```
CRTSAVF  FILE(RECEIVE)
RCVNETF  FROMFILE(TRANSFER)  TOFILE(RECEIVE)
CRTLIB   LIB(PGMLIB)
RSTOBJ   OBJ(ORDERENT)  SAVLIB(PGMLIB)  SAVF(RECEIVE)
CRTDDMF  FILE(CUSTMAST.PGMLIB)  RMTFILE(*NONSTD 'PGMLIB/CUSTMAST')
   DEVD(PHILLY)
```

These commands create a save file named RECEIVE, into which the TRANSFER file is copied after it is received as a network file from the client system in Philadelphia. A library is created on the Chicago system and the RECEIVE file is restored as the ORDERENT program in the newly created library named PGMLIB. Lastly, a DDM file is created on the Chicago system, which allows the Chicago system to access the CUSTMAST file on the Philadelphia system (remote location named PHILLY).

DDM example 2: SBMRMTCMD command method

Commands needed to accomplish the task are entered at the client system. The client system sends commands that are needed on the target IBM i operating system by using the Submit Remote Command (SBMRMTCMD) command between the systems.

The following commands are issued on the client system in Philadelphia to send the ORDERENT program to the server system in Chicago:

```
CRTSAVF  FILE(TRANSFER)
SAVOBJ   OBJ(ORDERENT)  LIB(PGMLIB)  SAVF(TRANSFER)
   UPDHIST(*NO)
SNDNETF  FILE(TRANSFER)  TOUSRID(ANDERSON CHICAGO)
CRTDDMF  FILE(CHICAGO)  RMTFILE(xxxxx)  RMTLOCNAME(CHIC)
SBMRMTCMD  CMD('CRTSAVF  FILE(RECEIVE)')  DDMFILE(CHICAGO)
SBMRMTCMD  CMD('RCVNETF  FROMFILE(TRANSFER)
   TOFILE(RECEIVE)')  DDMFILE(CHICAGO)
SBMRMTCMD  CMD('CRTLIB  LIB(PGMLIB)')  DDMFILE(CHICAGO)
SBMRMTCMD  CMD('RSTOBJ  OBJ(ORDERENT)  SAVLIB(PGMLIB)
   SAVF(RECEIVE)')  DDMFILE(CHICAGO)
SBMRMTCMD  CMD('CRTDDMF  FILE(CUSTMAST.PGMLIB)
   RMTFILE(*NONSTD ”PGMLIB/CUSTMAST”)  DEVD(PHILLY)')
   DDMFILE(CHICAGO)
```

These commands create a save file named TRANSFER, which saves the ORDERENT program and then sends it as a network file to the server system in Chicago. There, the commands embedded in the SBMRMTCMD command are used to create a save file (named RECEIVE) on the server system, receive the

TRANSFER file, and restore it as ORDERENT into the newly created PGMLIB library. Lastly, a DDM file is created on the Chicago system, which allows the Chicago system to access the CUSTMAST file on the Philadelphia system. The Create DDM File (CRTDDMF) command is in System/38 syntax.

After either of these two methods is used to send the ORDERENT program to, and to create the DDM file on, the Chicago system, the ORDERENT program on that system can be used to access the CUSTMAST file on the Philadelphia system.

DDM example 2: Copying a file

After performing the first task in example 2 (Transferring a program to a server system), you decide to copy the current level of the CUSTMAST file (in Philadelphia) to the system in Chicago so you can bring the CUSTMST2 file up to date.

This example assumes that the CUSTMST2 file already exists in Chicago.

The following commands can be used to copy the CUSTMAST file from the Philadelphia system to the CUSTMST2 file on the Chicago system. (These commands are issued on the system in Philadelphia.)

```
CRTDDMF  FILE(PHILLY/COPYMAST)  RMTFILE(*NONSTD 'CUSTMST2.CHICAGO')
   RMTLOCNAME(CHIC)
CPYF  FROMFILE(PGMLIB/CUSTMAST)  TOFILE(PHILLY/COPYMAST)
   MBROPT(*REPLACE)
```

Note: One might assume that, as an alternative method, you can create a DDM file on the client system, use the SBMRMTCMD command to submit a Create DDM File (CRTDDMF) command to the server system, and then attempt to use the newly created target DDM file with another SBMRMTCMD command to perform the copy function back to the original system. However, that method cannot work, because the IBM i operating system cannot be both a client and server system within the same job.

DDM example 3: Accessing multiple IBM i files

Using the same communications environment as in examples 1 and 2, this example is used to ask inventory questions of identically named files on the two remote IBM i platforms and the remote System/38.

To do so, a program must be written (shown here in pseudocode) on the central system that can access the files named LIB/MASTER on the systems in Chicago, in Toronto, and in New York. (In this example, the MASTER files are keyed files, and the first member of each of these files is the one used. Also, data description specifications (DDS) for the MASTER files exist on the central system in Philadelphia.)

The program asks the local order entry operator for an item number (ITEMNO), and returns the quantity-on-hand (QOH) information from the files in Chicago, Toronto, and New York.

The following commands are issued on the system in Philadelphia:

```
CRTDDMF  PGMLIB/CHIFILE  RMTFILE(*NONSTD 'MASTER.LIB')
         RMTLOCNAME(CHIC)
CRTDDMF  PGMLIB/TORFILE  RMTFILE(LIB/MASTER)   RMTLOCNAME(TOR)
CRTDDMF  PGMLIB/NYCFILE  RMTFILE(LIB/MASTER)   RMTLOCNAME(NYC)
```

Here is a sample of the pseudocode to accomplish the task:

```
DECLARE CHIFILE, TORFILE, NYCFILE INPUT
Open CHIFILE, TORFILE and NYCFILE
LOOP:  Show a display asking for ITEMNO
   Read ITEMNO from the display
      Read record from CHIFILE with the key ITEMNO
      Read record from TORFILE with the key ITEMNO
      Read record from NYCFILE with the key ITEMNO
      Write all QOH values to the display
   If not function key, go to LOOP
Close CHIFILE, TORFILE and NYCFILE
END
```

Before the program is compiled, Override with Database File (OVRDBF) commands can be used to override the three files used in the program with local files that contain the external description formats, identical to the remote files being accessed. Doing so significantly reduces the time required for the compile, because the files on the remote system do not have to be accessed then.

After the program has been compiled correctly, the overrides should be deleted so that the program is able to access the remote files.

An alternative to the use of overrides is to keep the file definitions in a different library. The program can be compiled using the file definitions in that library and then run using the real library.

DDM example 4: Accessing a file on System/36

This topic shows how the pseudocode program for example 3 can be changed so that a MASTER file on the System/36 in Dallas can be accessed in the same way as the MASTER files on the IBM i operating systems and System/38 in example 3.

Assume that either you have pass-through to the System/36, or that an operator at the System/36 can make changes, if necessary, on the System/36 for you.

The following command is issued on the server in Philadelphia:

```
CRTDDMF  FILE(PGMLIB/DALFILE)  RMTFILE(MASTER)
   RMTLOCNAME(DAL)  ACCMTH(*KEYED)
```

Because the remote file referred to by the DDM file named DALFILE is on a System/36, either of two things must be done:

- The record format of the remote file must be described in the program; that is, it must be a program-described file.
- The program must be compiled with the program referring to a local IBM i file instead of the System/36 file. This local file must have the same record format name as the DDM file name. The local file need not contain any data records.

Here is a sample of the pseudocode to accomplish the task:

```
DECLARE CHIFILE, TORFILE, NYCFILE, DALFILE INPUT
Open CHIFILE, TORFILE, NYCFILE and DALFILE
LOOP:  Show a display asking for ITEMNO
Read ITEMNO from the display
   Read record from CHIFILE with the key ITEMNO
   Read record from TORFILE with the key ITEMNO
   Read record from NYCFILE with the key ITEMNO
   Read record from DALFILE with the key ITEMNO
   Write all QOH values to the display
   If not function key, go to LOOP
Close CHIFILE, TORFILE, NYCFILE and DALFILE
END
```

User FAQs

You need to consider these conditions when working with another specific IBM product.

This topic collection has concentrated on describing IBM i support for distributed relational databases in a network of IBM i products (a *like* environment). However, many distributed relational database implementations exist in a network of different DRDA-supporting platforms.

This topic provides a list of tips and techniques you might need to consider when using the IBM i product in an *unlike* DRDA environment. It is not intended to be a comprehensive list. Many problems or conditions like the ones described here depend significantly on your application. You can get more information about the differences between the various IBM platforms from the *IBM SQL Reference Volume 2*, SC26-8416, or the *DRDA Application Programming Guide*, SC26-4773.

Related reference

IBM i support

You can change the CCSID for an IBM i job by using the Change Job (CHGJOB) command.

Connecting to a distributed relational database

When you connect from an application requester (AR) other than IBM i to a DB2 for IBM i application server (AS), columns tagged with CCSID 65535 are not converted. If the files that contain these columns do not contain any columns that have a CCSID explicitly identified, the CCSID of all character columns can be changed to another CCSID value. To change the CCSID, use the **Change Physical File (CHGPF)** command. If you have logical files built over the physical file, follow the directions given in the recovery topic of the error message (CPD322D) that you get.

IBM i system value QCCSID

The default system value for QCCSID (coded character set identifier) is 65535.

Data tagged with this CCSID is not to be converted by the receiving system. You might not be able to connect to an unlike system when your IBM i application requester (AR) is using this CCSID. Also, you might not be able to use source files that are tagged with this CCSID to create applications on unlike systems.

The CCSID used at connection time is determined by the job CCSID. When a job begins, its CCSID is determined by the user profile the job is running under. The user profile can, and as a default does, use the system value QCCSID.

If you are connecting to a system that does not support the system default CCSID, you need to change your job CCSID. You can change the job CCSID by using the **Change Job (CHGJOB)** command. However, this solution is only for the job you are currently working with. The next time you will have to change the job CCSID again.

A more permanent solution is to change the CCSID designated by the user profiles used in the distributed relational database. When you change the user profiles, you affect only those users that need to have their data converted. If you are working with a Db2 for i application server (AS), you need to change the user profile that the AS uses.

If an unlike application requester connects to a Db2 for i application server using job CCSID 65535, the job will be switched to use the job default CCSID. The job default CCSID is determined by the job's language identifier (LANGID). For better performance, the job's CCSID should be switched to a value other than 65535 in this case. For example, the CCSID value can be changed to the value of the user profile under which the server jobs are run.

The default CCSID value in a user profile is *SYSVAL. This references the QCCSID system value. You can change the QCCSID system value that is used by all user profiles with the **Change System Value (CHGSYSVAL)** command. If you do this, you would want to select a CCSID that represents most (if not all) of the users on your system.

If you suspect that you are working with a system that does not support a CCSID used by your job or your system, look for the following indicators in a job log, SQLCA, or SQL diagnostic area:

Message
SQ30073

SQLCODE or DB2_RETURNED_SQLCODE
-30073

SQLSTATE
58017

Text
Distributed Data Management (DDM) parameter X'0035' not supported.

Message
SQL0332

SQLCODE or DB2_RETURNED_SQLCODE
-332

SQLSTATE
57017

Text
Total conversion between CCSID &1 and CCSID &2 not valid.

Related concepts

IBM i globalization

Related reference

Change Job (CHGJOB) command

Change System Value (CHGSYSVAL) command

CCSID conversion considerations for DB2 for z/OS and DB2 Server for VM database managers

One of the differences between a Db2 for i and other DB2 databases is that the IBM i operating system supports a larger set of CCSIDs. This can lead to errors when the other database managers attempt to perform character conversion on the data (SQLCODE –332 and SQLSTATE 57017).

Certain fields in the DB2 SQL catalog tables might be defined to have a DBCS-open data type. This is a data type that allows both double-byte character set (DBCS) and single-byte character set (SBCS) characters. The CCSID for these field types is based on the default CCSID.

When these fields are selected from a DB2 for z/OS or DB2 Server for VM application requester (AR), the SELECT statement might fail because the DB2 for z/OS and DB2 Server for VM databases might not support the conversion to this CCSID.

To avoid this error, you must change the DB2 for z/OS database or the DB2 Server for VM AR to run with either one of the following items:

- The same mixed-byte CCSID as the DBCS-OPEN fields in the IBM i SQL catalog tables.
- A CCSID that the server allows conversion of data to when the data is from the mixed-byte CCSID of the DBCS-OPEN fields in the IBM i SQL catalog tables. This CCSID might be a single-byte CCSID if the data in the IBM i SQL catalog tables DBCS-OPEN fields is all single-byte data.

You need to analyze the CCSID conversions supported on the DB2 for z/OS or DB2 Server for VM to make the correct changes to your system. See the *DB2 UDB for z/OS Administration Guide* for specific information about how to handle this error.

Why am I getting an SQL5048N message when I attempt to connect from DB2 for Linux, UNIX, and Windows?

The definition of message SQL5048N states that the release level of the database client is not supported by the release level of the database server. However, the message can sometimes be misleading.

There are several common causes for this problem:

1. You will see this error message if you have only the Client Application Enabler installed. In this case, the client system must be connected to the IBM i platform through a gateway server. Direct connection is not supported.
2. You can also get this error if someone has made errors while manually configuring the connection.

Use the Client Configuration Assistant (CCA) to avoid getting SQL5048N.

Another potential cause for the problem concerns the collection NULLID. DB2 for Linux, UNIX, and Windows, IBM DB2 Universal Driver for SQLJ and JDBC, and other application requesters use the collection NULLID for building their needed SQL packages. The collection and packages are created upon first connection. If the user profile does not have sufficient authority to create the collection, another profile with higher authority should initially connect to get these objects created.

If you believe there is another cause for the error, see the Authorized Problem Analysis Report Web site. Enter **APAR II12722** in the **Search** field.

Do IBM i files have to be journaled?

Journaling is not required if the client application is using an isolation level of no-commit (NC) or uncommitted read (UR), and if the Db2 for i SQL function determines that the query data can be blocked. In that case, commitment control is not enabled, which makes journaling unnecessary.

The answer to this question is closely related to the question in "When will query data be blocked for better performance?" on page 314.

Examples of methods of changing isolation levels are:

- The DB2 for Linux, UNIX, and Windows precompiler uses the ISOLATION UR parameter to specify uncommitted read.
- The DB2 for Linux, UNIX, and Windows command line processor (CLP) uses the command DBM CHANGE SQLISL TO UR to specify uncommitted read.
- The DB2 for Linux, UNIX, and Windows command line processor (CLP) uses the command DBM CHANGE SQLISL TO NC to specify no-commit.
- JDBC clients set their connection property isolation level to TRANSACTION_READ_UNCOMMITTED to specify uncommitted read.

When will query data be blocked for better performance?

The query data will be blocked if none of these conditions are true.

- The cursor is updatable (see note 1).
- The cursor is potentially updatable (see note 2).
- The BLOCKING NO precompiler or bind option was used on SQLPREP or SQLBIND.

Unless you force single-row protocol with the BLOCKING NO precompile/bind option, blocking will occur in both of the following cases:

- The cursor is read-only (see note 3).
- All of the following are true:
  - There is no FOR UPDATE OF clause in the SELECT, and
  - There are no UPDATE or DELETE WHERE CURRENT OF statements against the cursor in the program, and
  - Either the program does not contain dynamic SQL statements or BLOCKING ALL was used.

Notes:

1. A cursor is updatable if it is not read-only (see note 3), and one of the following items is true:
   - The select statement contained the FOR UPDATE OF clause, or
   - There exists in the program an UPDATE or DELETE WHERE CURRENT OF against the cursor.
2. A cursor is potentially updatable if it is not read-only (see note 3), and if the program includes any dynamic statement, and the BLOCKING UNAMBIG precompile or bind option was used on SQLPREP or SQLBIND.
3. A cursor is read-only if one or more of the following conditions are true:
   - The DECLARE CURSOR statement specified an ORDER BY clause but did not specify a FOR UPDATE OF clause.
   - The DECLARE CURSOR statement specified a FOR FETCH ONLY clause.
   - One or more of the following conditions are true for the cursor or a view or logical file referenced in the outer subselect to which the cursor refers:
     - The outer subselect contains a DISTINCT keyword, GROUP BY clause, HAVING clause, or a column function in the outer subselect.
     - The select contains a join function.
     - The select contains a UNION operator.

