Db2 11.1 for Linux, UNIX, and Windows

IBM

CLPPlus Reference

Db2 11.1 for Linux, UNIX, and Windows

CLPPlus Reference

Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment.

Contents

Figures

Tables

Command line processor plus (CLPPlus)

Command line processor plus (CLPPlus) provides a command-line user interface that you can use to connect to databases and to define, edit, and run statements, scripts, and commands.

CLPPlus complements the functions that the command line processor (CLP) provides. CLPPlus includes the following features:

- Support for establishing connections to databases when you provide a database user ID and password.
- A buffer that you can use to store scripts, script fragments, SQL statements, SQL PL statements, or PL/SQL statements for editing and then execution. You can list, print, or edit the text in the buffer or run the text in the buffer as a batch script.
- A comprehensive set of processor commands that you can use to define variables and strings that you can store in the buffer.
- A set of commands that retrieve information about a database and database objects.
- The ability to store buffers or buffer output in a file.
- Multiple options for formatting the output of scripts and queries.
- Support for executing system-defined routines.
- Support for executing operating system commands.
- An option for recording the output of executed commands, statements, or scripts.

CLPPlus supports SERVER, SERVER_ENCRYPT, KERBEROS and GSSPLUGINauthentication only.

Starting a CLPPlus client session

The CLPPlus session must be started before any CLPPlus commands can be run.

Before you begin

Read the **CLPPLUS** command topic.

Procedure

To start the CLPPlus session:

- Issue the **CLPPLUS** command:
  1. In the command line processor (CLP) or theDb2® Command Window, run the **CLPPLUS** command. You can start the CLPPlus session and establish a connection to the database with a single command. For example, you can use the following CLPPlus command to start the CLPPlus session and connect to a local SAMPLE database with user ID db2admin. In the following example, the database is listening on the port number 50000.

     ```
     clpplus db2admin@localhost:50000/sample
     ```
  2. Enter your password. The following example shows prompting for the password that is associated with the user ID db2admin:

```
C:\DB2\10.5\db2 > clpplus db2admin@localhost:50000/sample
CLPPlus: Version 1.6
Copyright (c) 2009, 2011, IBM CORPORATION.  All rights reserved.
Enter password: ********

Database Connection Information :
---------------------------------
Hostname = localhost
Database server = DB2/NT64  SQL10012
SQL authorization ID = db2admin
Local database alias = SAMPLE
Port = 50000

SQL>
```

- On Windows operating systems, you can use the menu option:
  1. Click **Start** > **IBM Db2** > **CLPPlus**.
  2. Specify your user ID and connection information.

  The menu option is not available for the following IBM® data server client installations:
  – IBM Data Server Client
  – IBM Data Server Runtime Client
  – IBM Data Server Driver Package
- On Linux operating systems, you can use the menu option:
  1. Click **Main Menu** > **IBM Db2** > **Db2 Command Line Processor Plus**.
  2. Specify your user ID and connection information.

  The menu option is not available for the following IBM data server client installations:
  – IBM Data Server Client
  – IBM Data Server Runtime Client
  – IBM Data Server Driver Package

Results

The CLPPlus prompt (`SQL>`) is available, and you are connected to the specified database.

What to do next

You can now use CLPPlus commands and related features. Specify the commands at the CLPPlus prompt.

To end the CLPPLus session, issue the **`EXIT`** or **`QUIT`** CLPPlus command.

CLPPLUS command

Starts the command line processor plus (CLPPlus) session. After you start the CLPPlus session, you can issue CLPPlus commands, connect to databases, define, and run SQL statements and database commands, and run scripts that contain SQL statements and commands.

Invocation

You must run the **`CLPPLUS`** command from the operating system command prompt.

Authorization

None

Required connection

None

Command Syntax

```
►►─clpplus─┬──────────┬─┬─────┬─┬─────────┬─┬──────────────────────┬──────────►
           └─-verbose─┘ └─-nw─┘ ├─-s──────┤ ├─connection_identifier─┤
                                ├─-si─────┤ └─/─────────────────────┘
                                ├─-sil────┤
                                ├─-sile───┤
                                ├─-silen──┤
                                └─-silent─┘

►─┬─────────────┬─┬───────────────┬────────────────────────────────────────►◄
  └─@─dsn_alias─┘ └─@─script_name─┘
```

connection_identifier:

```
├──user─┬─────────────┬─┬─────────┬─┬─────────┬─┬─────────────┬──────────────┤
        └─/─password──┘ └─@─host──┘ └─:─port──┘ └─/─database──┘
```

Command parameters

-verbose
Sets verbose on. When verbose is set on all CLPPlus messages are printed to the console.

-nw
Specifies that the CLPPlus session is started in the current command-line window.

-s | -si | -sil | -sile | -silen | -silent
Suppresses the version information, copyright information, prompt messages, command echo, and connection information from being printed in the current CLPPlus session. During silent mode, by default, ECHO is set to OFF. Any attempt to set ECHO to ON is ignored.

/ Specifies the current operating system login user ID is used to connect to the database.

user
Specifies the user ID to connect to the database.

password
Specifies the password that corresponds to the user ID.

hostname
Specifies the name of the computer on which the database is located. For example, for a computer that is named ascender, specify `@ascender`.

port
Specifies the port number that receives connections on the computer where the database server is installed. The default is 50000.

database
: Specifies the database name to which the connection is made. The default is SAMPLE.

dsn_alias
: Specifies that the database connection information is picked up from the IBM data server driver configuration file (`db2dsdriver.cfg`) from the dsn with alias name *dsn_alias*. If the specified *dsn_alias* is not found in the IBM data server driver configuration file, the string *dsn_alias* is used as a database name and all other connection parameters are obtained interactively.

script-name
: Specifies the name of a script file. If the file is not in the current working directory, you must also include the fully qualified path to the location of the file. The script file can contain SQL statements that are automatically run after you start the CLPPlus session and establish a database connection.

Example

The following examples show how you can use the **CLPPLUS** command to start the CLPPlus session and optionally connect to databases.

The following command starts the CLPPlus session in client mode:

```
clpplus
```

No database connection is attempted. You can issue the **CONNECT** CLPPlus command to connect to a database.