- The select contains a subquery that refers to the same table as the table of the outer-most subselect.
- The select contains a complex logical file that had to be copied to a temporary file.
- All of the selected columns are expressions, scalar functions, or constants.
- All of the columns of a referenced logical file are input only.

How do you interpret an SQLCODE and the associated tokens reported in an SQL0969N error message?

The client support used with DB2 for Linux, UNIX, and Windows returns message SQL0969N when reporting host SQLCODEs and tokens for which it has no equivalent code.

Here is an example of message SQL0969N:

```
SQL0969N  There is no message text corresponding to SQL error
"-7008" in the Database Manager message file on this workstation.
The error was returned from module "QSQOPEN" with original
tokens "TABLE1    PRODLIB1    3".
```

Use the **Display Message Description (DSPMSGD)** command to interpret the code and tokens:

```
DSPMSGD SQL7008 MSGF(QSQLMSG)
```

Select option 1 (Display message text) and the system presents the Display Formatted Message Text display. The three tokens in the message are represented by &1, &2, and &3 in the display. The reason code in the example message is 3, which points to Code 3 in the list at the bottom of the display.

```
                         Display Formatted Message Text
System:   RCHASLAI
Message ID . . . . . . . . . :   SQL7008
Message file . . . . . . . . :   QSQLMSG
Library  . . . . . . . . . . :   QSYS

Message . . . . :    &1 in &2 not valid for operation.
Cause . . . . . :    The reason code is &3.  A list of reason codes follows:
-- Code 1 indicates that the table has no members.
-- Code 2 indicates that the table has been saved with storage free.
-- Code 3 indicates that the table is not journaled, the table is
journaled to a different journal than other tables being processed under
commitment control, or that you do not have authority to the journal.
-- Code 4 indicates that the table is in a production library but the user
is in debug mode with UPDPROD(*NO); therefore, production tables may not be
updated.
-- Code 5 indicates that a table, view, or index is being created into a
production library but the user is in debug mode with UPDPROD(*NO);
therefore, tables, views, or indexes may not be created.
More...
Press Enter to Continue.

F3=Exit   F11=Display unformatted message text    F12=Cancel
```

Related reference

Display Message Description (DSPMSGD) command

How can the host variable type in WHERE clauses affect performance?

One potential source of performance degradation on the IBM i operating system is the client's use in a C program of a floating-point variable for a comparison in the WHERE clause of a SELECT statement.

If the operating system has to do a conversion of the data for that column, that will prevent it from being able to use an index on that column. You should always try to use the same type for columns, literals, and host variables used in a comparison. If the column in the database is defined as packed or zoned decimal, and the host variable is of some other type, that can present a problem in C.

Related concepts

Programming techniques for database performance

Can I use a library list for resolving unqualified table and view names?

The IBM i operating system supports a limited capability to use the operating system naming option when accessing Db2 for i data from a DRDA client program other than IBM i, such as those that use the DB2 for Linux, UNIX, and Windows product.

Previously, only the SQL naming option has been available when connecting from unlike DRDA clients. System naming changes several characteristics of Db2 for i. For example:

1. The library list is searched for resolving unqualified table and view names.
2. When running a CREATE SQL statement, an unqualified object will be created in the current library.
3. A slash (/) instead of a period (.) is used to separate qualified objects' names from the library or collection in which they reside.
4. Certain authorization characteristics are changed.

For details, read about system naming in the SQL reference. For more information about the implications regarding authorization, see Planning and design.

DB2 for Linux, UNIX, and Windows supports the specification of generic bind options on two of its program preparation commands, the precompile (PREP) command and the (BIND) command. IBM i naming can be specified on either of them as in the following examples drawn from a Windows batch file:

For DB2 for Linux, UNIX, and Windows, Version 8, and later:

```
DB2 PREP %1.SQC BINDFILE OS400NAMING SYSTEM ...
DB2 BIND %1.BND OS400NAMING SYSTEM ...
```

For DB2 for Linux, UNIX, and Windows, version earlier than Version 8:

```
DB2 PREP %1.SQC BINDFILE GENERIC 'OS400NAMING SYSTEM' ...
DB2 BIND %1.BND GENERIC 'OS400NAMING SYSTEM' ...
```

Note that on the Windows development platform, single quotation marks are used around the generic option name/value pair. On an AIX or UNIX platform, quotation marks should be used.

Note: For OS/400 V4R5 and V5R1, the name of the option is AS400NAMING, not OS400NAMING.

The only valid value for the OS400NAMING option besides SYSTEM is SQL, which is the default value and is the only possible option from a non-IBM i client prior to the introduction of this feature.

If you use the OS400NAMING option on the (BIND) command but not on the (PREP) command, then you might need to code a parameter on the (PREP) command that indicates that a bind file should be created in spite of SQL errors detected by the system. In the case of DB2 for Linux, UNIX, and Windows, use the SQLERROR CONTINUE parameter for this purpose. The capability is described as limited because in certain situations, the client-side software might parse an SQL statement intended for execution on the remote system. If a slash instead of a period is used to separate a schema ID from a table ID, as is required for system naming, the statement might be rejected as having incorrect syntax.

Related concepts

Planning and design
To prepare for a distributed relational database, you must understand both the needs of the business and relational database technology. To prepare for the use of DDM, you need to meet several requirements including communication requirements, security requirements, and modification requirements.

Related reference

SQL reference

How can unlike clients override package options such as NLSS sort sequences, system naming and separate date/time formats?

The IBM i operating system supports DRDA generic bind options.

Many environmental options are specific to a server and thus not available on the precompiling and binding commands provided from other products. DRDA provides for this shortcoming by allowing generic options to be passed on the bind request. The following is a list of the generic bind options supported by DB2.

Table 31. Supported Generic Bind Options

Bind Option Name	**Bind Option Values**
PATH	*schema-names*
OS400NAMING	SQL (Default), SYSTEM
SORTSEQ	HEX (Default), JOBRUN
MINDIVSCALE	*number-char*
MAXSCALE	*number-char*
ROUNDING	HALFEVEN (Default), CEILING, DOWN, FLOOR, HALFDOWN, HALFUP, UP
EXTENDEDINDICATOR	NO (Default), YES
DATEOVERIDE	ISO (Default), USA, EUR, JIS
CONCURRENTACCESSRESOLUTION	NO (Default), YES

DB2 for Linux, UNIX and Windows supports the specification of generic bind options on two of its program preparation commands, the precompile (PREP) command and the (BIND) command. The following examples overrides the time format to Japanese Industrial Standard and date format to European. This is done since DATETIME is assumed to be the same format, which is not required on DB2. It is drawn from a Windows batch file:

```
DB2 PREP %1.SQC BINDFILE DATETIME JIS GENERIC "DATEOVERRIDE EUR"...
DB2 BIND %1.BND DATETIME JIS GENERIC "DATEOVERRIDE EUR"...
```

Note: On the Windows development platform, single quotation marks are used around the generic option name/value pair, but on an AIX or UNIX platform, quotation marks should be used.

Why are no rows returned when I perform a query?

One potential cause of this problem is a failure to add an entry for the IBM i platform in the DB2 for Linux, UNIX, and Windows Database Communication Services directory.

What level of DB2 for Linux, UNIX, and Windows is required to interact with DB2 for IBM i?

These FixPaks are required for interaction.

- DB2 for Linux, UNIX and Windows Version 7 FixPak 10
- DB2 for Linux, UNIX and Windows Version 8 FixPak 4

You can get these FixPaks from the DB2 9 for Linux, UNIX, and Windows Web site.

How can I get scrollable cursor support enabled from DB2 for Linux, UNIX, and Windows to the IBM i platform?

On the DB2 for Linux, UNIX, and Windows V8 client, you must be using FixPak 4, or later. You can also use a client that runs a later version of DB2 for LUW that provides scrollable cursor support to the IBM i platform.

If you are using FixPak 4 on a client that runs DB2 for Linux, UNIX, and Windows V8, you must perform ***one*** of the following actions.

Note: If you are using a later version of the DB2 for LUW client that provides the scrollable cursor support to the IBM i platform, you do not need to perform these actions.

- Issue this command:

```
UPDATE CLI CFG FOR SECTION dbname USING CURSORTYPES 1
```

  Where *dbname* is the name of your IBM i database.
- Edit the db2cli.ini file using this syntax:

```
CURSORTYPES = 1
```

Other tips for interoperating in unlike environments

This topic collection provides additional information for using DB2 for IBM i with DB2 for Linux, UNIX, and Windows.

DB2 Connect as opposed to DB2 for Linux, UNIX, and Windows

Users are sometimes confused over what products are needed to perform the DRDA application server function as opposed to the application requester (client) function. The AR is sometimes referred to as DB2 Connect; and both the AR and AS as DB2 for Linux, UNIX, and Windows. DB2 refers to the following products:

- DB2 for AIX
- DB2 for HP-UX
- DB2 for Linux
- DB2 for Sun Solaris
- DB2 for Windows

Proper configuration and maintenance level

Be sure to follow the installation and configuration instructions given in the product manuals carefully. Make sure that you have the most current level of the products. Apply the appropriate fix packs if not.

Table and collection naming

SQL tables accessed by DRDA applications have three-part names: the first part is the *database name*, the second part is a *collection ID*, and the third part is the *base table name*. The first two parts are optional. Db2 for i qualifies table names at the second level by a collection (or library) name. Tables reside in the Db2 for i database.

In DB2, tables are qualified by a user ID (that of the creator of the table), and reside in one of possibly multiple databases on the platform. DB2 for Linux, UNIX, and Windows has the same notion of using the user ID for the collection ID.

In a dynamic query from DB2 for Linux, UNIX, and Windows to Db2 for i, if the name of the queried table is specified without a collection name, the query uses the user ID of the target side job (on the IBM i operating system) for the default collection name. This might not be what is expected by the user and can cause the table to not be found.

A dynamic query from Db2 for i to DB2 would have an implied table qualifier if it is not specified in the query in the form *qualifier.table-name*. The second-level DB2 table qualifier defaults to the user ID of the user making the query.

You might want to create the DB2 databases and tables with a common user ID. Remember, for DB2 there are no physical collections as there are in Db2 for i. There is only a table qualifier, which is the user ID of the creator.

APPC communications setup

IBM i communications must be configured properly, with a controller and device created for the workstation when you use APPC with either DB2 for Linux, UNIX, and Windows as an AR, or with DB2 as an AS.

Setting up the RDB directory

When adding an entry in the RDB directory for each DB2 database that a IBM i product will connect to, use the Add Relational Database Directory Entry (ADDRDBDIRE) command. The RDB name is the DB2 database name.

When using APPC communications, the remote location name is the name of the workstation.

When using TCP/IP, the remote location name is the domain name of the workstation, or its IP address. The port used by the DRDA server is typically not 446, the well-known DRDA port that the IBM i operating system uses.

Consult the DB2 product documentation to determine the port number. A common value used is 50000. An example DSPRDBDIRE display screen showing a properly configured RDB entry for a DB2 server follows.

```
Display Relational Database Detail
      Relational database  . . . . . . :   SAMPLE
      Remote location:
        Remote location  . . . . . . . :   9.5.36.17
          Type . . . . . . . . . . . . :   *IP
        Port number or service name  . :   50000
      Text . . . . . . . . . . . . . . :   My DB2 server
```

How do I create the NULLID packages used by DB2 for Linux, UNIX, and Windows and IBM DB2 Universal Driver for SQLJ and JDBC?

Before using DB2 for Linux, UNIX, and Windows to access data on Db2 for i, you must create IBM i SQL packages for application programs and for the DB2 for Linux, UNIX, and Windows utilities.

The DB2 (PREP) command can be used to process an application program source file with embedded SQL. This processing will create a modified source file containing host language calls for the SQL statements and it will, by default, create an SQL package in the database you are currently connected to.

To bind DB2 for Linux, UNIX, and Windows to a Db2 for i server, follow these steps:

1. CONNECT TO rdbname
2. Bind path@ddcs400.lst BLOCKING ALL SQLERROR CONTINUE MESSAGES DDCS400.MGS GRANT PUBLIC

   Replace 'path' in the path@ddcs400.lst parameter above with the default path C:\SQLLIB\BND\ (c:/sqllib/bin/ on non-INTEL platforms), or with your value if you did not install to the default directory.
3. CONNECT RESET

How do I set up the interactive SQL packages?

To use interactive SQL, you need the IBM DB2 Query Manager and SQL Development Kit for i product installed on IBM i. To access data on DB2, do the following:

1. When starting a session with STRSQL, use session attributes of NAMING(*SQL), DATFMT(*ISO), and TIMFMT(*ISO). Other formats besides *ISO work, but not all, and what is used for the date format (DATFMT) must also be used for the time format (TIMFMT).
2. Note the correspondence between schemas on the IBM i operating system and table qualifier (the creator's user ID) for DB2.
3. For the first interactive session, you must do this sequence of SQL statements to get a package created on DB2: (1) RELEASE ALL, (2) COMMIT, and (3) CONNECT TO rdbname (where 'rdbname' is replaced with a particular database).

As part of your setup for the use of interactive SQL, you might also want to use the statement GRANT EXECUTE ON PACKAGE QSQL400.QSQLabcd TO PUBLIC (or to specific users), so that others can use the SQL PKG created on the PC for interactive SQL. The actual value for abcd in the following GRANT statement can be determined from the following table, which gives the package names for various sets of options that are in effect when the package is created. For example, you would use the statement GRANT EXECUTE ON PACKAGE QSQL400.QSQL0200 TO *some-user* if the following options were in use when you created the package: *ISO for date, *ISO for time, *CS for commitment control, a single quotation mark for string delimiter, and single byte for character subtype.

Position	Option	Value
a	Date Format	0 = ISO, JIS date format 1 = USA date format 2 = EUR date format
b	Time Format	0 = JIS time format 1 = USA time format 2 = EUR, ISO time format
c	Commitment Control Decimal Delimiter	0 = *CS commitment control period decimal delimiter 1 = *CS commitment control comma decimal delimiter 2 = *RR commitment control period decimal delimiter 3 = *RR commitment control comma decimal delimiter
d	String Delimiter Default Character Subtype	0 = single quotation mark string delimiter, single byte character subtype 1 = single quotation mark string delimiter, double byte character subtype 2 = quotation marks string delimiter, single byte character subtype 3 = quotation marks string delimiter, double byte character subtype

Close of queries

DB2 for Linux, UNIX, and Windows provides an option to request that read locks be released when queries are closed either implicitly or explicitly. It is not considered an error if the system does not honor the request, which is the case for IBM i products. DB2 for Linux, UNIX, and Windows provides another option to specify whether the system should close the query implicitly for a nonscrollable cursor when there are no more rows to read. Previously, the system made this decision. The IBM i AS supports this feature.

What is the maximum length of user IDs and passwords in a heterogeneous environment?

Db2 for i running as the application requester (AR) allows user IDs and passwords longer than ten characters when running to an unlike application server (AS). The exact limits are specified in the description of the specific interface being used. For example, see the DB2 for IBM i SQL reference topic for limits on the SQL CONNECT statement.

Creating interactive SQL packages on DB2 Server for VM

On DB2 Server for VM, a collection name is synonymous with a user ID. To create packages to be used with interactive SQL or Db2 for i Query Manager on an DB2 Server for VM application server, create a user ID of QSQL400 on the IBM i operating system. This user ID can be used to create all the necessary packages on the DB2 Server for VM application server. Users can then use their own user IDs to access Server for VM through interactive SQL or DB2 Query Manager on IBM i.

Troubleshooting DRDA and DDM

When a problem occurs accessing a distributed relational database, determine the nature of the problem and whether it is a problem with the application or a problem with the local or remote system.

To resolve a problem or obtain customer support assistance to resolve the problem, you need the following skills:

- An understanding of the IBM i licensed program
- An idea about whether the problem is on a client or a server
- Familiarity using IBM i problem management functions

For more information about diagnosing problems in a distributed relational database, see the *Distributed Relational Database Problem Determination Guide*, SC26-4782.

Related concepts

DRDA and DDM performance
No matter what kind of application programs you are running on a system, performance can always be a concern. For a distributed relational database, network, system, and application performance are all crucial.

IBM i problem handling overview

The IBM i licensed program helps you manage problems for both user-detected and system-detected problems that occur on local and remote IBM i platforms.

The IBM i operating system tracks both user- and system-detected problems using the problem log and the problem manager. A problem state is maintained from when a problem is detected (OPENED) to when it is resolved (CLOSED) to help you with tracking.

Problem handling support includes:

- Messages with initial problem handling information
- Automatic alerting of system-detected problems
- Integrated problem logging and tracking
- First-failure data capture (FFDC)
- Electronic customer-support service requisition
- Electronic customer support, program temporary fix (PTF) requisition

System-detected problems

The IBM i operating system and its attached devices are able to detect some types of problems. These are called *system-detected problems*. When a problem is detected, several operations take place:

- An entry in the product activity log is created
- A problem record is created
- A message is sent to the QSYSOPR message queue

Information is recorded in the error log and in the problem record. When serious problems are detected, a spooled file of FFDC information is also created. The error log and the problem record might contain the following information:

- Vital product data
- Configuration information
- Reference code
- The name of the associated device
- Additional failure information

User-detected problems

User-detected problems are usually related to program errors that can cause any of the following problems to occur:

- Job problems
- Incorrect output
- Messages indicating a program failure
- Device failures that are not detected by the system
- Poor performance

When a user detects a problem, no information is gathered by the server until problem analysis is run or you select the option to save information to help resolve a problem from the Operational Assistant USERHELP menu.

Related concepts

System and communications problems
When a problem with a system or its communications occurs, a message is generated. System-detected problems are automatically entered into the problem log, where they can be viewed and analyzed.

System and communications problems

When a problem with a system or its communications occurs, a message is generated. System-detected problems are automatically entered into the problem log, where they can be viewed and analyzed.

You can run problem analysis on logged problems at any time by entering the **`Analyze Problem (ANZPRB)`** command from any system command line. This command takes you through an analysis procedure and stores additional problem-related information in the problem log.

Use the **`Work with Problems (WRKPRB)`** command to view the problem log.

Using the problem log, you can display all the problems that have been recorded on the local system. You can also display detailed information about a specific problem, such as the following items:

- Product type and serial number of device with a problem
- Date and time of the problem
- Part that failed and where it is located
- Problem status

From the problem log you can also analyze a problem, report a problem, or determine any service activity that has been done.

Related concepts

IBM i problem handling overview
The IBM i licensed program helps you manage problems for both user-detected and system-detected problems that occur on local and remote IBM i platforms.

Related reference

Analyze Problem (ANZPRB) command

Work with Problem (WRKPRB) command

DRDA application problems

The best time to handle a problem with an application is before it goes into production. However, it is impossible to anticipate all the conditions that will exist for an application when it gets into general use.

A problem you encounter when running a distributed relational database application can exhibit two general symptoms: incorrect output or the application does not complete in the expected time. The figures in these topics show generally how you can classify problems as application program problems, performance-related problems, and system-related problems so that you can use standard IBM i problem analysis methods to resolve the problem.

The job log of either the application requester (AR) or the application server (AS) can tell you that a package failed; the listings of the program or the package can tell you why it failed. The SQL compilers provide diagnostic tests that show the SQLCODEs and SQLSTATEs generated by the precompile process on the diagnostic listing.

For Integrated Language Environment (ILE) precompiles, you can optionally specify OPTION(*XREF) and OUTPUT(*PRINT) to print a precompile source and cross-reference listing. For non-ILE precompiles, you can optionally specify *SOURCE and *XREF on the OPTIONS parameter of the Create SQL program (CRTSQL*xxx*) commands to print a precompile source and cross-reference listings.

Resolving incorrect output problems
Use the error messages, SQLCODEs, or SQLSTATEs to determine the cause of a problem where there is incorrect output.

See the following figure for the process of isolating the incorrect output problem. When you receive an error message for an incorrect output problem, the message description indicates what the problem is and provides corrective actions. If you do not receive an error message, you must determine whether distributed relational database is causing the failure. To determine the cause, run the failing statement on the local application server (AS) or use Interactive Structured Query Language (ISQL) to run the statement on the AS. If you can create the problem locally, the problem is not with distributed relational database support. Use IBM i problem analysis methods to provide specific information for your support staff, depending on the results of this operation.

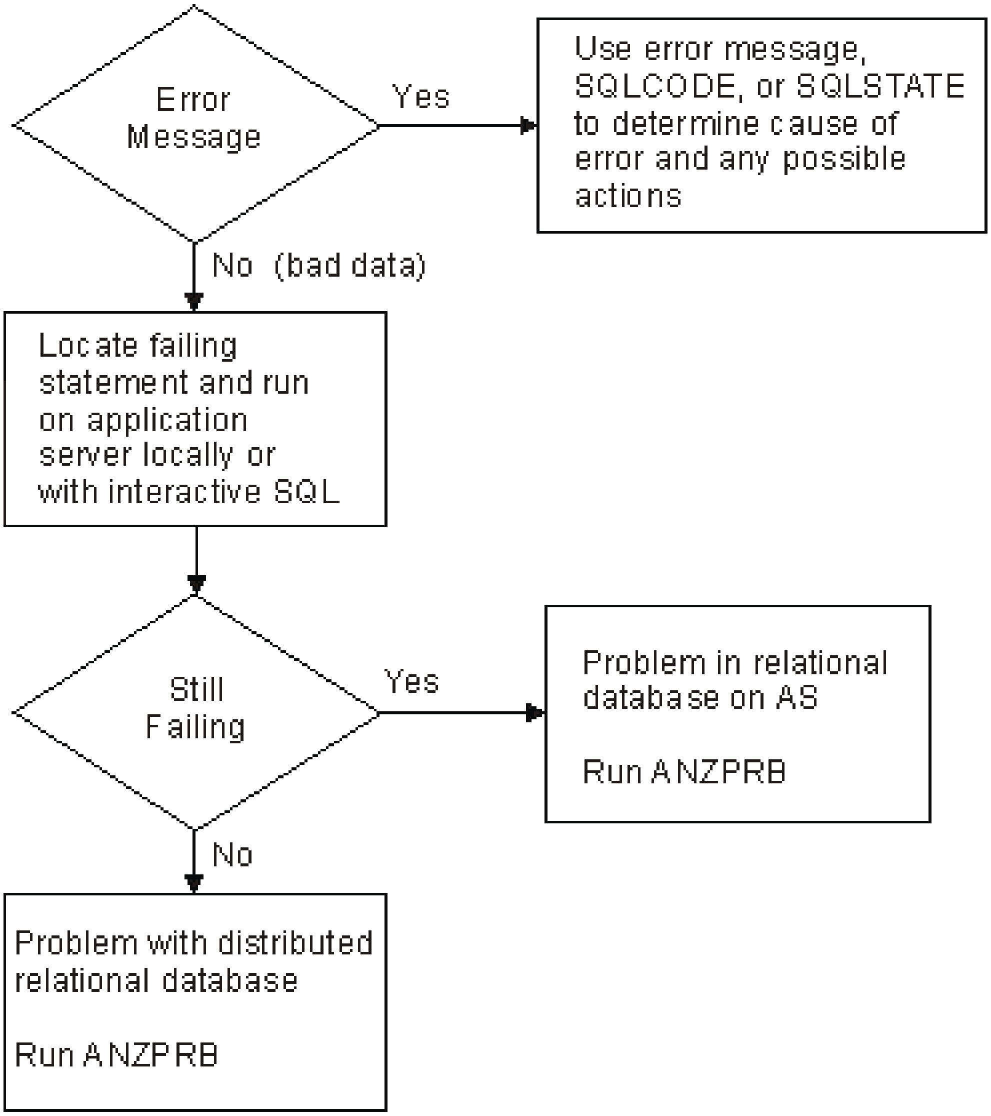

Error Message
Yes
Use error message, SQLCODE, or SQLSTATE to determine cause of error and any possible actions
No (bad data)
Locate failing statement and run on application server locally or with interactive SQL
Still Failing
Yes
Problem in relational database on AS
Run ANZPRB
No
Problem with distributed relational database
Run ANZPRB

Figure 44. Resolving incorrect output problems

Resolving loop, wait, or performance problems

If a request takes longer than expected to complete, check the application requester (AR) first.

1. Check the job log for message SQL7969, which indicates that a connection to a relational database is complete. This tells you that the application is a distributed relational database application.
2. Determine whether the system or the application is waiting or looping.
   a. Check the AR for a loop using the **`Work with Job (WRKJOB)`** command to display the program stack.
   b. Check the program stack to determine whether the system is looping. If the application itself is looping, contact the application programmer for resolution.
3. Determine whether the application requester is waiting.
   - If you see QAPDEQUE and QCNSRCV on the stack, the AR is waiting for the application server (AS).
   - If the system is not in a communications wait state, use problem analysis procedures to determine whether there is a performance problem or a wait problem.

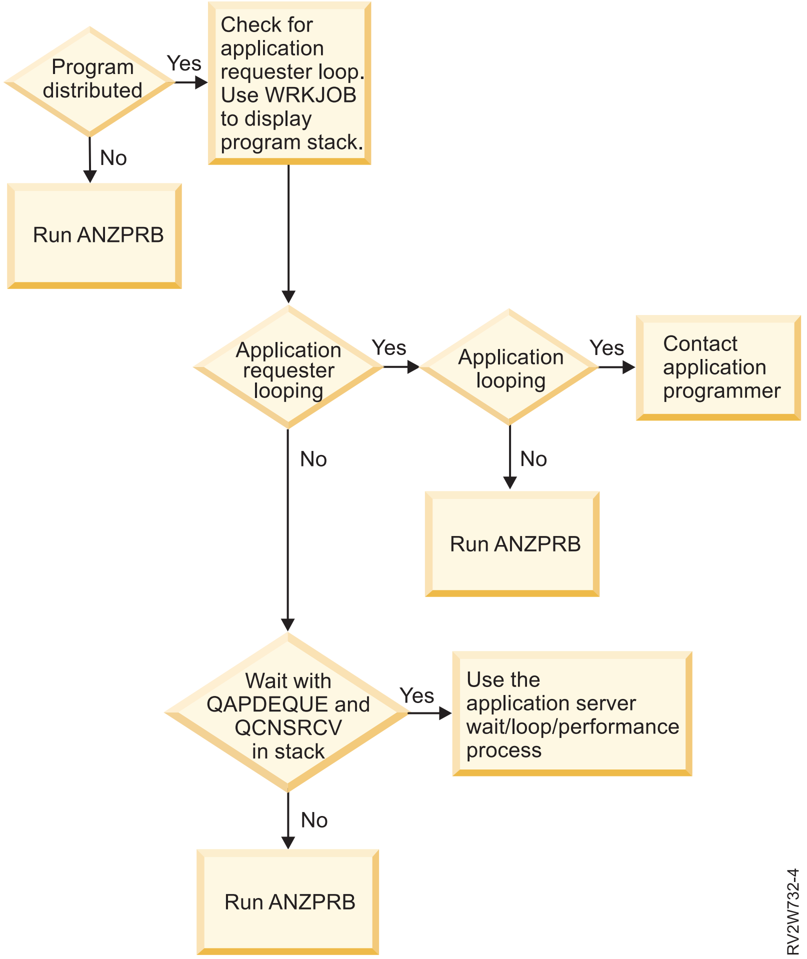

Program distributed
Yes
No
Check for application requester loop. Use WRKJOB to display program stack.
Run ANZPRB
Application requester looping
Yes
No
Application looping
Yes
No
Contact application programmer
Run ANZPRB
Wait with QAPDEQUE and QCNSRCV in stack
Yes
No
Use the application server wait/loop/performance process
Run ANZPRB
RV2W732-4

Figure 45. Resolving loop, wait, or performance problems on the application requester

You can find the AR job name by looking at the job log on the AS. When you need to check the AS job, use the **Work with Job (WRKJOB)**, **Work with Active Jobs (WRKACTJOB)**, or **Work with User Jobs (WRKUSRJOB)** command to locate the job on the AS.

From one of these job displays, look at the program stack to see if the AS is looping:

- If it is looping, use problem analysis to handle the problem.
- If it is not looping, check the program stack for WAIT with QCNTRCV, which means the AS is waiting for the AR.
- If both systems are in this communications wait state, there is a problem with the network.
- If the AS is not in a wait state, there is a performance issue.

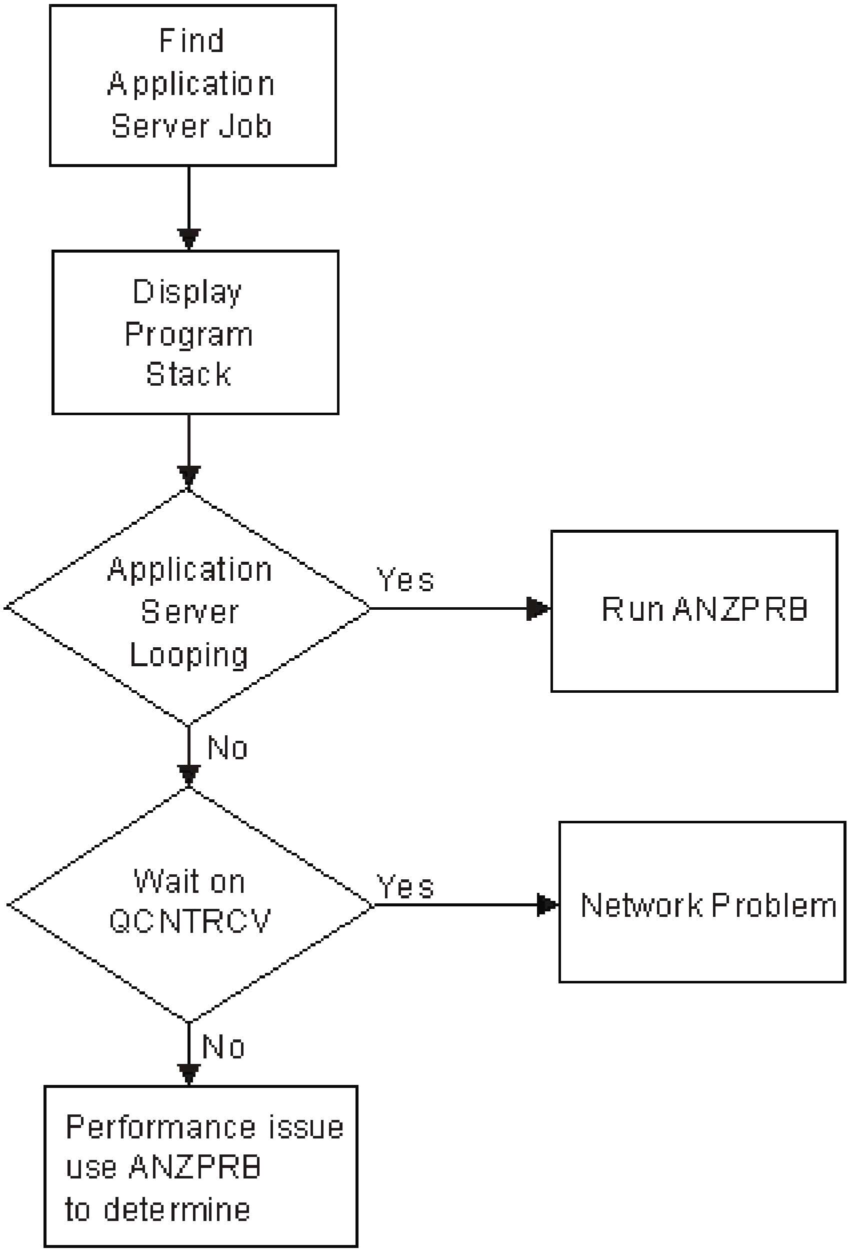

Find Application Server Job
Display Program Stack
Application Server Looping
Yes
Run ANZPRB
No
Wait on QCNTRCV
Yes
Network Problem
No
Performance issue use ANZPRB to determine

Figure 46. Resolving wait, loop, or performance problems on the application server

Two common sources of slow query performance are:

- An accessed table does not have an index. If this is the case, create an index using appropriate fields as the key.
- The rows returned on a query request are not blocked. Whether the rows are blocked can cause a significant difference in query performance. Tune the application to take advantage of blocking.