The following command starts the CLPPlus session and attempts to connect to a database named SAMPLE on a local host with user ID adminuser and port number 50000:

```
clpplus adminuser@localhost:50000/sample
```

The following command starts the CLPPlus session and prompts for a password for the user ID adminuser. If the password is valid, the CLPPlus interface attempts to connect to a database named SAMPLE, which is the default database name.

```
clpplus adminuser
```

The following command starts the CLPPlus session and attempts to connect to a database named SAMPLE with user ID adminuser and password mypassw0rd. If the user ID and password are valid, a database connection is established. The drawback to specifying a password is that the password is displayed on the screen.

```
clpplus adminuser/mypassw0rd
```

The following command starts the CLPPlus session and attempts to connect to a database named SAMPLE on the remote machine ascender using port 50000, user ID adminuser, and password mypassw0rd. If these values are valid, a database connection is established.

```
clpplus adminuser/mypassw0rd@ascender:50000/sample
```

The following command starts the CLPPlus session and attempts to connect to a database named DB on the local computer with user ID adminuser, password mypassw0rd, and the default port number, which is 50000:

```
clpplus adminuser/mypassw0rd@localhost/db
```

The following command starts the CLPPlus session and attempts to connect to a database by first locating an IBM data server driver configuration file. If one is found, the default_dsn is read for the host, port, and database values. The current logon ID is used in the connection attempt. If no IBM data server driver configuration file is found, all required parameters are requested interactively.

```
clpplus /
```

The following command starts the CLPPlus session and attempts to connect to a database by first locating an IBM data server driver configuration file. If one is found, the default_dsn is read for the host, port, and database values. The adminuser ID is used in the connection attempt. If no IBM data server driver configuration file is found, all required parameters are requested interactively.

```
clpplus adminuser
```

The following command starts the CLPPlus session and attempts to connect to a database by fetching the parameters from the data_dsn alias in the IBM data server driver configuration file. Since no user ID is specified, the current logon user ID is used. Any parameters that cannot be read are requested interactively.

```
clpplus /@data_dsn
```

The following command starts the CLPPlus session and attempts to connect to a database by fetching the parameters from the data_dsn alias in the IBM data server driver configuration file. The adminuser user ID is used in the connection. Any parameters that cannot be read are requested interactively.

```
clpplus adminuser@data_dsn
```

Session Global Variables in CLPPlus

CLPPlus has support for setting the value of the Session Global Variable from the `db2dsdriver.cfg` file.

A global variable is a named memory variable that is retrieved or modified through SQL statements. Global variables enable applications to share relational data among SQL statements without the need for extra application logic to support this data transfer.

Authorization
No special authorization is required.

Declaration
When the global variables are created by using the CREATE VARIABLE and these variables exist at the server, users can set their value in the *sessionglobalvariables* section in `db2dsdriver.cfg` file.

Scope The value of a session global variable is uniquely associated with each session that uses this particular global variable.

Restrictions
None.

Sample `db2dsdiver.cfg` file

```
<configuration>
   <dsncollection>
      <dsn alias="sample" name="sample"  host="localhost" port="50000"/>
         <sessionglobalvariables>
            <parameter name="EMPID" value="12345"/>
                <parameter name="SQL_COMPAT" value="NPS"/>
         </sessionglobalvariables>
      </dsn>
```

```
          </dsncollection>
          <databases>
             <database name="sample" host="localhost" port="50000">
               <sessionglobalvariables>
                       <parameter name="SQL_COMPAT" value="NPS"/>
               </sessionglobalvariables>
             </database>
             </databases>
          <parameters>
             <sessionglobalvariables>
                <parameter name="EMPID" value="100"/>
             </sessionglobalvariables>
          </parameters>
       </configuration>
```

CLPPlus console types

CLPPlus has two console modes that are called window and non-windowed mode

Window mode

When CLPPlus is started with the **CLPPLUS** command, the window mode console is spawned by default. This new console window has better command editing capabilities. As such, when using CLPPlus in an interactive mode, this window mode might be the preferred console choice.

Non-window mode

When CLPPlus is started with the **CLPPLUS** command and the `-nw` option is specified, the current command line environment is used, that is, no new window is spawned. This might be a preferred choice if you are calling a script file and do not require a spawned console window.

Window mode CLPPlus console and UTF-8 character support

The window mode CLPPlus console supports UTF-8 characters.

The window mode CLPPlus console can read UTF-8 characters from the file system, database, and interactively from the keyboard. The window mode CLPPlus console can write UTF-8 characters into the file system, database or interactively to the standard output. Also, column alignment issues that can be present with the non-window mode CLPPlus console are resolved with the window mode CLPPlus console.

UTF-8 character support is available only in the window mode CLPPlus console, which is the default CLPPlus console type. For the non-window mode CLPPlus console, support for non-ASCII characters is limited. In the non-window mode CLPPlus console, you might not be able to enter certain non-ASCII characters. Misalignment of result set columns can occur if non-ASCII characters are included in that result set.

Setting the font in CLPPlus window mode

You can set different fonts in the CLPPlus window mode. Settings include font name, style, and size.

Procedure

To set the font name, style, and size in the CLPPlus window mode:

1. Using the **CLPPLUS** CLPPlus command, start the CLPPlus session. The CLPPlus session is started, by default, in window mode. For more information, see the Related reference.
2. Use the **SET** CLPPlus command with the FONT keyword, **SET FONT name, style, size**. For more information, see the Related reference.

Example

The following example sets the font that is displayed in the CLPPlus window mode to serif, in bold typeset and size of 32 points.

```
SET FONT serif,1,32
```

The following example sets the font that is displayed in the CLPPlus window mode to SansSerif, in plain typeset and size of 48 points.

```
SET FONT "SansSerif", 0, 48
```

The following example sets the font that is displayed in the CLPPlus window mode to Lucida console, in bold typeset and size of 35 points.

```
SET FONT "lucida console",1, 35
```

Setting the font color in CLPPlus window mode

You can set different font colors in the CLPPlus window mode.

Procedure

To set the font color in the CLPPlus window mode:

1. Using the **CLPPLUS** CLPPlus command, start the CLPPlus session. The CLPPlus session is started, by default, in window mode. For more information, see the Related reference.
2. Use the **SET** CLPPlus command with the COLOR keyword, **SET COLOR color|<r,g,b>**. For more information on the **SET** command, see the Related reference.

Example

The following example sets the font color that is displayed in the CLPPlus window mode to red.

```
SET COLOR RED
```

The following example sets the font color that is displayed in the CLPPlus window mode to white.

```
SET COLOR 255,255,255
```

Setting the background color in CLPPlus window mode

You can set different background colors in the CLPPlus window mode.