If you have not already created the SQL packages for the product in DB2 for IBM i, the first time you connect to DB2 for IBM i from a workstation using a product such as DB2 JDBC Universal Driver or DB2 for Linux, UNIX, and Windows, the packages are created automatically. The NULLID collection might need to be created automatically as well. This results in a delay in getting a response from the system for one of the first SQL statements issued after the initial connection.

A long delay can occur if the system to which you are trying to connect over TCP/IP is not available. A several minute timeout delay precedes the message `A remote host did not respond within the timeout period.` An incorrect IP address in the RDB directory can also cause a delay.

Related concepts

Factors that affect blocking for DRDA
A very important performance factor is whether blocking occurs when data is transferred between the application requester (AR) and the application server (AS). A group of rows transmitted as a block of data requires much less communications overhead than the same data sent one row at a time.

Related tasks

Locating distributed relational database jobs
When you are looking for information about a distributed relational database job on a client and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

Working with jobs in a distributed relational database
The **`Work with Job (WRKJOB)`** command presents the Work with Job menu. This menu allows you to select options to work with or to change information related to a specified job. Enter the command without any parameters to get information about the job you are currently using.

Working with user jobs in a distributed relational database
If you know the user profile (user name) being used by a job, you can use the **`Work with User Jobs (WRKUSRJOB)`** command to display or change job information. Enter the command without any parameters to get a list of the jobs on the system with your user profile.

Working with active jobs in a distributed relational database
Use the **`Work with Active Jobs (WRKACTJOB)`** command if you want to monitor the jobs running for several users, or if you are looking for a job and you do not know the job name or the user ID.

Listings

The listing from the Create SQL program (CRTSQL*xxx*) command provides these kinds of information.

- The values supplied for the parameters of the precompile command
- The program source
- The identifier cross-references
- The messages resulting from the precompile

Precompiler listing

Here is an example precompiler listing.

```
57xxST1 VxRxMx yymmdd             Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10   Page   1
Source type...............C
Object name...............TST/UPDATEPGM
Source file...............*LIBL/QCSRC
Member....................*OBJ
Options...................*XREF
Listing option............*PRINT
Target release............*CURRENT
INCLUDE file..............*LIBL/*SRCFILE
Commit....................*CHG
Allow copy of data........*YES
Close SQL cursor..........*ENDACTGRP
Allow blocking............*READ
Delay PREPARE.............*NO
Generation level..........10
Margins...................*SRCFILE
Printer file..............*LIBL/QSYSPRT
Date format...............*JOB
Date separator............*JOB
Time format...............*HMS
Time separator ...........*JOB
Replace...................*YES
Relational database.......RCHASLKM
User .....................*CURRENT
RDB connect method........*DUW
Default Collection........*NONE
Package name..............*OBJLIB/*OBJ
Created object type.......*PGM
Debugging view............*NONE
Dynamic User Profile......*USER
Sort Sequence.............*JOB
Language ID...............*JOB
IBM SQL flagging..........*NOFLAG
ANS flagging..............*NONE
Text......................*SRCMBRTXT
Source file CCSID.........37
Job CCSID.................65535
Source member changed on 02/10/06  14:25:33
57xxST1 VxRxMx yymmdd             Create SQL ILE C Object  UPDATEPGM 02/10/06 14:30:10   Page
2
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR
Last change
1   /*******************************************************************/               100
2   /*  This program is called to update the DEPTCODE of file RWDS/DPT1 */               200
3   /* to NULL.  This is run once a month to clear out the old         */               300
4   /* data.                                                           */               400
5   /*                                                                 */               500
6   /*  NOTE: Because this program was compiled with an RDB name, it is */               600
7   /* not necessary to do a connect, as an implicit connect will take  */               700
8   /* place when the program is called.                               */               800
9   /*******************************************************************/               900
10   #include <stdio.h>                                                             1000
11   #include <stdlib.h>                                                            1100
12   exec sql include sqlca;                                                            1200
13                                                                                      1300
14   main()                                                                             1400
15     {                                                                                1500
16        /*  Just update RWDS/DPT1, setting deptcode = NULL         */                   1600
17        exec sql update RWDS/DPT1                                                     1700
18               set deptcode = NULL;                                                   1800
19     }                                                                                1900
* * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 47. Listing from a precompiler

Figure 48. Listing from a precompiler (continued)

```
57xxST1 VxRxMx yymmdd             Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10
Page     3
CROSS REFERENCE
Data Names                    Define    Reference
```

```
DEPTCODE                         ****     COLUMN
18
DPT1                             ****     TABLE IN RWDS
17
RWDS                             ****     COLLECTION
17
57xxST1 VxRxMx yymmdd             Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10
Page     4
DIAGNOSTIC MESSAGES
MSG ID  SEV  RECORD  TEXT
SQL0088   0      17  Position 15 UPDATE applies to entire table.
SQL1103  10      17  Field definitions for file DPT1 in RWDS not found.
Message Summary
Total    Info    Warning    Error    Severe    Terminal
2      1        1         0        0          0
10 level severity errors found in source
19 Source records processed
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

CRTSQLPKG listing

The example listing from the Create Structured Query Language Package (CRTSQLPKG) command provides these types of information.

- The values used on the parameters of the command
- The statement in error, if any
- The messages resulting from running the Create Structured Query Language Package (CRTSQLPKG) command

The following figure illustrates this information:

```
57xxST1 VxRxMx yymmdd                 Create SQL package 02/10/06 14:30:31   Page   1
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR
Last change
Program name..............TST/UPDATEPGM
Relational database.......*PGM
User .....................*CURRENT
Replace...................*YES
Default Collection........*PGM
Generation level..........10
Printer file..............*LIBL/QSYSPRT
Object type...............*PGM
Module list...............*ALL
Text......................*PGMTXT
Source file...............TST/QCSRC
Member....................UPDATEPGM
```

```
57xxST1 VxRxMx yymmdd              Create SQL package 02/10/06 14:30:31   Page   2
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR
Last change
17   UPDATE RWDS / DPT1 SET deptcode = NULL
DIAGNOSTIC MESSAGES
MSG ID  SEV  RECORD  TEXT
SQL0204  10      17  Position 17 DPT1 in RWDS type *FILE not found.
SQL5057              SQL Package UPDATEPGM in TST created at KC000 from
module UPDATEPGM.
Message Summary
Total    Info    Warning    Error    Severe    Terminal
1      0        1         0        0          0
10 level severity errors found in source
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 49. Listing from CRTSQLPKG

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

SQLCODEs and SQLSTATEs

Program interfaces using SQL return error information to the application program when an error occurs. SQLSTATEs and their corresponding SQLCODEs are returned to the application program in either the SQL communication area (SQLCA) or the SQL diagnostic area.

An SQLCA is a collection of variables in a control block in space provided by the application that is updated by the database management system with information about the SQL statement most recently run. An SQL diagnostic area is a more complex storage area in space provided by the database manager that is designed to communicate more extensive information about the SQL statement most recently run.

When an SQL error is detected, a five-character global variable called the SQLSTATE identifies the nature of the error. In addition to the SQLSTATE, an integer SQLCODE is also available. However, the SQLCODE does not return the same return code for the same error condition among the current four IBM relational database products. SQLSTATE has been designed so that application programs can test for specific error conditions or classes of errors regardless of which DB2 product the application is connected to.

If SQL encounters a hard error while processing a statement, the SQLCODE is a negative number (for example, SQLCODE -204). If SQL encounters an exceptional but valid condition (warning) while processing a statement, the SQLCODE is a positive number (for example, SQLCODE +100). If SQL encounters no error or exceptional condition while processing a statement, the SQLCODE is 0. Every Db2 for i SQLCODE has a corresponding message in message file QSQLMSG in library QSYS. For example, SQLCODE -204 is logged as message ID SQL0204.

Because the returned error information is a valuable problem-diagnosis tool, it is a good idea to include in your application programs the instructions necessary to display some of the information contained in either the returned SQLCA or SQL diagnostic area. The message tokens discussed here are also very valuable for problem diagnosis:

- SQLSTATE

  Return code.
- SQLCODE (SQLCA) or DB2_RETURNED_SQLCODE (SQL diagnostic area)

  Return code.
- SQLERRD(3) (SQLCA) or ROW_COUNT (SQL diagnostic area)

  The number of rows updated, inserted, or deleted by SQL.

The complete message can be viewed online by using the **`Display Message Description (DSPMSGD)`** command.

Related concepts

SQL messages and codes

Related reference

SQL reference

Display Message Description (DSPMSGD) command

Distributed relational database SQLCODEs and SQLSTATEs

The list provides some of the common SQLCODEs and SQLSTATEs associated with distributed relational database processing.

In these brief descriptions of the SQLCODEs (and their associated SQLSTATEs), message data fields are identified by an ampersand (&), and a number (for example, &1). The replacement text for these fields is stored in SQLERRM if the application program is using an SQLCA, or in DB2_ORDINAL_TOKEN_n (where n is the token number) if the application program is using the SQL diagnostic area. More detailed cause and recovery information for any SQLCODE can be found by using the **`Display Message Description (DSPMSGD)`** command.

Table 32. SQLCODEs and SQLSTATEs

SQLCODE	SQLSTATE	Description
+100	02000	This SQLSTATE reports a No Data exception warning due to an SQL operation on an empty table, zero rows identified in an SQL UPDATE or SQL DELETE statement, or the cursor in an SQL FETCH statement was after the last row of the result table.
+114	0A001	Relational database name &1 not the same as current system &2.
+304	01515	This SQLSTATE reports a warning on a FETCH or SELECT into a host variable list or structure that occurred because the host variable was not large enough to hold the retrieved value. The FETCH or SELECT does not return the data for the indicated SELECT item, the indicator variable is set to -2 to indicate the return of a NULL value, and processing continues.
+331	01520	Character conversion cannot be performed.
+335	01517	Character conversion has resulted in substitution characters.
+551	01548	Not authorized to object &1 in &2 type &3.
+552	01542	Not authorized to &1.
+595	01526	Commit level &1 has been escalated to &2 lock.
+802	01519	This SQLSTATE reports an arithmetic exception warning that occurred during the processing of an SQL arithmetic function or arithmetic expression that was in the SELECT list of an SQL select statement, in the search condition of a SELECT or UPDATE or DELETE statement, or in the SET clause of an UPDATE statement. For each expression in error, the indicator variable is set to -2 to indicate the return of a NULL value. The associated data variable remains unchanged, and processing continues.
+863	01539	Only SBCS characters allowed to relational database &1.
+990	01587	This SQLSTATE reports a pending response or a mixed outcome from at least one participant during the two-phase process.
+30104	01615	Bind option ignored.
-114	42961	Relational database &1 not the same as current system &2.
-144	58003	Section number &1 not valid. Current high section number is &3 Reason &2.
-145	55005	Recursion not supported for heterogeneous application server.
-175	58028	The commit operation failed.
-189	22522	Coded character set identifier &1 is not valid.
-191	22504	A mixed data value is invalid.
-250	42718	Local relational database not defined in the directory.
-251	2E000 42602	Character in relational database name &1 is not valid.
-300	22024	A NUL-terminated input host variable or parameter did not contain a NUL.

Table 32. SQLCODEs and SQLSTATEs (continued)

SQLCODE	SQLSTATE	Description
-302	22001	Conversion error on input host variable &2.
-330	22021	Character conversion cannot be performed.
-331	22021	Character conversion cannot be performed.
-332	57017	Character conversion between CCSID &1 and CCSID &2 not valid.
-334	22524	Character conversion resulted in truncation.
-351 -352	56084	An unsupported SQLTYPE was encountered in a select-list or input-list.
-426	2D528	Operation invalid for application run environment. This SQLSTATE reports the attempt to use EXCSQLIMM or EXCSQLSTT to execute a COMMIT in a dynamic COMMIT restricted environment.
-427	2D529	Operation invalid for application run environment.
-501 -502 -507	24501	Execution failed due to an invalid cursor state. The identified cursor is not open.
-510	42828	This SQLSTATE reports an attempt to DELETE WHERE CURRENT OF CURSOR or UPDATE WHERE CURRENT OF CURSOR on a cursor that is fetching rows using a blocking protocol.
-525	51015	Statement is in error.
-551	42501	Not authorized to object &1 in &2 type *&3.
-552	42502	Not authorized to &1.
-683	42842	FOR DATA clause or CCSID clause not valid for specified type.
-752	0A001	Application process is not in a connectable state. Reason code &1.
-802	22003 22012	A numeric value is out of range and division by zero is invalid.
-805	51002	SQL package &1 in &2 not found.
-818	51003	Consistency tokens do not match.
-842	08002	The connection already exists.
-862	55029	Local program attempted to connect to remote relational database.
-871	54019	Too many CCSID values specified.
-900	08003	The connection does not exist.
-918	51021	SQL statements cannot be executed until the application process executes a rollback operation.
-922	42505	This SQLSTATE reports a failure to authenticate the user during connection processing to an application server.
-925 -926	2D521	SQL COMMIT or ROLLBACK statements are invalid in the current environment.

Table 32. SQLCODEs and SQLSTATEs (continued)

SQLCODE	SQLSTATE	Description
-950	42705	Relational database &1 not in relational directory.
-969	58033	Error occurred when passing request to application requester driver program.
-7017	42971	Commitment control is already active to a DDM target.
-7018	42970	COMMIT HOLD or ROLLBACK HOLD is not allowed.
-7021	57043	Local program attempting to run on application server.
-30000	58008	Distributed Relational Database Architecture (DRDA) protocol error.
-30001	57042	Call to distributed SQL program not allowed.
-30020	58009	Distributed Relational Database Architecture (DRDA) protocol error.
-30021	58010	Distributed relational database not supported by remote system.
-30040	57012	DDM resource &2 at relational database &1 unavailable.
-30041	57013	DDM resources at relational database &1 unavailable.
-30050	58011	DDM command &1 is not valid while bind process in progress.
-30051	58012	Bind process with specified package name and consistency token not active.
-30052	42932	Program preparation assumptions are incorrect.
-30053	42506	Not authorized to create package for owner &1.
-30060	08004	User not authorized to relational database &1.
-30061	08004	Relational database &1 not found.
-30070	58014	Distributed Data Management (DDM) command &1 not supported.
-30071	58015	Distributed Data Management (DDM) object &1 not supported.
-30072	58016	Distributed Data Management (DDM) parameter &1 not supported.
-30073	58017	Distributed Data Management (DDM) parameter value &1 not supported.
-30074	58018	Distributed Data Management (DDM) reply message &1 not supported.
-30080	08001	Communication error occurred during distributed database processing.
-30082	08001	Authorization failure on distributed database connection attempt.
-30090	25000 2D528 2D529	Change request not valid for read-only application server.

Table 32. SQLCODEs and SQLSTATEs (continued)

SQLCODE	SQLSTATE	Description
-30104	56095	Bind option not valid. This SQLSTATE reports that one or more bind options were not valid at the system. The bind operation ends. The first bind option in error is reported in SQLERRMC.
-30105	56096	Conflicting bind options. The bind operation terminates. The bind options in conflict are reported in SQLERRMC.
Unrecognized by AR	58020	SQLSTATE value not defined for the error or warning.

Related concepts

SQL messages and codes

Related reference

Display Message Description (DSPMSGD) command

Finding first-failure data capture data

You can use the tips in this topic to locate first-failure data capture (FFDC) data on the IBM i operating system. The information is most useful if the failure causing the FFDC data output occurred on the server. The FFDC data for a client can usually be found in one of the spooled files associated with the job running the application program.

Note: No FFDC data is produced unless the QSFWERRLOG system value is set to *LOG.

1. Execute a Display Messages (DSPMSG) QSYSOPR command and look for a `Software problem detected in Qccxyyyy` message in the QSYSOPR message log. (`cc` in the program name is usually RW, but could be CN or SQ.) The presence of this message indicates that FFDC data was produced. You can use the help key to get details on the message. The message help gives you the problem ID, which you can use to identify the problem in the list presented by the Work with Problems (WRKPRB) command. You might be able to skip this step because the problem record, if it exists, might be at or near the top of the list.
2. Enter the Work with Problems (WRKPRB) command and specify the program name (Qccxyyyy) from the `Software problem detected in Qccxyyyy` message. Use the program name to filter out unwanted list items. When a list of problems is presented, specify option 5 on the line containing the problem ID to get more problem details, such as symptom string and error log ID.
3. When you have the error log ID, enter the Start System Service Tools (STRSST) command.

   On the first screen, select `Start a service tool`. On the next screen, enter 1 to select `Error log utility`. On the next screen, enter 2 to select `Display or print by error log ID`. In the next screen, you can:

   - Enter the error log ID.
   - Enter Y to get the hexadecimal display.
   - Select the Print or Display option.

The Display option gives 16 bytes per line instead of 32. This can be useful for online viewing and printing screens on an 80-character workstation printer. If you choose the Display option, use F6 to see the hexadecimal data after you press Enter.

The hexadecimal data contains the first one KB of the FFDC dump data, preceded by some other data. The start of the FFDC data is identified by the FFDC data index. The name of the target job (if this is on the server) is before the data index. If the FFDC dump spool file has not been deleted, use this fully qualified job name to find the spool file. If the spool file is missing, either:

- Use the first one KB of the dump stored in the error log.
- Re-create the problem if the one KB of FFDC data is insufficient.

When interpreting FFDC data from the error log, the FFDC data in the error log is not formatted for reading as well as the data in the spooled files. Each section of the FFDC dump in the error log is prefixed by a four-byte header. The first two bytes of the header are the length of the following section (not counting the prefix). The second two bytes, which are the section number, correspond to the section number in the index.

Related reference

Display Messages (DSPMSG) command

Work with Problem (WRKPRB) command

Start System Service Tools (STRSST) command

Getting data to report a failure

The IBM i licensed program provides data that you can print to help you diagnose a problem in a distributed relational database.

You can also use system operator messages and the application program (along with its data) to diagnose problems.

Printing a job log

Every IBM i job has a job log that contains information related to requests entered for that job. When a user is having a problem at a client, the information in the job log might be helpful in diagnosing the problem.

One easy way to get this information is to have the user sign off with the command:

```
SIGNOFF *LIST
```

This command prints a copy of the user's job log, or places it in an output queue for printing.

Another way to print the job log is by specifying LOG(4 00 *SECLVL) on the application job description. After the job is finished, all messages are logged to the job log for that specific job. You can print the job log by locating it on an output queue and running a print procedure.

The job log for the server might also be helpful in diagnosing problems.

Related tasks

Tracking request information with the job log of a distributed relational database
Every IBM i job has a job log that contains information related to requests entered for a job.

Locating distributed relational database jobs
When you are looking for information about a distributed relational database job on a client and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

Finding job logs from TCP/IP server prestart jobs

When the connection ends that is serviced by one of the QRWTSRVR prestart jobs associated with the distributed data management (DDM) TCP/IP server, the prestart job is recycled for use by another connection. When this happens, the job log associated with the ended connection is cleared.

However, in certain cases the job log is spooled to a printer file before it is cleared. The job log is not printed to a spooled file if the client user ID and password were not successfully validated. If validation was successful, these are the conditions under which the job log is printed to a spooled file:

- If the operating system is at V5R1 or later and the server job's message logging text level is *SECLVL or *MSG.
- If the system-request-handler routing program detects that a serious error occurred in processing the request that ended the connection.
- If the prestart job was being serviced (by use of the Start Service Job (STRSRVJOB) command).
- If the QRWOPTIONS data area on the client or server specified a job log output condition that was satisfied by the server job.

You might want to get job log information for several reasons. It is obviously useful for diagnosing errors. It can also be useful for analyzing performance problems. For example, if you want to get SQL optimizer data that is generated when running under debug, you can either manually start a service job and run the Start Debug (STRDBG) command, or you can set one or more applicable options in the QRWOPTIONS data area to cause the job log to be retained.

The logs of jobs in which failures occur during the connection phase will not be saved by the process described above. Jobs that are saved by that process will not be stored under the prestart job ID. To find them, run the following command:

```
WRKJOB userid/QPRTJOB
```

where *userid* is the user ID used on the CONNECT to the server. You can find that user ID if you do not know it with the Display Log (DSPLOG) command on the server.

You can filter out unwanted messages by use of parameters like this:

```
DSPLOG PERIOD(('11:00')) MSGID(CPI3E34)
```

Look for the following message. Note, however, that if the QRWOPTIONS data area has been used to suppress this message (CPI3E34), it will not appear in the history log.

```
DDM job xxxx servicing user yyy on ddd at ttt.
```

Related concepts

QRWOPTIONS data area
When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related reference

Start Service Job (STRSRVJOB) command
Display Log (DSPLOG) command
Start Debug (STRDBG) command

Printing the product activity log

The IBM i product activity log is a record of machine checks, device errors, and tape statistics. It also contains first-failure data capture (FFDC) information including the first 1000 bytes of each FFDC dump. By reviewing these errors, you might be able to determine the nature of a problem.

To print the product activity log for a system on which you are signed on, follow these steps:

1. Type the **`Print Error Log (PRTERRLOG)`** command on any command line and press F4 (Prompt). The Print Error Log display is shown.
2. Type the parameter value for the kind of log information you want to print and press the Enter key. The log information is sent to the output queue identified for your job.
3. Enter the **`Work with Job (WRKJOB)`** command. The Work with Job display is shown.
4. Select the option to work with spooled files. The Work with Job Spooled Files display is shown.
5. Look for the log file you just created at or near the bottom of the spooled file list.
6. Type the work with printing status option in the *Opt* column next to the log file. The Work with Printing Status display is shown.
7. On the Work with Printing Status display, use the change status option to change the status of the file and specify the printer to print the file.

Related reference

Print Error Log (PRTERRLOG) command
Work with Job (WRKJOB) command

Job tracing

Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, IBM i data acquisition and CL commands executed.

These tools should be used when the problem analysis procedures do not supply sufficient information about the problem. For distributed database applications, these commands are also useful for capturing distributed database requests and response data streams.

If you need to get a job trace of the server job, you need to start a service job at the server.

Related concepts

Starting a service job to diagnose application server problems
When an application uses Distributed Relational Database Architecture (DRDA), the SQL statements are run in the application server job. Because of this, you might need to start debug or a job trace for the application server job that is running on the IBM i operating system. The technique for doing this differs based on the use of either Advanced Program-to-Program Communication (APPC) or TCP/IP.

Communications trace
If you get a message in the CPF3E*xx* range or the CPF91*xx* range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Trace job

The Trace Job (TRCJOB) command is the older of the two tracing tools. As the trace records are generated, the records are stored in an internal trace storage area. When the trace is ended, the trace records can be written to a spooled printer file (QPSRVTRC) or directed to a database output file.

A sample trace scenario is as follows:

```
TRCJOB SET(*ON) TRCTYPE(*ALL) MAXSTG(2000)
       TRCFULL(*WRAP) EXITPGM($SCFTRC)
CALL QCMD
TRCJOB SET(*OFF) OUTPUT(*PRINT)
WRKOUTQ output-queue-name
```

You will see a spooled file with a name of QPSRVTRC. The spooled file contains your trace.

Related reference

Trace Job (TRCJOB) command

Start trace

You can also use the Start Trace (STRTRC) command to perform traces. The STRTRC command is more flexible and less intrusive than the Trace Job (TRCJOB) command. It allows tracing across multiple jobs and shows more in-depth details about the module flow.

As the trace records are generated, the records are stored in an internal trace storage area that is identified by a session ID. When the trace is ended using End Trace (ENDTRC), the trace records are placed in a user-specified library as a set of database files. These files can then be written to a spooled printer file (QPSRVTRCJ) or directed to a database output file by issuing the PRTTRC.

A sample trace scenario is as follows:

```
STRTRC SSNID(DRDATRACE) JOB((*ALL/QUSER/QRWTSRVR)) MAXSTG(160000)
        TRCFULL(*STOPTRC)
```

Run the failing DRDA scenario:

```
ENDTRC SSNID(DRDATRACE) DTALIB(TRACELIB)
PRTTRC DTAMBR(DRDATRACE) DTALIB(TRACELIB)
```

Related reference

End Trace (ENDTRC) command

Start Trace (STRTRC) command

Communications trace

If you get a message in the CPF3E*xx* range or the CPF91*xx* range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

The following list shows common messages you might see in these ranges.

Table 33. Communications trace messages

MSG ID	Description
CPF3E80	DDM data stream syntax error.
CPF91*xx*	DDM protocol error.
CPF3E83	Invalid FD0:CA descriptor.
CPF3E84	Data mismatch error.

You can perform two types of communications traces. The first is the standard communications trace. The second is the TRCTCPAPP function. The TRCTCPAPP function provides for intelligible traces where IPSec or the secure sockets protocol has encrypted the data streams. It captures the data before encryption and after decryption. However, it also works well for getting traces of unencrypted data streams. It is required for getting traces of intra-system DRDA flows where LOOPBACK is used.

Related concepts

Connection security protocols
Several connection security protocols are supported by the current Db2 for i implementation of distributed data management (DDM) or Distributed Relational Database Architecture (DRDA) over TCP/IP.

Job tracing
Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, IBM i data acquisition and CL commands executed.

Related tasks

TCP/IP communications trace
One of the uses of the trace tool is to show the clear text of a transmission in an environment where the data is encrypted.

Standard communications trace

The communications trace function lets you start or stop a trace of data on communications configuration objects. After you have run a trace of data, you can format the data for printing or viewing. You can view the printer file only in the output queue.

Communication trace options run under system service tools (SST). SST lets you use the configuration objects while communications trace is active. You can trace and format data for any of the communications types that you can use in a distributed database network.

You can run the communications trace from any display that is connected to the system. Anyone with the special authority (SPCAUT) of *SERVICE can run the trace on the IBM i operating system. Communications trace supports all line speeds.

You should use communications trace in the following situations:

- The problem analysis procedures do not supply sufficient information about the problem.
- You suspect that a protocol violation is the problem.
- You suspect a line noise to be the problem.
- The error messages indicate that there is a Systems Network Architecture (SNA) bind problem.

You must have detailed knowledge of the line protocols that you use to correctly interpret the data that is generated by a communications trace.

Whenever possible, start the communications trace before varying on the lines. This gives you the most accurate sample of your line as it varies on.

To run an APPC trace and to work with its output, you have to know on what line, controller, and device you are running.

To format and avoid unwanted data in the output of a TCP/IP trace, you can specify the IP addresses of the source and servers. Sometimes it is sufficient to just specify the port number instead, which is easier.

The following commands start, stop, print, delete, and dump communications traces:

Start Communications Trace (STRCMNTRC) command

Starts a communications trace for a specified line or network interface description. Specify *MAX for value of Beginning bytes in Number of bytes to trace parameter. A communications trace continues until you run the End Communications Trace (ENDCMNTRC) command.

For example, you can enter this command to start the communications trace for your Ethernet line:

```
STRCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN) MAXSTG(20000) USRDTA(*MAX)
```

To determine what Ethernet line you have, enter this command:

```
WRKCFGSTS *LIN
```

End Communications Trace (ENDCMNTRC) command

Ends the communications trace running on the specified line or network interface description.

Enter the following command to end the communications trace for your Ethernet line:

```
ENDCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN)
```

Dump Communications Trace (DMPCMNTRC) command

Copies the unformatted trace data for the specified line, network interface description, or network server description to a user-specified stream file.

You can dump the communications trace for your Ethernet line to an integrated file system file: the extra header information is dumped to the integrated file system file.

Note: The STRDST communications trace cannot be dumped so it does not show the header information in the output.

Enter the following command to dump the communications trace for your Ethernet line:

```
DMPCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN) TOSTMF(COMMTRACE)
```

Print Communications Trace (PRTCMNTRC) command

Moves the communications trace data for the specified line or network interface description to a spooled file or an output file. Specify *YES for the format SNA data only parameter.

To print the communications trace to a spooled file from your integrated file system file on port 446 on the server, enter the following command:

```
PRTCMNTRC FROMSTMF(COMMTRACE) SLTPORT(446) FMTBCD(*NO)
```

Delete Communications Trace (DLTCMNTRC) command

Deletes the communications trace for a specified line or network interface description.

To delete the communication trace from your integrated file system file filtering on port 446 on the server, enter the following command:

```
DLTCMNTRC CFGOBJ(ETHLINE) CFGTYPE(*LIN)
```

To delete the integrated file system stream file, enter the following commands:

```
qsh
rm commtrace
```

Related concepts

Communications Management PDF

Related reference

Delete Communications Trace (DLTCMNTRC) command

End Communications Trace (ENDCMNTRC) command

Print Communications Trace (PRTCMNTRC) command

Start Communications Trace (STRCMNTRC) command

Finding your line, controller, and device descriptions

Use the Work with Configuration Status (WRKCFGSTS) command to find the controller and device under which your server job starts.

For example:

```
WRKCFGSTS CFGTYPE(*DEV)
          CFGD(*LOC)
          RMTLOCNAME(DB2ESYS)
```

The value for the RMTLOCNAME keyword is the server's name.

The Work with Configuration Status (WRKCFGSTS) command shows all of the devices that have the specified system name as the remote location name. You can tell which device is in use because you can vary on only one device at a time. Use option 8 to work with the device description and then option 5 to display it. The attached controller field gives the name of your controller. You can use the WRKCFGSTS command to work with the controller and device descriptions. For example:

```
WRKCFGSTS CFGTYPE(*CTL)
          CFGD(PCXZZ1205)    /* workstation */
WRKCFGSTS CFGTYPE(*CTL)
          CFGD(LANSLKM)      /* IBM i on token ring */
```

The CFGD values are the controller names that are acquired from the device descriptions in the first example in this topic.

The output from the Work with Configuration Status (WRKCFGSTS) command also includes the name of the line description that you need when working with communications traces. If you select option 8 and then option 5 to display the controller description, the active switched line parameter displays the name of the line description. The LAN remote adapter address gives the token-ring address of the remote system.

Another way to find the line name is to use the Work with Line Descriptions (WRKLIND) command, which lists all of the line descriptions for the system.

Related reference

Work with Configuration Status (WRKCFGSTS) command

Work with Line Descriptions (WRKLIND) command

TCP/IP communications trace

One of the uses of the trace tool is to show the clear text of a transmission in an environment where the data is encrypted.

The trace data is captured before encryption at the sender, and after encryption at the receiver. However, the trace tool is useful in other environments as well. You can only use this function when you are using TCP/IP for communication.

To use the Trace TCP/IP Application (TRCTCPAPP) command, you must have a user profile with *SERVICE special authority. To start the trace, enter the following line:

```
TRCTCPAPP *DDM
```

If you want to restrict the trace to a certain port, for example port 448 for SSL, follow this example:

```
TRCTCPAPP *DDM *ON RMTNETADR(*INET *N '255.255.255.255' 448)
```

After the communication that you are tracing has finished, run the following command and look at the resulting spooled file:

```
TRCTCPAPP *DDM *OFF
```

If you traced more than one connection, you will need to locate and match your spooled files to each QRWTSRVR job. The spooled file name is QZBSTRC (for OS/400 V5R2, or earlier) or QCNTRC (for IBM i V5R3, and later) and the job is QRW*xxxxxx*, where xxxxxx is the job number placed in the user data for the spooled file.

Restriction for use with *DDM application

When you use the **`Trace TCP/IP Application (TRCTCPAPP)`** command with the *DDM application, the maximum amount of data you can trace for a single sent or received message is limited to 6000 bytes.

Related concepts

Communications trace
If you get a message in the CPF3E*xx* range or the CPF91*xx* range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Related reference

Trace TCP/IP Application (TRCTCPAPP) command

TCP/IP communication trace formatting

The **`Trace TCP/IP Application (TRCTCPAPP)`** command can be used to break down DRDA and DDM flows into an easier-to-read logical representation. It also displays the information in ASCII which can be of help in unlike environments.

To request this formatting, enter the following while ending the communications trace:

```
TRCTCPAPP APP(*DDM) SET(*OFF) ARGLIST('lvl=2')
```

Here is an example of an unformatted trace, edited to fit the width of this topic:

```
0080D0010001007A 200100162110D9C3 C8C1E2D5E3E24040
4040404040404040 *..}....:......RCHASNTS *
0006210F2407000D 002FD8E3C4E2D8D3 F4F0F0000C112ED8
E2D8F0F5F0F3F000 *..........QTDSQL400....QSQ05030.*
0A00350006119C00 2500062121241E00 062120241E0010D1
2A01000000000000 *.......................J........*
0000000000001621 35C1D7D7D54BD3D7 F0F6F6C1C2B9191C
F706F90005213BF1 *.........APPN.LP066AB...7.9....1*
```

This is the same trace, formatted using TRCTCPAPP:

```
-Datastream-----------------------------------------------------------
DATA:             (ASCII)           (EBCDIC)
0080D0010001007A  200100162110D9C3  .8'.....a...b.êã .0}....:......RC
C8C1E2D5E3E24040  4040404040404040   ç ë+èë.......... HASNTS
0006210F2407000D  002FD8E3C4E2D8D3  .Lb.f"....éèàëé< ..........QTDSQL
F4F0F0000C112ED8  E2D8F0F5F0F3F000  .......éëé...... 400....QSQ05030.
0A00350006119C00  2500062121241E00  C...L.ó...LbbfK. ......ä.........
062120241E0010D1  2A01000000000000  LbafK..¢k....... .......J........
0000000000001621  35C1D7D7D54BD3D7  .......b. &&+.<& .........APPN.LP
F0F6F6C1C2B9191C  F706F90005213BF1  ... â¨.c.L...bB. 066AB¾..7.9....1
-Parsed---------------------------------------------------------------

RECV(AS) RQSDSS - Request Data Stream Structure

LL: 128 CORR: 0001 CHAINED: n CONT ON ERR: n SAME CORR FOR NEXT DSS: n

NM: ACCRDB - Access RDB

LL: 122 CP: 2001

NM: RDBNAM - Relational Database Name

LL: 22 CP: 2110
```

```
ASCII: êãç ë+èë..........
EBCDIC: RCHASNTS
NM: RDBACCCL - RDB Access Manager Class
LL: 6 CP: 210F
CODE POINT DATA: 2407
NAME: SQLAM - SQL Application Manager
NM: TYPDEFNAM - Data Type Definition Name
LL: 13 CP: 002F
ASCII: éèàëé<...
EBCDIC: QTDSQL400
NM: PRDID - Product-Specific Identifier
LL: 12 CP: 112E
DATA:                (ASCII)       (EBCDIC)
D8E2D8F0F5F0F3F0     éëé.....     QSQ05030
NM: TYPDEFOVR - TYPDEF Overrides
LL: 10 CP: 0035
NM: CCSIDSBC - CCSID for Single-Byte Characters
LL: 6 CP: 119C
DATA:      (ASCII)     (EBCDIC)
0025       ..           ..
NM:  STTDECDEL - Statement Decimal Delimiter
LL: 6 CP: 2121
CODE POINT DATA: 241E
NAME: DFTPKG - Package Default
NM: STTSTRDEL - Statement String Delimiter
LL: 6 CP: 2120
CODE POINT DATA: 241E
NAME: DFTPKG - Package Default
NM: SXXPRDDTA - Extended Product Data
LL: 16 CP: D12A
DATA:              (EBCDIC)
0100000000000000   00000000 ............
NM: CRRTKN - Correlation Token
LL: 22 CP: 2135
DATA:             (ASCII)            (EBCDIC)
C1D7D7D54BD3D7F0  F6F6C1C2B9191CF7  &&+.<&... â¨.c. APPN.LP066AB¾..7
06F9 L. .9
NM: TRGDFTRT - Target Default Value Return
LL: 5 CP: 213B
BOOLEAN: TRUE
```

Related reference

Trace TCP/IP Application (TRCTCPAPP) command

Starting a service job to diagnose application server problems

When an application uses Distributed Relational Database Architecture (DRDA), the SQL statements are run in the application server job. Because of this, you might need to start debug or a job trace for the application server job that is running on the IBM i operating system. The technique for doing this differs based on the use of either Advanced Program-to-Program Communication (APPC) or TCP/IP.

Related concepts

Job tracing

Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, IBM i data acquisition and CL commands executed.

Service jobs for APPC servers

When the Db2 for i application server recognizes a special transaction program name (TPN), it causes the application server to send a message to the system operator and then wait for a reply.

This allows you to issue a **`Start Service Job (STRSRVJOB)`** command that allows job trace or debug to be started for the application server job.

To stop the Db2 for i application server job and restart it in debug mode, follow these steps:

1. Specify QCNTSRVC as the TPN at the application requester. There is a different method of doing this for each platform. This topic collection describes the different methods. When the IBM i application receives a TPN of QCNTSRVC, it sends a CPF9188 message to QSYSOPR and waits for a G (for *go*) reply.
2. Before entering the G reply, use the **`Start Service Job (STRSRVJOB)`** command to start a service job for the application server job and put it into debug mode. (Request help on the CPF9188 message to display the jobname.)
3. Enter the **`Start Debug (STRDBG)`** command.
4. After starting debug for the application server job, reply to the QSYSOPR message with a G. After receiving the G reply, the application server continues with normal DRDA processing.
5. After the application runs, look at the application server job log to see the SQL debug messages.

Related concepts

Creating your own transaction program name and setting QCNTSRVC

To create your own transaction program name (TPN) and set QCNTSRVC, do the following tasks.

Related reference

Start Service Job (STRSRVJOB) command

Start Debug (STRDBG) command

Creating your own transaction program name and setting QCNTSRVC

To create your own transaction program name (TPN) and set QCNTSRVC, do the following tasks.

Related tasks

Service jobs for APPC servers

When the Db2 for i application server recognizes a special transaction program name (TPN), it causes the application server to send a message to the system operator and then wait for a reply.

Setting QCNTSRVC as a transaction program name on a Db2 for i application requester

Specify the QCNTSRVC on the TNSPGM parameter of the **`Add Relational Database Directory Entry (ADDRDBDIRE)`** or **`Change Relational Database Directory Entry (CHGRDBDIRE)`** command.

It might be helpful to make a note of the special transaction program name (TPN) in the text of the RDB directory entry as a reminder to change it back when you are finished with debugging.

Note: Transaction program names only work with the directory entries of SNA relational databases.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Creating your own transaction program name for debugging a Db2 for i application server job

You can create your own transaction program name (TPN) by compiling a CL program that contains debug statements and a TFRCTL QSYS/QCNTEDDM statement at the end. The advantage of this is that you do not need any manual intervention when making the connection.

See the following code for an example program:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 362.

```
PGM
   MONMSG CPF0000
      STRDBG UPDPROD(*YES)  PGM(CALL/QRWTEXEC) MAXTRC(9999)
      ADDBKP STMT(CKUPDATE) PGMVAR((*CHAR (SQLDA@))) OUTFMT(*HEX) +
              LEN(1400)
      ADDTRC PGMVAR((DSLENGTH ()) (LNTH ()) (FDODTA_LNTH ()))
      TRCJOB *ON TRCTYPE(*DATA) MAXSTG(2048) TRCFULL(*STOPTRC)
      TFRCTL QSYS/QCNTEDDM
ENDPGM
```

The TPN name in the RDB directory entry of the application requester (AR) is the name that you supply. Use the text field to provide a warning that the special TPN is in use, and be sure to change the TPN name back when you have done debugging.

Note: Transaction program names only work with SNA RDB entries.

When you change the TPN of an RDB, all connections from that AR will use the new TPN until you change it back. This might cause surprises for unsuspecting users, such as poor performance, long waits for operator responses, and the filling up of storage with debug data.

Setting QCNTSRVC as a transaction program name on a DB2 for VM client

Change the UCOMDIR NAMES file to specify QCNTSRVC in the TPN tag.

For example:

```
          :nick.RCHASLAI :tpn.QCNTSRVC
                                  :luname.VM4GATE RCHASLAI
                                  :modename.MODE645
                                  :security.NONE
```

Then issue SET COMDIR RELOAD USER.

Setting QCNTSRVC as a transaction program name on a DB2 for z/OS client

Update the SYSIBM.LOCATIONS table to specify QCNTSRVC in the TPN column for the row that contains the RDB-NAME of the Db2 for i server.

Setting QCNTSRVC as a transaction program name on a DB2 for Linux, UNIX, and Windows client

If you are working with DB2 for Linux, UNIX, and Windows and would like instructions on how to set up the TPN on this family of products, there is a Web page to help you.

See the Web page DB2 for Linux, UNIX, and Windows. There you can find the several books specific to different versions.

Service jobs for TCP/IP servers

The DDM TCP/IP server does not use transaction program names (TPNs) as the Advanced Program-to-Program Communication (APPC) server does. However, the use of prestart jobs by the TCP/IP server provides a way to start a service job in that environment.

Note, however, that with the introduction of the function associated with the QRWOPTIONS data area usage, you might not need to start a service job in many cases. That feature allows one to start traces and do other diagnostic functions. You might still need to start a service job if you need a trace of the connection phase of the job.

You can use the **Display Log (DSPLOG)** command to find the CPI3E34 message reporting the name of the server job being used for a given connection if the following statements are true:

- You do not need to trace the actions of the server during the connect operation
- You choose not to use the QRWOPTIONS function
- You have the ability to delay execution of the application requester (AR) job until you can do some setup on the server, such as from interactive SQL

You can then use the **Start Service Job (STRSRVJOB)** command.

If you do need to trace the connect statement, or do not have time to do manual setup on the server after the connection, you will need to anticipate what prestart job will be used for the connection before it happens. One way to do that is to prevent other users from connecting during the time of your test, if possible, and end all of the prestart jobs except one.

You can force the number of prestart jobs to be 1 by setting the following parameters on the Change Prestart Job Entry (CHGPJE) command for the QRWTSRVR job running in the QSYSWRK system (OS/400 V5R2, or earlier) or QUSRWRK (OS/400 V5R2, IBM i V5R3, and later) to the values specified here:

- Initial number of jobs: **1**
- Threshold: **1**
- Additional number of jobs: **0**
- Maximum number of jobs: **1**

If you use this technique, be sure to change the parameters back to values that are reasonable for your environment; otherwise, users will get the message that `'A connection with a remote socket was reset by that socket'` when trying to connect when the one prestart job is busy.

Related concepts

QRWOPTIONS data area
When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related reference

Start Service Job (STRSRVJOB) command

Change Prestart Job Entry (CHGPJE) command

Display Log (DSPLOG) command

QRWOPTIONS data area

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Note: The information in the data area must be entered in uppercase in CCSID 37 or 500.

The format of the data area is as follows:

Table 34. Data area format

Columns	Contents
1-15	Client IP address in dotted decimal format for use when I is specified as a switch value (ignored otherwise).
16	Reserved area ignored by server (can contain a character for human usability)
17–26	User profile name for comparison when U is specified as a switch value (ignored otherwise)
27	Switch to cause job log to be kept if set to A, I or U (see notes 1 and 2)

Table 34. Data area format (continued)

Columns	Contents
28	Switch to cause DSPJOB output to be printed if set to A, I or U (see notes 1 and 2)
29	Switch to cause job to be traced if set to A, I or U (see notes 1 and 2).
30	Switch to cause debug to be started for job if set to A, I or U (see note 1).
31	Switch to invoke the **Change Query Attributes (CHGQRYA)** command with a QRYOPTLIB value if set to A, I or U. The QRYOPTLIB value is extracted from columns 39-48 which must contain the name of the library containing the proper QAQQINI file (see Note 1) **Note:** If an I or A is specified in this column, QUSER must have *JOBCTL special authority for it to take effect.
32	Switch to shadow client debug options if set to A, I or U (see note 1).
33	Switch to use old TRCJOB instead of new STRTRC for job trace if set to T and column 29 requests tracing. **Note:** If this column is set to T, TRCJOB will be used for the job trace. Set it to blank or S to use STRTRC.
34	Set this to N to suppress CPI3E34 messages in the history log (This is available in OS/400 V5R1 only with PTF SI02613)
35	Switch to start special subroutine trace if set to A, I, or U (see notes 1 and 2).
36–38	Reserved
39–48	General data area (contains library name if the **Change Query Attributes (CHGQRYA)** command is triggered by the appropriate value in column 31)

Notes:

1. These are the switch values that activate the function corresponding to the column in which they appear:
   - A activates the function for all uses of the server job.
   - I activates the function if the client IP address specified in columns 1-15 matches that used on the connect attempt.
   - U activates the function if the user ID specified in columns 17-26 matches that used on the connect attempt.
2. To find the spooled files resulting from this function, use **Work with Job command (WRKJOB user-profile/QPRTJOB)**, where user-profile is the user ID used on the connect request. Take option 4 and you should see one or more of these files.

   Table 35. File list from WRKJOB user-profile/QPRTJOB command

File	Device or queue	User data
QPJOBLOG	QEZJOBLOG	QRWTSRVR
QPDSPJOB	PRT01	
QPSRVTRC	PRT01	

3. The file containing the special DRDA subroutine trace will be created in library QGPL, with a name in this format: QRWDB*mmddy*, where mm represents the month, *dd* the day, and *y* the last digit of the year in which the trace was recorded. Not all system programs are traced.

Related tasks

Displaying the history log
Each time a client user establishes a successful connection with a server job, that job is swapped to run under the profile of that client user.

Related reference

Distributed relational database messages
If an error message occurs at either a server or a client, the system message is logged on the job log to indicate the reason for the failure.

Finding job logs from TCP/IP server prestart jobs
When the connection ends that is serviced by one of the QRWTSRVR prestart jobs associated with the distributed data management (DDM) TCP/IP server, the prestart job is recycled for use by another connection. When this happens, the job log associated with the ended connection is cleared.

Service jobs for TCP/IP servers
The DDM TCP/IP server does not use transaction program names (TPNs) as the Advanced Program-to-Program Communication (APPC) server does. However, the use of prestart jobs by the TCP/IP server provides a way to start a service job in that environment.

Change Query Attributes (CHGQRYA) command

Work with Job (WRKJOB) command

Example: CL command to create the data area

This example requests the functions indicated in the table.