Procedure

To set the background color in the CLPPlus window mode:

1. Using the **CLPPLUS** CLPPlus command, start the CLPPlus session. The CLPPlus session is started, by default, in the window mode. For more information, see the Related reference.

2. Use the **SET** CLPPlus command with the BGCOLOR keyword, **SET BGCOLOR color|<r,g,b>**. For more information on the **SET** command, see the Related reference.

Example

The following example sets the background color that is displayed in the CLPPlus window mode to cyan.

```
SET BGCOLOR cyan
```

The following example sets the background color that is displayed in the CLPPlus window mode to white.

```
SET BGCOLOR 255,255,255
```

DSN aliases in CLPPlus

The CLPPlus interface supports connecting to DSN aliases defined in the IBM data server driver configuration file (`db2dsdriver.cfg`). Before this support, only interactive connects were allowed in the CLPPlus interface.

The IBM data server driver configuration file is an XML file that contains a list of DSN aliases and their properties. It is used to store connection details in one place. The CLPPlus interface can use that information to automatically connect to a data source instead of interactively asking for all the connection details on every connect attempt.

You can set the *DB2DSDRIVER_CFG_PATH* environment variable to point to the IBM data server driver configuration file. For more information about the *DB2DSDRIVER_CFG_PATH* environment variable, see the Miscellaneous variables topic listed in the Related reference.

If *DB2DSDRIVER_CFG_PATH* is not set, the CLPPlus interface searches for the IBM data server driver configuration file in the default directory location. For information about the default location of the IBM data server driver configuration file, see the Related concepts.

If a configuration file is found and it is readable, the CLPPlus interface uses it on the subsequent connect attempts.

A user who attempts a connect is asked for a database name. That database name is treated as a DSN alias and searched in the configuration file. If that DSN alias is found, the connection attributes are read and a password is requested to complete the connection. If the DSN alias is not found, then the host name, port number, user name, and password are requested interactively to go with the original database name, and all information gathered is used to attempt a connection.

You can specify whether the ExtendedIndicators property is enabled or disabled during a database connection. By default, this property is disabled. For more information, see the Related reference.

Example

Consider the following IBM data server driver configuration file contents:

```
<configuration>
 <dsncollection>
  <dsn alias="S" name="SAMPLE" host="9.121.221.159" port="50000">
```

```
      <parameter name="ExtendedIndicators" value="0"/>
  </dsn>
 </dsncollection>
 <databases>
  <database name="SAMPLE" host="9.121.221.159" port="50000">
   <parameter name="UserID" value="john"/>
  </database>
 </databases>
</configuration>
```

The following example shows a connection established with the contents of the IBM data server driver configuration file.

First the user sets the *DB2DSDRIVER_CFG_PATH* environment variable.

```
C:\>set DB2DSDRIVER_CFG_PATH="C:\john\clpplus"
```

In the following example, the **clpplus** command is used to start a CLPPlus session and the **connect** command is issued to establish connection with DSN alias "S".

```
C:\>clpplus
CLPPlus: Version 1.1
Copyright (c) 2009, IBM CORPORATION.  All rights reserved.

SQL> connect

Enter DATABASE NAME [SAMPLE]: S
Enter ID [john] :
Enter Password: ********

Database Connection Information
-------------------------------
Hostname = 9.121.221.159
Database server = DB2/NT  SQL09071
SQL authorization ID = john
Local database alias = S
Port = 50000SQL>
```

The following example shows a connection when the database name entered is not found as an alias in the IBM data server driver configuration file.

```
SQL> connect

Enter DATABASE NAME [SAMPLE]: sample
Enter HOSTNAME [localhost]:
Enter PORT [50000]:
Enter ID : john
Enter Password: ********

Database Connection Information
-------------------------------
Hostname = 9.121.221.159
Database server = DB2/NT  SQL09071
SQL authorization ID = john
Local database alias = SAMPLE
Port = 50000SQL>
```

Since "sample" is not found as a DSN alias in the configuration file, the remaining values are requested interactively by the CLPPlus interface and then a connection attempt is made.

Supported IBM data server driver configuration keywords

The CLPPlus interface supports a subset of the keywords you can use in the IBM data server driver configuration file (`db2dsdriver.cfg`).

The following IBM data server driver configuration keywords are supported in a CLPPlus environment.

Authentication (database)
: The **Authentication** keyword specifies the authentication mechanism that is used when you connect to a database. The CLPPlus interface supports the `Kerberos`, `SERVER_ENCRYPT`, `SERVER_ENCRYPT_AES`, and `SERVER` authentication mechanisms.

Authentication (LDAP)
: The **Authentication** keyword specifies the authentication mechanism that is used during the LDAP server connection. Supported authentication mechanisms vary based on the LDAP service provider. Check your LDAP server documentation as different LDAP servers support different authentication mechanisms. Examples of authentication mechanisms are `"none"`, `"simple"`, and `sasl_mech`. If you specify an unsupported authentication mechanism, the attempted connection fails.

  The Authentication keyword must be defined in the `ldapserver` section of the IBM data server driver configuration file.

BaseDN
: The **BaseDN** keyword defines the base distinguished name for the LDAP server.

EnableLDAP
: The **EnableLDAP** keyword defines whether LDAP support is enabled. If the EnableLDAP keyword is set to `No`, the CLPPlus interface does not read the server details from the `ldapserver` section of the IBM data server driver configuration file.

ExtendedIndicators
: The **ExtendedIndicators** keyword specifies whether the **ExtendedIndicators** property is enabled or disabled during a database connection.

LDAPServerHost
: The **LDAPServerHost** keyword defines the IP address or host name of the LDAP server.

LDAPServerPort
: The **LDAPServerPort** keyword defines the LDAP Server port number.

Password
: If the Password keyword is defined in the `dsncollection` section or `databases` section of the IBM data server driver configuration file, the specified Password keyword value is used as the default password when you connect to the database under which the UserID keyword is specified.

  If the Password keyword is defined in the `ldapserver` section of the IBM data server driver configuration file, the specified Password keyword value is used to connect to the LDAP server.

SecurityTransportMode
: The **SecurityTransportMode** keyword sets the communication security type. You can set the **SecurityTransportMode** keyword to `SSL` for establishing SSL connections. The SSL CLPPlus connections are implemented by the JDBC driver that uses the Java™ SSL APIs and it does not use the IBM Global Security Kit (GSKit) libraries.

  Starting in Fix Pack 7, the CLPPlus interface supports the **SecurityTransportMode** keyword. If you specify the **SecurityTransportMode** keyword in earlier fix packs or specify a value other than `SSL`, the **SecurityTransportMode** keyword is silently ignored.

SSLServerCertificate

The **SSLServerCertificate** keyword specifies the fully qualified name of a self-signed server certificate. You can set the **SSLServerCertificate** keyword when the **SecurityTransportMode** keyword is set to SSL and the self-signed certificate from a server is used.

Starting in Fix Pack 6, the CLPPlus interface supports the **SSLServerCertificate** keyword. If you specify the **SSLServerCertificate** keyword in earlier fix packs or specify an invalid value, the **SSLServerCertificate** keyword is silently ignored.

UserID

If the UserID keyword is defined in the `dsncollection` section or `databases` section of the IBM data server driver configuration file, the specified UserID keyword value is used as the default user ID when you connect to the database under which the UserID keyword is specified.

If the UserID keyword is defined in the `ldapserver` section of the IBM data server driver configuration file, the specified UserID keyword value is used to connect to the LDAP server.

For more information about supported keywords, see the Related reference.

The SSL protocol support in CLPPlus

Starting in Fix Pack 7, you can establish SSL connections in the CLPPlus interface with use of the **SecurityTransportMode** keyword.

The IBM data server driver configuration file is an XML file that contains a list of DSN aliases, database directory information, and their properties. If the DSN alias or database entry contains the **SecurityTransportMode** parameter value that is set to SSL, the SSL protocol is used in CLPPlus connections to the server. If the **SecurityTransportMode** parameter value is set to SSL and the SSL connection requires the use of self-signed certificate on the server, you can copy the certificate to the client and set the **SSLServerCertificate** parameter to the absolute path and name of the certificate. For more information, see the Supported IBM data server driver configuration keyword topic.

The SSL CLPPlus connections are implemented by the JDBC driver that uses the Java SSL APIs and it does not use the IBM Global Security Kit (GSKit) libraries.

Example

Consider the following IBM data server driver configuration file contents:

```
<configuration>
  <dsncollection>
    <dsn alias="S" name="SAMPLE" host="9.121.221.159" port="50001">
    </dsn>
  </dsncollection>
  <databases>
    <database name="SAMPLE" host="9.121.221.159" port="50001">
      <parameter name="SecurityTransportMode" value="SSL"/>
    </database>
  </databases>
</configuration>
```

The following example shows a connection being established with the contents of the IBM data server driver configuration file, which includes the **SecurityTransportMode** parameter value.

The user starts a CLPPlus session and attempts a connection to the DSN alias "S".

```
C:\>clpplus
CLPPlus: Version 1.1
Copyright (c) 2009, IBM CORPORATION.  All rights reserved.

SQL> connect

Enter DATABASE NAME [SAMPLE]: S
Enter ID [john] :
Enter Password: ********

Database Connection Information
-------------------------------
Hostname = 9.121.221.159
Database server = DB2/XXXXXXXXX  SQL10055
SQL authorization ID = john
Local database alias = S
Port = 50001SQL>
```

Kerberos authentication in CLPPlus

The CLPPlus interface supports connecting to DSN aliases using Kerberos authentication as defined in the IBM data server driver configuration file (`db2dsdriver.cfg`).

The IBM data server driver configuration file is an XML file that contains a list of DSN aliases and their properties. If the DSN alias entry contains the **`Authentication`** property value that is set to *kerberos*, the Kerberos authentication mechanism is used. For more information, see the DSN aliases in CLPPlus topic.

The CLPPlus interface does not request a Kerberos TGT ticket on its own. It uses the ticket that is already obtained by the user for use with other applications or tools.

Example

Consider the following IBM data server driver configuration file contents:

```
<configuration>
 <dsncollection>
  <dsn alias="S" name="SAMPLE" host="9.121.221.159" port="50000">
  </dsn>
 </dsncollection>
 <databases>
  <database name="SAMPLE" host="9.121.221.159" port="50000">
   <parameter name="UserID" value="john"/>
  </database>
 </databases>
 <parameters>
  <parameter name="Authentication" value="KERBEROS"/>
 </parameters>
</configuration>
```

The following example shows a connection being established with the contents of the IBM data server driver configuration file, which includes the **`Authentication`** parameter value.

The user starts a CLPPlus session and attempts a connection to the DSN alias "S".

```
C:\>clpplus
CLPPlus: Version 1.1
Copyright (c) 2009, IBM CORPORATION.  All rights reserved.
```

```
SQL> connect

Enter DATABASE NAME [SAMPLE]: S
Enter ID [john] :
Enter Password: ********

Database Connection Information
-------------------------------
Hostname = 9.121.221.159
Database server = DB2/NT  SQL09071
SQL authorization ID = john
Local database alias = S
Port = 50000SQL>
```

SERVER_ENCRYPT authentication in CLPPlus

The CLPPlus interface supports connections to DSN aliases with `SERVER_ENCRYPT` authentication as defined in the IBM data server driver configuration file (`db2dsdriver.cfg`).

The IBM data server driver configuration file is an XML file that contains a list of DSN aliases and their properties. If the DSN alias entry contains the **`Authentication`** property value that is set to `SERVER_ENCRYPT`, the SERVER_ENCRYPT authentication mechanism is used. For more information, see the DSN aliases in CLPPlus topic.

Example

The following example shows a connection being established with the contents of the IBM data server driver configuration file, which includes the **`Authentication`** parameter value.

Consider the following IBM data server driver configuration file contents:

```
<configuration>
 <dsncollection>
  <dsn alias="S" name="SAMPLE" host="9.121.221.159" port="50000">
       <parameter name="Authentication" value="SERVER_ENCRYPT"/>
  </dsn>
 </dsncollection>
 <databases>
  <database name="SAMPLE" host="9.121.221.159" port="50000">
   <parameter name="UserID" value="john"/>
  </database>
 </databases>
</configuration>
```

The user starts a CLPPlus session and attempts a connection to the DSN alias "S".

```
C:\>clpplus -nw
CLPPlus: Version 1.5
Copyright (c) 2009, 2011, IBM CORPORATION.  All rights reserved.

SQL> connect
Enter DATABASE NAME [SAMPLE]: S
Enter ID [john]:
Enter password: **********

Database Connection Information :
---------------------------------
Hostname = 9.121.221.159
```

```
Database server = DB2/NT  SQL09075
SQL authorization ID = john
Local database alias = S
Port = 50000
```

SERVER_ENCRYPT_AES authentication in CLPPlus

In Db2 Cancun Release 10.5.0.4, the CLPPlus interface adds support for connections to DSN aliases that use `SERVER_ENCRYPT_AES` authentication. The `SERVER_ENCRYPT_AES` authentication option is defined in the IBM data server driver configuration file (`db2dsdriver.cfg`).