```
CRTDTAARA DTAARA(QGPL/QRWOPTIONS) TYPE(*CHAR) LEN(48)
          VALUE('9.5.114.107    :MYUSERID  AAUIU TN    INILIBRARY')
          TEXT('DRDA TCP SERVER DIAGNOSTIC OPTIONS'
```

Note: Because the proper spacing in the example is critical, the contents of the VALUE parameter are repeated in table form.

Table 36. Explanation of data elements in VALUE parameter of CRTDTAARA example

Columns	Contents	Explanation
1–15	9.5.114.107	IP address for use with switch in column 30.
16	:	Character to mark the end of the IP address field.
17–26	MYUSERID	User ID for use with switches in columns 29 and 31.
27	A	Spool the server job log (for QRWTSRVR) for all users.
28	A	Spool the DSPJOB output for all uses of the server.
29	U	Trace the job with the (TRCJOB) command if the user ID on the connect request matches what is specified in columns 17 through 26 ('MYUSERID' in this example) of the data area.
30	I	Start debug with the **Start Debug (STRDBG)** command (specifying no program) if the client IP address ('9.5.114.107' in this example) matches what is specified in columns 1 through 15 of the data area.
31	U	Invoke the command **Change Query Attributes (CHGQRYA)** QRYOPTLIB(INILIBRARY) if the user ID on the connect request matches what is specified in columns 17 through 26 ('MYUSERID' in this example) of the data area. **Note:** The library name is taken from columns 39 through 48 of the data area.
32		Do not shadow client debug options to server.

Table 36. Explanation of data elements in VALUE parameter of CRTDTAARA example (continued)

Columns	Contents	Explanation
33	T	Use the old TRCJOB facility for job traces.
34	N	Do not place CPI3E34 messages in the history log.
35–38		Reserved bytes.
39–48	INILIBRARY	Library used with switch 31.

Related reference

Change Query Attributes (CHGQRYA) command

Start Debug (STRDBG) command

Working with distributed relational database users

Investigating a problem usually begins with the user. Users might not be getting the results they expect when running a program or they might get a message indicating a problem. Sometimes the best way to diagnose and solve a problem is to go through the procedure with a user.

The Copy screen function allows you to do this either in real time with the user or in examining a file of the displays the user saw.

You can also gather more information from Messages than just the line of text that appears at the bottom of a display. This topic collection discusses how you can copy displays being viewed by another user and how you can obtain more information about the messages you or a user receives when doing distributed relational database work.

In addition to programming problems, you might have problems when starting the program or connecting to the system if you use Advanced Program-to-Program Communication (APPC) or TCP/IP.

Copy screen

The **Start Copy Screen (STRCPYSCN)** command allows you to be at your workstation and see the same displays that are seen by someone at another workstation.

You must be signed on to the same IBM i platform as the user. If that user is on a remote system, you can use display station pass-through to sign on that system, and then enter the **STRCPYSCN** command to see the other displays. Screen images can be copied to a database file at the same time they are copied to another workstation or when another workstation cannot be used. This allows you to process this data later and prepares an audit trail for the operations that occur during a problem.

To copy the display image to another display station, the following requirements must be met:

- Both displays are defined to the system
- Both displays are color or both are monochrome, but not one color and the other monochrome
- Both displays have the same number of character positions horizontally and vertically

When you type your own display station ID as the sending device, the receiving display station must have the sign-on display shown when you start copying screen images. Graphics are copied as blanks.

If not already signed on to the same system, use the following process to see the displays that another user sees on a remote system:

1. Enter the **Start Pass-Through (STRPASTHR)** command:

   ```
   STRPASTHR RMTLOCNAME(KC105)
   ```

2. Log on to the application server (AS).
3. Enter the **STRCPYSCN** command using the SRCDEV, OUTDEV, and OUTFILE parameters. The parameters have the following meaning:

   - SRCDEV specifies the name of the source device, which is the display station that is sending the display image. To send your display to command to another device, enter the *REQUESTER value for this parameter.
   - OUTDEV specifies the name of the output device to which the display image is sent. In this example, the display image is sent to the display station of the person who enters the command (*REQUESTER). You can also name another display station, another device (where a third user is viewing), or to no other device (*NONE). When the *NONE value is used, specify an output file for the display images.
   - OUTFILE specifies the name of the output file that will contain an image of all the displays viewed while the command is active.

   ```
   STRCPYSCN SRCDEV(KC105)
             OUTDEV(*REQUESTER)
             OUTFILE(KCHELP/TEST)
   ```

   An inquiry message is sent to the source device to notify the user of that device that the displays will be copied to another device or file.
4. Type a g (Go) to start sending the images to the requesting device.

The sending display station's screens are copied to the other display station. The image shown at the receiving display station trails the sending display station by one screen. If the user at the sending display station presses a key that is not active (such as the Home key), both display stations will show the same display.

While you are copying screens, the operator of the receiving display station cannot do any other work at that display station until the copying of screens is ended.

To end the copy screen function from the sending display station, enter the **End Copy Screen (ENDCPYSCN)** command from any command line and press the Enter key.

```
ENDCPYSCN
```

The display you viewed when you started the copy screen function is shown.

Related reference

Start Pass-Through (STRPASTHR) command

Messages

The IBM i operating system sends a variety of system messages that indicate conditions ranging from simple typing errors to problems with system devices or programs.

Listed here are the messages you might get:

- An error message on your current display.

  These messages can interrupt your job or sound an alarm. You can display these messages by entering DSPMSG on any command line.
- A message regarding a system problem that is sent to the system-operator message queue and displayed on a separate Work with Messages display.

  To see these messages, enter DSPMSG QSYSOPR on any system command line.
- A message regarding a system problem that is sent to the message queue specified in a device description.

  To see these messages, enter DSPMSG message-queue-name on any system command line.

The system sends informational or inquiry messages for certain system events. *Informational messages* give you status about the system. *Inquiry messages* give you information about the system, and also require a reply.

In some message displays, a message is accompanied by a letter and number code such as:

```
CPF0083
```

The first two or three letters indicate the message category. The remaining four digits (five digits if the prefix is SQ) indicate the sequence number of the message. In the preceding example, the preceding message ID that is shown indicates that this is a message number 0083 from the operating system. Some message categories for distributed relational database are:

Table 37. Message categories

Category	Description	Library
CPA through CPZ	Messages from the operating system	QSYS/QCPFMSG
MCH	Licensed internal code messages	QSYS/QCPFMSG
SQ and SQL	Structured Query Language (SQL) messages	QSYS/QSQLMSG
TCP	TCP/IP messages	QTCP/QTCPMSGF

To obtain more information about a message, on the message line of a display or in a message queue, follow these steps:

1. Move the cursor to the same line as the message.
2. Press the Help key. The Additional Message Information display is shown.

You can get more information about a message that is not showing on your display if you know the message identifier and the library in which it is located. To get this information, enter the **`Display Message Description (DSPMSGD)`** command:

```
DSPMSGD RANGE(SQL0204) MSGF(QSYS/QSQLMSG)
```

You can view the following information about a message using the DSPMSDG command:

- Message text
- Field data
- Message attributes

The text is the same message and message help text that you see on the Additional Message Information display. The field data is a list of all the substitution variables that are defined for the message and the attributes of the variables. The message attributes are the values (when defined) for severity, logging, level of message, default program, default reply, and dump parameters. You can use the information to help you determine what the user was doing when the message appeared.

Related reference

Display Message Description (DSPMSGD) command

Message types

On the Additional Message Information display, you see the message type and severity code for the message.

The following table shows the different message types for IBM i messages and their associated severity codes.

Table 38. Message severity codes

Message type	Severity code
Informational messages. For informational purposes only; no reply is needed. The message can indicate that a function is in progress or that a function has completed successfully.	00
Warning. A potential error condition exists. The program might have taken a default, such as supplying missing data. The results of the operation are assumed to be successful.	10
Error. An error has been found, but it is one for which automatic recovery procedures probably were applied; processing has continued. A default might have been taken to replace the wrong data. The results of the operation might not be correct. The function might not have completed; for example, some items in a list ran correctly, while other items did not.	20
Severe error. The error found is too severe for automatic recovery procedures and no defaults are possible. If the error was in the source data, the entire data record was skipped. If the error occurred during a program, it leads to an abnormal end of program (severity 40). The results of the operation are not correct.	30
Severe error: abnormal end of program or function. The operation has ended, possibly because the program was not able to handle data that was not correct or because the user canceled it.	40
Abnormal end of job or program. The job was not started or failed to start, a job-level function might not have been done as required, or the job might have been canceled.	50
System status. Issued only to the system operator message queue. It gives either the status of or a warning about a device, a subsystem, or the system.	60
Device integrity. Issued only to the system operator message queue, indicating that a device is not working correctly or is in some way no longer operational.	70
System alert and user messages. A condition exists that, although not severe enough to stop the system now, could become more severe unless preventive measures are taken.	80
System integrity. Issued only to the system operator message queue. Describes a condition where either a subsystem or system cannot operate.	90

Table 38. Message severity codes (continued)

Message type	Severity code
Action. Some manual action is required, such as entering a reply or changing printer forms.	99

Distributed relational database messages

If an error message occurs at either a server or a client, the system message is logged on the job log to indicate the reason for the failure.

A system message exists for each SQLCODE returned from an SQL statement supported by the Db2 for i program. The message is made available in precompiler listings, on interactive SQL, or in the job log when you are running in debug mode. However, when you are working with an AS that is not a IBM i product, there might not be a specific message for every error condition in the following cases:

- The error is associated with a function not used by the IBM i product.

  For example, the special register CURRENT SQLID is not supported by Db2 for i, so SQLCODE -411 (SQLSTATE 56040) `CURRENT SQLID cannot be used in a statement that references remote objects` does not exist.
- The error is product-specific and will never occur when using Db2 for i.

  Db2 for i will never have SQLCODE -925 (SQLSTATE 56021), `SQL commit or rollback is invalid in an IMS or CICS environment.`

For SQLCODEs that do not have corresponding messages, a generic message is returned that identifies the unrecognized SQLCODE, SQLSTATE, and tokens, along with the relational database name of the AS which generated the message. To determine the specific condition and how to interpret the tokens, consult the product documentation corresponding to the particular release of the connected AS.

Messages in the ranges CPx3E00 through CPx3EFF and CPI9100 through CPI91FF are used for distributed relational database messages. The following list is not inclusive, but shows more common messages you might see in a distributed database job log on the IBM i operating system.

Table 39. Distributed relational database messages

MSG ID	Description
CPA3E01	Attempt to delete *LOCAL RDB directory entry
CPC3EC5	Some parameters for RDB directory entry ignored
CPD3E30	Conflicting remote network ID specified
CPD3E35	Structure of remote location name not valid for ...
CPD3E36	Port identification is not valid
CPD3E38	Type conflict for remote location
CPD3E39	Value &3 for parameter &2 not allowed
CPD3E3B	Error occurred retrieving server authorization information for ...
CPD3ECA	RDB directory operation may not have completed
CPD3E01	DBCS or MBCS CCSID not supported.
CPD3E03	Local RDB name not in RDB directory
CPD3E05	DDM conversation path not found
CPD3E31	DDM TCP/IP server is not active

Table 39. Distributed relational database messages (continued)

MSG ID	Description
CPD3E32	Error occurred ending DDM TCP/IP server
CPD3E33	DDM TCP/IP server error occurred with reason code ...
CPD3E34	DDM TCP/IP server communications error occurred
CPD3E37	DDM TCP/IP get host by name failure
CPF9190	Authorization failure on DRDA/DDM TCP/IP connection attempt
CPF3E30	Errors occurred starting DDM TCP/IP server
CPF3E31	Unable to start DDM TCP/IP server
CPF3EC6	Change DDM TCP/IP attributes failed
CPF3EC9	Scope message for interrupt RDB
CPF3E0A	Resource limits error
CPF3E0B	Query not open
CPF3E0C	FDOCA LID limit reached
CPF3E0D	Interrupt not supported
CPF3E01	DDM parameter value not supported
CPF3E02	AR cannot support operations
CPF3E04	SBCS CCSID not supported
CPF3E05	Package binding not active
CPF3E06	RDB not found
CPF3E07	Package binding process active
CPF3E08	Open query failure
CPF3E09	Begin bind error
CPF3E10	AS does not support DBCS or MC
CPF3E12	Commit/rollback HOLD not supported
CPF3E13	Commitment control operation failed
CPF3E14	End RDB request failed
CPF3E16	Not authorized to RDB
CPF3E17	End RDB request is in progress
CPF3E18	COMMIT/ROLLBACK with SQLCA
CPF3E19	Commitment control operation failed
CPF3E20	DDM conversation path not found
CPF3E21	RDB interrupt fails
CPF3E22	Commit resulted in a rollback at the server
CPF3E23	DDM data stream violates conversation capabilities
CPF3E30	Errors occurred starting DDM TCP/IP server

Table 39. Distributed relational database messages (continued)

MSG ID	Description
CPF3E32	Server error occurred processing client request
CPF3E80	Data stream syntax error
CPF3E81	Invalid FDOCA descriptor
CPF3E82	ACCRDB sent twice
CPF3E83	Data mismatch error
CPF3E84	DDM conversational protocol error
CPF3E85	RDB not accessed
CPF3E86	Unexpected condition
CPF3E87	Permanent agent error
CPF3E88	Query already open
CPF3E89	Query not open
CPF3E99	End RDB request has occurred
CPI9150	Database conversation started over SNA.
CPI9151	Database conversation ended over SNA.
CPI9152	Target DDM job started by source system.
CPI9160	Database connection started over TCP/IP.
CPI9161	Database connection ended over TCP/IP.
CPI9162	Target job assigned to handle DDM connection started by source system over TCP/IP.
CPI3E01	Local RDB accessed successfully
CPI3E02	Local RDB disconnected successfully
CPI3E04	Connection to relational database &1; ended
CPI3E30	DDM TCP/IP server already active
CPI3E31	DDM TCP/IP server does not support security mechanism
CPI3E32	DDM server successfully started
CPI3E33	DDM server successfully ended
CPI3E34	DDM job xxxx servicing user yyy on mm/dd/yy at hh:mm:ss (This can be suppressed with QRWOPTIONS)
CPI3E35	No DDM server prestart job entry
CPI3E36	Connection to relational database xxxx ended
SQ30082	A connection attempt failed with reason code...
SQL7992	Connect completed over TCP/IP
SQL7993	Already connected

Related concepts

QRWOPTIONS data area

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

SQL messages and codes

Related tasks

Tracking request information with the job log of a distributed relational database
Every IBM i job has a job log that contains information related to requests entered for a job.

Handling program start request failures for APPC

When a program start request is received by an IBM i subsystem on the server system, the server attempts to start a job based on the information sent with the program start request. The client user's authority to the server, existence of the requested database, and many other items are checked.

If the server subsystem determines that it cannot start the job (for example, the user profile does not exist on the server, the user profile exists but is disabled, or the user is not properly authorized to the requested objects on the server), the subsystem sends a message, CPF1269, to the QSYSMSG message queue (or QSYSOPR when QSYSMSG does not exist). The CPF1269 message contains two reason codes (one of the reason codes might be zero, which can be ignored).

The nonzero reason code gives the reason why the program start request was rejected. Because the remote job was to have started on the server, the message and reason codes are provided on the server, and not the client. The user at the client only knows that the program start request failed, not why it failed. The user on the client must either talk to the system operator at the server, or use display station pass-through to the server to determine the reason why the request failed.

Related concepts

ICF Programming PDF

Handling connection request failures for TCP/IP

The main causes for failed connection requests at a Distributed Relational Database Architecture (DRDA) server configured for TCP/IP use is that the DDM TCP/IP server is not started, an authorization error occurred, or the machine is not running.

Server is not started or the port ID is not valid

The error message given if the DDM TCP/IP server is not started is CPE3425.

The message is:

```
A remote host refused an attempted connect operation.
```

You can also get this message if you specify the wrong port on the **Add Relational Database Directory Entry (ADDRDBDIRE)** or the **Change Relational Database Directory Entry (CHGRDBDIRE)** command. For a Db2 for i server, the port should usually be *DRDA (the DRDA well-known port of 446). However, if you have configured port 447 for use with IPSec, you might want to use that port for transmitting sensitive data. If you are using a DRDA client that supports Secure Sockets Layer (SSL), you must connect to port 448 on the server.

To start the DDM server on the remote system, run the **Start TCP/IP Server (STRTCPSVR)** *DDM command. You can request that the DDM server be started whenever TCP/IP is started by running the Change DDM TCP/IP Attributes (CHGDDMTCPA) AUTOSTART(*YES) command.

Related concepts

Secure Sockets Layer
Db2 for i Distributed Relational Database Architecture (DRDA) clients & servers support Secure Sockets Layer (SSL). A similar function is available with Internet Protocol Security Architecture (IPSec).

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Start TCP/IP Server (STRTCPSVR) command

DRDA connection authorization failure

The error messages given for an authorization failure is SQ30082.

The message text is:

```
Authorization failure on distributed database connection attempt.
```

The cause section of the message gives a reason code and a list of meanings for the possible reason codes. Reason code 17 means that there was an unsupported security mechanism.

Db2 for i implements several Distributed Relational Database Architecture (DRDA) security mechanisms that an IBM i application requester (AR) can use:

- User ID only
- User ID with password
- Encrypted password security mechanism
- Encrypted user ID and password security mechanism
- Kerberos

The encrypted password is sent only if a password is available at the time the connection is initiated.

The default security mechanism for IBM i is the user ID with password mechanism. With the default security configuration, if the application requester sends a user ID with no password to the system, error message SQ30082 with reason code 17 is displayed.

Solutions for the unsupported security mechanism failure are:

- If the client is trusted by the server and authentication is not required, change the DDM TCP/IP server's authentication setting to password not required.
- If the client is not trusted by the server and authentication is required, change the application to send either a password or authenticated security token (for example, a Kerberos token).

To change the authentication setting of the DDM TCP/IP server, you can use the Change DDM TCP/IP Attributes (CHGDDMTCPA) command or System i Navigator. If you use System i Navigator, expand **Network** > **Servers** > **TCP/IP** > **DDM**, right-click **DDM**, and select **Properties** to change the setting.

You can send a password by using the USER/USING form of the SQL CONNECT statement. You can also send a password by using the Add Server Authentication Entry (ADDSVRAUTE) command. The command adds the remote user ID and the password in a server authentication entry for the user profile that you use to make a connection attempt. An attempt is automatically made to send the password encrypted.

Note that you have to have system value QRETSVRSEC (retain server security data) set to 1 to be able to store the remote password in the server authentication entry.

Note: You must enter the RDB name on the Add Server Authentication Entry (ADDSVRAUTE) command in uppercase for use with DRDA or the name will not be recognized during the connection processing and the information in the authentication entry will not be used.

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

System not available

If a remote system is not up and running, or if you specify an incorrect IP address in the RDB directory entry for the server, you will receive message CPE3447.

The message text is:

```
A remote host did not respond within the timeout period.
```

There is normally a several minute delay before this message occurs. It might appear that something is hung up or looping during that time.

Connection failures specific to interactive SQL

Sometimes when you are running a CONNECT statement from interactive SQL, a general SQ30080 message is given.

The text of that message is:

```
Communication error occurred during distributed database processing
```

In order to get the details of the error, you should exit from Interactive SQL and look at the job log.

If you receive message SQL7020, `SQL package creation failed`, when connecting for the first time (for any given level of commitment control) to a system that has only single-phase-commit capabilities, the cause might be that you accessed the remote system as a read-only system and you need to update it to create the SQL package.

You can verify that by looking at the messages in the job log. The solution is to do a RELEASE ALL and COMMIT to get rid of all connections before connecting, so that the connection will be updatable.

Related tasks

Setting up SQL packages for interactive SQL
This topic applies only to server systems that are not running IBM i.

Not enough prestart jobs at server

If the number of prestart jobs associated with the TCP/IP server is limited by the QRWTSRVR prestart job entry of the QSYSWRK (OS/400 V5R1, or earlier) or (OS/400 V5R2, IBM i V5R3, and later) QUSRWRK subsystem, and all prestart jobs are being used for a connection, an attempt at a new connection fails with these messages.

CPE3426
A connection with a remote socket was reset by that socket.

CPD3E34
DDM TCP/IP communications error occurred on recv() — MSG_PEEK.

You can avoid this problem at the server by setting the MAXJOBS parameter of the Change Prestart Job Entry (CHGPJE) command for the QRWTSRVR entry to a higher number or to *NOMAX, and by setting the ADLJOBS parameter to something other than 0.

Related reference

Change Prestart Job Entry (CHGPJE) command

Related information for Distributed database programming

Product manuals, IBM Redbooks publications, Web sites, and other information center topic collections contain information that relates to the Distributed database programming topic collection. You can view or print any of the PDF files.

IBM i information

These books and information center topics contain information you might need.

- The APPC, APPN, and HPR topic provides the application programmer with information about the Advanced Peer-to-Peer Networking (APPN) support provided by the IBM i operating system. This topic provides information for configuring an APPN network and presents considerations to apply when using APPN. This is also a guide for programming and defining the communications environment for APPC communications.
- The Application programming interfaces topic provides information about how to create, use, and delete objects that help manage system performance, use spooling efficiently, and maintain database files

efficiently. This topic also includes information about creating and maintaining the programs for system objects and retrieving IBM i information by working with objects, database files, jobs, and spooling.

- The Backup and recovery category provides the system programmer with information about the different media available to save and restore system data, as well as a description of how to record changes made to database files and how that information can be used for system recovery and activity report information.
- The CL programming topic provides a wide-ranging discussion of programming topics, including a general discussion of objects and libraries, control language (CL) programming, controlling flow and communicating between programs, working with objects in CL programs, and creating CL programs. Other topics include predefined and immediate messages and message handling, defining and creating user-defined commands and menus, and application testing, including debug mode, breakpoints, traces, and display functions.
- Communications Management contains information about working with communications status, communications-related work management topics, communications errors, performance, line speed and subsystem storage.
- The Control language topic provides the application programmer with a description of the IBM i control language (CL) and its commands.
- DSNX Support provides information for configuring a IBM i product to use the remote management support (distributed host command facility), the change management support (distributed systems node executive), and the problem management support (alerts).
- ICF Programming provides the application programmer with the information needed to write application programs that use IBM i communications and ICF files. It also contains information about data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and programming examples.
- ILE Concepts describes, for the application programmer, the concepts and terminology of the Integrated Language Environment of the IBM i operating system. It includes an overview of the ILE model; concepts of program creation, run time, and debugging; discussion of storage and condition management, and descriptions of calls and APIs.
- LAN, Frame-Relay and ATM Support contains information about using a IBM i product in a token-ring network, an Ethernet network, or a bridged network environment.
- Local Device Configuration provides the system operator or system administrator with information about how to do an initial local hardware configuration and how to change that configuration. It also contains conceptual information for device configuration, and planning information for device configuration on the 9406, 9404, and 9402 System Units.
- OptiConnect provides information about installing, using, and managing communications using OptiConnect.
- Query Management Programming provides the application programmer with information about determining the database files to be queried for a report, defining a Structured Query Language (SQL) query definition, and using and writing procedures that use query management commands. This manual also includes information about how to use the query global variable support and understanding the relationship between the query management function of the operating system and the IBM Query for IBM i.
- Remote Work Station Support provides information about how to set up and use remote workstation support, such as display station pass-through, distributed host command facility, and 3270 remote attachment. It also provides information for the application programmer or system programmer about configuration commands and defining lines, controllers, and devices.

- The Security topic provides the system programmer (or someone who is assigned the responsibilities of a security officer) with information about system security concepts, planning for security, and setting up security on the system.
- The SQL programming topic provides the application programmer, programmer, or database administrator with an overview of how to design, write, test and run SQL statements. It also describes interactive Structured Query Language (SQL).
- The SQL reference topic provides the application programmer, programmer, or database administrator with detailed information about SQL statements and their parameters.
- X.25 Network Support contains information about using IBM i products in an X.25 network.
- The following manuals are not included in the V6R1 IBM i Information Center. However, these manuals might be a useful reference to you. Each of the manuals is available from the IBM Publications Center as a printed hardcopy that you can order, in an online format that you can download at no charge, or both.
  - Communications Configuration
  - SNA Distribution Services

Distributed relational database library

The books in this topic provide background and general support information for IBM Distributed Relational Database Architecture (DRDA) implementations.

- *DRDA: Every Manager's Guide*, GC26-3195, provides concise, high-level education on distributed relational database and distributed file. This book describes how IBM supports the development of distributed data systems, and discusses some current IBM products and announced support for distributed data. The information in this book is intended to help executives, managers, and technical personnel understand the concepts of distributed data.
- *DRDA: Planning for Distributed Relational Database*, SC26-4650, helps you plan for distributed relational data. It describes the steps to take, the decisions to make, and the options from which to choose in making those decisions. The book also covers the distributed relational database products and capabilities that are now available or that have been announced, and it discusses IBM's stated direction for supporting distributed relational data in the future. The information in this book is intended for planners.
- *DRDA: Connectivity Guide*, SC26-4783, describes how to interconnect IBM products that support Distributed Relational Database Architecture. It explains concepts and terminology associated with distributed relational database and network systems. This book tells you how to connect unlike systems in a distributed environment. The information in the Connectivity Guide is not included in any product documentation. The information in this book is intended for system administrators, database administrators, communication administrators, and system programmers.
- *DRDA: Application Programming Guide*, SC26-4773, describes how to design, build, and modify application programs that access IBM's relational database management systems. This manual focuses on what a programmer should do differently when writing distributed relational database applications for unlike environments. Topics include program design, preparation, and execution, as well as performance considerations. Programming examples written in IBM C are included. The information in this manual is designed for application programmers who work with at least one of IBM's high-level languages and with Structured Query Language (SQL).
- *DRDA: Problem Determination Guide*, SC26-4782, helps you define the source of problems in a distributed relational database environment. This manual contains introductory material on each product, for people not familiar with those products, and gives detailed information on how to diagnose and report problems with each product. The guide describes procedures and tools unique to each host system and those common among the different systems. The information in this book is intended for the people who report distributed relational database problems to the IBM Support Center.
- *IBM SQL Reference, Volume 2*, SC26-8416, makes references to DRDA and compares the facilities of:
  - IBM SQL relational database products
  - IBM SQL

- ISO-ANSI SQL (SQL92E)
- X/Open SQL (XPG4-SQL)
- ISO-ANSI SQL Call Level Interface (CLI)
- X/Open CLI
- Microsoft Open Database Connectivity (ODBC) Version 2.0

Other IBM distributed relational database platform libraries

This topic describes other IBM distributed relational database platform libraries.

DB2 Connect and Universal Database

If you are working with DB2 Connect and Universal Database and would like more information, see the Web page DB2 for Linux, UNIX, and Windows. There you can find the following books:

- *DB2 Connect Enterprise Edition Quick Beginning*
- *DB2 Connect Personal Edition Quick Beginning*
- *DB2 Connect User's Guide*
- *DB2 UDB Administration Guide*
- *DB2 UDB Command Reference*
- *DB2 UDB for OS/2 Quick Beginnings*
- *DB2 UDB for UNIX Quick Beginnings*
- *DB2 UDB for Windows NT Quick Beginnings*
- *DB2 UDB Messages Reference*
- *DB2 UDB Personal Edition Quick Beginnings*
- *DB2 UDB SQL Getting Started*
- *DB2 UDB SQL Reference*
- *DB2 UDB Troubleshooting Guide*

DB2 for z/OS and OS/390

If you are working with DB2 for z/OS and OS/390® and would like more information, see the Web page DB2 for z/OS. There you can find the following books:

- *DB2 for z/OS and OS/390 Command Reference*
- *DB2 for z/OS and OS/390 Messages and Codes*
- *DB2 for z/OS and OS/390 Reference for Remote DRDA*
- *DB2 for z/OS and OS/390 SQL Reference*
- *DB2 for z/OS and OS/390 Utility Guide and Reference*

DB2 Server for VSE &VM

If you are working with DB2 Server for VSE & VM and would like more information, see the Web page DB2 Server for VSE & VM. There you can find the following books:

- *DB2 and Data Tools for VSE and VM*
- *DB2 for VM Control Center Installation*
- *DB2 Server Data Spaces Support for VM/ESA*
- *DB2 Server for VM Application Programming*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VM Database Services Utilities*

- *DB2 Server for VM Diagnosis Guide*
- *DB2 Server for VM Interactive SQL Guide*
- *DB2 Server for VM Master Index and Glossary*
- *DB2 Server for VM Messages and Codes*
- *DB2 Server for VM Operation*
- *DB2 Server for VM System Administration*
- *DB2 Server for VM/VSE Training Brochure*
- *DB2 Server for VSE & VM Quick Reference*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM LPS*
- *DB2 Server for VSE & VM Data Restore*
- *SBOF for DB2 Server for VM*

Architecture books

Listed here are the DDM and DRDA architecture books.

Distributed Data Management Architecture

- *Distributed Data Management Architecture: General Information*, GC21-9527
- *Distributed Data Management Architecture: Implementation Programmer's Guide*, SC21-9529
- *Distributed Data Management Architecture: Reference*, SC21-9526

DRDA Architecture

- *Character Data Representative Architecture: Details*, SC09-2190

  This manual includes a CD-ROM, which contains the two CDRA publications in online BOOK format, conversion tables in binary form, mapping source for many of the conversion binaries, a collection of code page and character set resources, and character naming information as used in IBM. The CD also includes a viewing utility to be used with the provided material. Viewer works with OS/2, Windows 3.1, and Windows 95.
- *Character Data Representative Architecture: Overview*, GC09-2207
- *DRDA V5 Vol. 1: Distributed Relational Database Architecture*

  This Technical Standard is one of three volumes documenting the Distributed Relational Database Architecture Specification. This volume describes the connectivity between relational database managers that enables applications programs to access distributed relational data. It describes the necessary connection between an application and a relational database management system in a distributed environment; the responsibilities of the participants and when flow should occur; and the formats and protocols required for distributed database management system processing. It does not describe an API for distributed database management system processing. This document is available on the Open Group Web site at https://collaboration.opengroup.org/dbiop/?gpid=453&gpid=453&gpid=453.
- *DRDA V5 Vol. 2: Formatted Data Object Content Architecture*

  This document is one of three Technical Standards documenting the Distributed Relational Database Architecture, Version 4. This volume describes the functions and services that make up the Formatted Data Object Content Architecture (FD:OCA). This architecture makes it possible to bridge the connectivity gap between environments with different data types and data representations methods. FD:OCA is embedded in DRDA. This document is available on the Open Group Web site at https://collaboration.opengroup.org/dbiop/?gpid=453&gpid=453&gpid=453.
- *DRDA V5 Vol. 3: Distributed Data Management Architecture*

This document is one of three Technical Standards documenting the Distributed Relational Database Architecture (DRDA). This volume describes the architected commands, parameters, objects, and messages of the DDM data stream. This data stream accomplishes the data interchange between the various pieces of the DDM model. This document is available on the Open Group Web site at https://collaboration.opengroup.org/dbiop/?gpid=453&gpid=453&gpid=453.

IBM Redbooks

This topic describes the IBM Redbooks that are available for distributed relational database.

- DRDA Client/Server for VM and VSE Setup for System and Performance Management (about 3,792 KB)
- WOW! DRDA Supports TCP/IP: DB2 Server for OS/390 and DB2 Universal Database (about 3,519 KB)

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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 or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country 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 improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Programming interface information

This Distributed database programming publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Oracle, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.