The IBM data server driver configuration file is an XML file that contains a list of DSN aliases and their properties. If the DSN alias entry contains the **`Authentication`** property value that is set to `SERVER_ENCRYPT_AES`, the `SERVER_ENCRYPT_AES` authentication mechanism is used. For more information, see the DSN aliases in CLPPlus topic.

Example

The following example shows a connection being established with the contents of the IBM data server driver configuration file, which includes the **`Authentication`** parameter value.

Consider the following IBM data server driver configuration file contents:

```
<configuration>
 <dsncollection>
  <dsn alias="S" name="SAMPLE" host="9.121.221.159" port="50000">
       <parameter name="Authentication" value="SERVER_ENCRYPT_AES"/>
  </dsn>
 </dsncollection>
 <databases>
  <database name="SAMPLE" host="9.121.221.159" port="50000">
   <parameter name="UserID" value="john"/>
  </database>
 </databases>
</configuration>
```

The user starts a CLPPlus session and attempts a connection to the DSN alias "S".

```
C:\>clpplus -nw
CLPPlus: Version 1.6
Copyright (c) 2009, 2011, IBM CORPORATION.  All rights reserved.

SQL> connect
Enter DATABASE NAME [SAMPLE]: S
Enter ID [john]:
Enter password: **********

Database Connection Information :
---------------------------------
Hostname = 9.121.221.159
Database server = DB2/NT  SQL09075
SQL authorization ID = john
Local database alias = S
Port = 50000
```

LDAP support in CLPPlus

CLPPlus connections support DSN alias searches in a configured LDAP directory server.

Description

If you specify a DSN alias name that is not found in the IBM data server driver configuration file (`db2dsdriver.cfg`), the CLPPlus interface attempts to connect to the LDAP directory server that is specified in the IBM data server driver configuration file to resolve the DSN alias name. The CLPPlus interface looks up the DSN alias name on the LDAP directory server and use the required connection details from the DSN entry, such as host name, port number, user ID, and password, to make a connection. If no match is found on the LDAP directory server or the connection to the LDAP directory server fails, the DSN alias name is treated as a database name during an interactive connection.

To enable the LDAP support, use the **`<ldapserver>`** section in the IBM data server driver configuration file to specify the LDAP directory server information. A single LDAP directory server entry is allowed in the IBM data server driver configuration file. The **`UserID`** and **`Password`** fields in the **`<ldapserver>`** section are optional; you can enter user ID and password information at run time. User ID and password information is cached during the CLPPlus session.

If you set the **`UserID`** parameter in the IBM data server driver configuration file to `"*anonymous"`, an anonymous connection to the LDAP directory server is attempted; user ID and password information is not passed. You are not prompted for a password, and if you set the **`Password`** parameter in the IBM data server driver configuration file, the parameter is ignored.

Important:
If you set the **`UserID`** parameter to `"*anonymous"`, you must configure the LDAP directory server to support anonymous connections.

Examples

Consider the following sample IBM data server driver configuration file:

```
<configuration>
 <dsncollection>
  <dsn alias="alias1" name="name1" host="server1.net1.com" port="50001"/>
 </dsncollection>

 <databases>

  <database name="name1" host="server1.net1.com" port="50001">
   <parameter name="CurrentSchema" value="OWNER1"/>
   <wlb>
    <parameter name="enableWLB" value="true"/>
    <parameter name="maxTransports" value="50"/>
   </wlb>
   <acr>
    <parameter name="enableACR" value="true"/>
   </acr>
  </database>

 </databases>

 <ldapserver>
  <parameter name="EnableLDAP" value="YES"/>
  <parameter name="LDAPServerHost" value="ipv6lab7.torolab.ibm.com"/>
  <parameter name="LDAPServerPort" value="389"/>
  <parameter name="UserID" value="root"/>
  <parameter name="Password" value="itdsv23"/>
  <parameter name="BaseDN" value="O=IBM"/>
  <parameter name="Authentication" value="simple"/>
 </ldapserver>
</configuration>
```

The following example connects to a DSN alias DBLDAP1 with the sample IBM data server driver configuration file. The DBLDAP1 DSN alias name is not found in the IBM data server driver configuration file and the DSN alias entry on the LDAP directory server ipv6lab7.torolab.ibm is searched. The host, port, and database information from the DBLDAP1 DSN alias name that is found on the LDAP directory server is retrieved to establish a connection.

```
SQL> connect
Enter DATABASE NAME [SAMPLE]: DBLDAP1
Enter ID : db2admin
Enter password: ********

Database Connection Information :
---------------------------------
Hostname = winguest.torolab.ibm.com
Database server = DB2/NT  SQL09075
SQL authorization ID = db2admin
Local database alias = DBLDAP1
Port = 50000
```

The following example connects to the DBLDAP1 DSN from the CLPPlus session in the **VERBOSE** mode:

```
SQL> connect
DB250001I: CLPPlus has successfully read the configuration file named
'C:\Documents and Settings\All Users\Application data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

Enter DATABASE NAME [SAMPLE]: DBLDAP1

DB250014I: DSN alias 'DBLDAP1' is not found in the configuration file named
'C:\Documents and Settings\All Users\Application Data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

DB250015I: CLPPlus successfully established a connection with LDAP directory
server 'ipv6lab7.torolab.ibm.com:389'

Enter ID : db2admin
Enter password: ********

Database Connection Information :
---------------------------------
Hostname = winguest.torolab.ibm.com
Database server = DB2/NT  SQL09075
SQL authorization ID = db2admin
Local database alias = DBLDAP1
Port = 50000
```

The following example connects to a DSN alias DBLDAP2 with the sample IBM data server driver configuration file. The CLPPlus session is running in the **VERBOSE** mode. When the DSN alias DBLDAP2 is not found in the IBM data server driver configuration file or on the specified LDAP directory server, then the interactive CLPPlus connection attempt occurs.

```
SQL> connect
DB250001I: CLPPlus has successfully read the configuration file named
'C:\Documents and Settings\All Users\Application data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

Enter DATABASE NAME [SAMPLE]: DBLDAP2

DB250014I: DSN alias 'DBLDAP2' is not found in the configuration file named
'C:\Documents and Settings\All Users\Application Data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

DB250015I: CLPPlus successfully established a connection with LDAP directory server 'ipv6lab7.torolab.ibm.com:389'

DB250016E: DSN alias 'DBLDAP2' was not found in LDAP directory server 'ipv6lab7.torolab.ibm.com:389'.
'DBLDAP2' is used as the database name in the subsequent interactive CLPPlus connect attempt.
```

```
Enter HOSTNAME [localhost]: 9.128.34.89
Enter PORT [50000]: 50003
Enter ID: db2admin
Enter password:*******

Database Connection Information :
---------------------------------
Hostname = 9.128.34.89
Database server = DB2/NT  SQL09075
SQL authorization ID = db2admin
Local database alias = DBLDAP2
Port = 50003
```

The following example connects to the DBLDAP2 DSN from the CLPPlus session in the **VERBOSE** mode:

```
SQL> connect
DB250001I: CLPPlus has successfully read the configuration file named
'C:\Documents and Settings\All Users\Application data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

Enter DATABASE NAME [SAMPLE]: DBLDAP2

DB250014I: DSN alias 'DBLDAP2' is not found in the configuration file named
'C:\Documents and Settings\All Users\Application Data\IBM\DB2\DB2COPY1\cfg\db2dsdriver.cfg'.

DB250017E: CLPPlus failed to establish a connection with LDAP
directory server 'ipv6lab7.torolab.ibm.com:389'. 'DBLDAP2' is
used as the database name in an interactive connect attempt.

Enter HOSTNAME [localhost]: 9.128.34.89
Enter PORT [50000]: 50003
Enter ID: db2admin
Enter password:*******

Database Connection Information :
---------------------------------
Hostname = 9.128.34.89
Database server = DB2/NT  SQL09075
SQL authorization ID = db2admin
Local database alias = DBLDAP2
Port = 50003
```

Consider the following modified version of the IBM data server driver configuration file. The IBM data server driver configuration file does not include the **UserID** and **Password** parameters for the LDAP directory server configuration. The LDAP directory server ipv6lab7.torolab.ibm is specified.

```
<configuration>
 <dsncollection>
  <dsn alias="alias1" name="name1" host="server1.net1.com" port="50001"/>
 </dsncollection>

 <databases>

  <database name="name1" host="server1.net1.com" port="50001">
   <parameter name="CurrentSchema" value="OWNER1"/>
   <wlb>
    <parameter name="enableWLB" value="true"/>
    <parameter name="maxTransports" value="50"/>
   </wlb>
   <acr>
    <parameter name="enableACR" value="true"/>
   </acr>
  </database>

 </databases>

 <ldapserver>
  <parameter name="EnableLDAP" value="YES"/>
  <parameter name="LDAPServerHost" value="ipv6lab7.torolab.ibm.com"/>
  <parameter name="LDAPServerPort" value="389"/>
  <parameter name="BaseDN" value="O=IBM"/>
  <parameter name="Authentication" value="simple"/>
 </ldapserver>
</configuration>
```

Using the updated IBM data server driver configuration file, a connection to alias name `SAMPLE32` is attempted. The alias name is not found in the IBM data server driver configuration file. When the user ID and password to the LDAP directory server are entered interactively, as shown in the following example, the CLPPlus interface connects to the `ipv6lab7.torolab.ibm` LDAP directory server. The LDAP directory server is successfully searched for the `SAMPLE32` DSN alias, and the host, port, and database information is retrieved. A CLPPlus connection is established with the database information that is retrieved from the LDAP server. In the following example, the CLPPlus session is not running in the **VERBOSE** mode.

```
C:\Documents and Settings>clpplus /@SAMPLE32

CLPPlus: Version 1.4
Copyright (c) 2009, 2011, IBM CORPORATION.  All rights reserved.

Connecting to LDAP server '9.234.67.89:389'.
Enter LDAP server user ID: root
Enter LDAP server password: ********

Enter password: **********

Database Connection Information :
---------------------------------
Hostname = 9.128.32.149
Database server = DB2/NT  SQL09075
SQL authorization ID = db2admin
Local database alias = SAMPLE32
Port = 50002
```

Running a script file in the CLPPlus session

In the CLPPlus session, a script file can be run in many ways. You can provide the name of a script file that contains database commands and SQL commands as a token for the **CLPPLUS** command. You can run a script file using the **START** CLPPlus command. You can also run a script by copying its contents into the CLPPlus SQL buffer using the **GET** CLPPlus command and then issuing the **RUN** CLPPlus command.

About this task

You can run a script with the **CLPPLUS** command. For other methods, see the related links.

Procedure

To run a script with the **CLPPLUS** command:

Run the **CLPPLUS** command, specifying a script name. For example, consider the following script file named `dept_query.sql`.

```
SET PAGESIZE 9999
SET ECHO ON
SELECT * FROM DEPT;
EXIT
```

To run the `dept_query.sql` script on the default SAMPLE database on port 50000 with a user name of **db2user** and password **passw0rd**, issue the following command:

```
clpplus db2user/passw0rd @dept_query
```

The `dept_query.sql` script file is run after the user connects to the database. When the script is run, the commands **SET PAGESIZE** and **SET ECHO ON** and the statement SELECT * FROM are issued.
The output of the script is as follows. **ECHO ON** displays the statement that was issued in the script file, and the values of DEPT are displayed up to a page limit of 9999.

```
clpplus db2user/passw0rd @dept_query.sql
 Connected to XXX v X.X (localhost:5444/db2sampl) AS db2user

SQL>
SELECT * FROM dept;

DEPT      NODNAME     LOC
------    ----------  -----------
   10     ACCOUNTING  NEW YORK
   20     RESEARCH    DALLAS
   30     SALES       CHICAGO
    4     OPERATIONS  BOSTON

SQL >

EXIT
```

Skipping and interrupting CLPPlus commands

CLPPlus allows you to skip and interrupt command execution as well as script execution.

You can interrupt any command or script that CLPPlus is running using the Ctrl+C keystroke. This is helpful when you encounter a long running query or script and need to return control back to the CLPPlus interface.

You can also skip to the next `SQL>` prompt by pressing the ENTER key twice in succession. This is helpful when you enter an incorrect command and want to cancel it. The incorrect command is retained in the buffer and can be edited using any of the CLPPlus commands you use for editing and review.

Example

The following example shows a command being skipped.

```
SQL> select *
  2  from employee              <- first carriage return
  3                             <- second carriage return
SQL>                            <- next sql prompt
```

Comments in CLPPlus

CLPPlus has the ability for you to include comments in your scripts and commands.

In CLPPlus, comments can span one ore more lines. Comments that are contained on a single line start with # or `--`. Comments that span multiple lines are enclosed in `/*` and `*/`.

Example

The following examples show both single and multiple line comments.

```
SQL> # This is a single line comment
SQL>
SQL> -- This is also a single line comment
SQL>
SQL> /* This comment
spans
multiple lines. */
SQL>
```

Escape characters in CLPPlus

You can use escape characters in CLPPlus commands and queries.

Description

In CLPPlus, you can use the ampersand (&) character to substitute variables in SQL statements. Escape characters can be used to escape the ampersand character in input values to avoid substitution, for example "AT&M".

Escape characters can also be used to escape the characters "$" and "%", which are used to reference shell and environment variables in CLPPlus.

You can define the escape character with the **SET** command. The default escape character is "\". For more information about the **SET** command, see the related reference.

Example

1. This example shows the use of the default escape character which avoids "&M" being treated as a substitution variable.

   ```
   SQL> set escape ON
   SQL> insert into testtab values('AT\&M');
   DB250000I: The command completed successfully.

   SQL> select * from testtab;
   TEXT
   -------------------
   AT&M
   ```

2. This example shows the use of a user-defined escape character which avoids "&G" being treated as a substitution variable.

   ```
   SQL> set escape ^
   SQL> insert into testtab values('L^&G');
   DB250000I: The command completed successfully.

   SQL> select * from testtab;
   TEXT
   -------------------
   AT&M
   L&G
   ```

3. This example shows the behavior when no escape character is used. "&V" is treated as a substitution variable and requires the user to provide the input value of "Planet".

   ```
   SQL> set escape OFF
   SQL> insert into testtab values('Smarter &V');
   Enter a value for variable V: Planet
   ```

```
Original statement: insert into testtab values('Smarter &V')
New statement with substitutions: insert into testtab values('Smarter Planet')
DB250000I: The command completed successfully.

SQL> select * from testtab;
TEXT
-------------------
AT&M
L&G
Smarter Planet
```

4. This example shows the behavior when no escape character is used. "&V" is treated as a substitution variable and requires the user to provide the input value of "Gene".

```
SQL> set escape OFF
SQL> insert into testtab values('Blue \&V');
Enter a value for variable V: Gene

Original statement: insert into testtab values('Blue \&V')
New statement with substitutions: insert into testtab values('Blue \Gene')
DB250000I: The command completed successfully.

SQL> select * from testtab;
TEXT
-------------------
AT&M
L&G
Smarter Planet
Blue \Gene
```

5. This example shows the behavior when an escape character is used. "$100" is treated as a value and not a shell or environment variable.

```
SQL> set escape ON
SQL> insert into testsub values('\$100');
DB250000I: The command completed successfully.

SQL> select * from testsub;
TEXT
-------------------
$100
```

6. This example shows the behavior when an escape character is used. "86%" is treated as a value and not a shell or environment variable.

```
SQL> set escape ON
SQL> insert into testsub values('86\%');
DB250000I: The command completed successfully.

SQL> select * from testsub;
TEXT
-------------------
$100
86%
```

Bind variables in CLPPlus

Bind variables are used in place of literal values. If you issue SQL statements multiple times, you can use bind variables to reduce the number of literal values.

Authorization

No special authorization is required.

Declaration

A bind variable can be declared using the following syntax:

```
►►—VARIABLE—name—datatype;——————————————————————————————————————►◄
```

name
Specifies the name of the bind variable.

`datatype`
Specifies the data type that is associated with the bind variable. The data type can be one of: BOOLEAN, CHARACTER, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, REAL, SMALLINT, or VARCHAR.

REFCURSOR is also supported. REFCURSOR is used to receive the **`OUT`** parameter values of type **`CURSOR`** in procedures, functions, and anonymous PL/SQL blocks.

NUMBER, NUMBER(p[,s]), and VARCHAR2 are also supported. NUMBER and NUMBER(p[,s]) are implicitly mapped to the DECIMAL data type. VARCHAR2 is implicitly mapped to the VARCHAR data type.

CLPPlus allows the use of BOOLEAN, ROW, and ARRAY data types as parameters for stored procedures with Db2 servers. You can run a stored procedure with the **`CALL`** or **`EXEC`** CLPPlus statements.

Scope

Bind variables persist over the duration of a user's CLPPlus session. When a CLPPlus session is started, bind variables can be declared and used during that session. When a CLPPlus session is ended, any bind variables are cleared.

Restrictions

When used in an SQL statement or an anonymous PL/SQL block, a bind variable can appear only once. If the bind variable is used more than once, an error from the database server is returned.

Db2 for z/OS® and Informix® Dynamic Server data servers have the following limitations with the usage of bind variables:

- Bind variables cannot be initialized using the **`EXEC`** command.

```
Exec :var_name:='john' /* this is not supported */
```

- Bind variables cannot be initialized using a begin-end block.

```
begin
:var_name:='john'; /* this is not supported in a begin-end block */
end;
```

- Since PL/SQL is not supported on Db2 for z/OS and Informix Dynamic Server data servers, bind variables are not supported in a PL/SQL body.
- Variables with type CURSOR are not supported.

```
SQL> CREATE PROCEDURE getEmployeeData( ID INT, OUT NAME char(10),
         OUT DOB Date, OUT SAL DECIMAL(7,2))
         LET NAME='dummy';
         LET DOB='10/10/2010';
         LET SAL=0;
    SELECT empname, empdob, salary INTO name, dob, sal FROM emp   WHERE empid = ID;
    END PROCEDURE;
/
DB250000I: The command completed successfully.
```

```
SQL> define var_id=1001  /* usage of substitution variable */
SQL> Variable name varchar(10)
DB250000I: The command completed successfully.
SQL> Variable dob date
DB250000I: The command completed successfully.
SQL> Variable salary double
DB250000I: The command completed successfully.

Call getEmployeeData(&var_id,  :name, :dob, :salary)
DB250000I: The command completed successfully.
SQL> Print name
'JOHN'
SQL> Print dob
'26/04/1982'
SQL> Print salary
10000.50
```

- Precision and scale values can be specified while creating bind variables of with the NUMBER and DECIMAL data types. There is a limitation in precision support. Any decimal or number values that are assigned are not modified to the precision specified in the definition of the variable. See example 13 for more details.

These restrictions apply to the **EXECUTE** CLPPlus command as well.

Examples

The following examples show how you can define, initialize, and use bind variables.

1. Bind variables that are named **ID** and **LNAME** of type **VARCHAR**:
   ```
    VARIABLE ID VARCHAR
    VARIABLE LNAME VARCHAR
   ```
2. A bind variable that is named **ID** initialized in a PL/SQL block:
   ```
   BEGIN
      :ID := '000020';
   END;
   /
   ```
3. Bind variables **ID** and **LNAME** used in a PL/SQL block:
   ```
   BEGIN
      SELECT lastname INTO :LNAME FROM employee
    WHERE empno = :ID;
   END;
   /
   ```
4. A single PL/SQL statement initializes a bind variable named **ID** :
   ```
   EXECUTE :ID := '000022';
   ```
5. The variable **ID** is initialized from a substitution variable *a* (*a* is defined with the **DEFINE** CLPPlus command):
   ```
   EXECUTE :ID := &a;
   ```
6. The **ID** bind variable is used in a SELECT statement:
   ```
   SELECT lastname FROM employee WHERE empno = :ID;
   ```
7. The **ID** and **LNAME** bind variables are used in an UPDATE statement:
   ```
   UPDATE employee SET lastname = :LNAME WHERE empno = :ID;
   ```
8. The **salary** bind variable is defined with the `number` data type:
   ```
   variable salary number
   exec :salary := 1000.00
   ```
9. The **bonus** bind variable is defined with the `number(p[,s])` data type:
   ```
   variable bonus number(6)
   exec :bonus:= 999.999
   ```

10. The **comm** bind variable is defined with the `number(p[,s])` data type:
    ```
    variable bonus comm(4,2)
    exec :comm:= 10.455

    SQL> print comm
    10.45
    ```
11. The **name** bind variable is defined with the `varchar2` data type:
    ```
    variable name varchar2
    exec :name:='MICHAEL'
    ```
12. This example shows the substitution of bind variables as input and output arguments in procedure execution. Assume a file example_proc.db2 contains the following statement:
    ```
       CREATE PROCEDURE DEPT_MEDIAN
       (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
    ```
    Bind variables are substituted as input and output arguments. Define the bind variables:
    ```
       variable in_var integer
       variable out_var double
    ```
    Run the procedure and substitute the variables as parameters:
    ```
       call dept_median(:in_var, :out_var)
    ```
    Optionally print the contents of the output argument, which is the **out_var** bind variable:
    ```
       print out_var
    ```
13. This example shows a bind variable **var1**, which does not reflect the precision in the definition:
    ```
       variable var1 number(4,2)
       DB250000I: The command completed successfully.
    ```
    Assign a value that has precision of 6 digits and scale of 3 digits:
    ```
       exec :var1 := 333.333
       /
    DB250000I: The command completed successfully.
    ```
    Print the contents of **var1**:
    ```
       print var1
       333.33
    ```
    The scale is correct, 2. The precision is not 4 as defined. This scenario is a current limitation in functionality.

Shell and environment variables in CLPPlus

You can use shell and environment variables in CLPPlus commands and queries.

Description

With this support, any shell or environment variable value can be used in a CLPPlus command or query. This use is identical to how these variables are accessed in shell or command windows.

Shell or environment variable substitution can be enabled or disabled with the **SET ENVVARSUBST** CLPPlus command. For details on this command, see the Related reference.

Any variable prefixed with "$" is treated as a shell or environment variable. This character provides operating system independence to scripts which are used across differing operating systems. Environment variables on Microsoft platforms can also be accessed by wrapping "%" around the variable name, for example, %VAR%.

CLPPlus understands and resolves a shell or environment variable if it includes the characters a-z, A-Z, 0-9, and "_". If any other character is encountered, then it is treated as the end of the variable name. The following examples highlight this naming rule:

```
$clpplus&version    <==== "clpplus" followed by "&"
$clpplus[version]   <==== "clpplus" followed by "["
$clpplus version    <==== "clpplus" followed by " "(space)
$clpplus#version    <==== "clpplus" followed by "#"
$clpplus-version    <==== "clpplus" followed by "-"
$clpplus(version)   <==== "clpplus" followed by "("
```

If CLPPlus attempts to use a variable and it is defined, then its value is retrieved and used as expected. If CLPPlus attempts to use a variable and it is not defined, an empty string value is substituted and an informational message is logged. The informational message is visible when CLPPlus is operating in verbose mode.

You can access shell and environment variables in CLPPlus with the following methods:

- While starting CLPPlus, you can pass shell or environment variables as arguments to a script. These passed variables are converted into position parameters which are accessed with the "&" character and their position (number)
- Shell or environment variables can be directly accessed in CLPPlus with the "$" character followed by the variable name. This method is irrespective of the operating system on which the script is run.

Examples

1. The following example shows how you can use shell or environment variables while starting CLPPlus:

   On UNIX and Linux platforms:

   ```
   Export TABLE_NAME=employee
   Export SCRIPT_PATH=/home/user

   clpplus -nw @$SCRIPT_PATH/script.sql  $TABLE_NAME
   ```

   On Windows platforms:

   ```
   Set  TABLE_NAME=employee
   Set  SCRIPT_PATH=c:\testfiles

   clpplus -nw @%SCRIPT_PATH%\script.sql  %TABLE_NAME%
   ```

   where as script.sql contains

   ```
   Select * from &1
   ```

   and *&1* resolves to the *TABLE_NAME* variable value.

2. The following example shows how you can use shell or environment variables directly in CLPPlus:

On UNIX and Linux platforms:

```
select * from $TABLE_NAME;
insert into test values ($NAME, $ID, $DESG);
```

On Windows platforms:

```
select * from %TABLE_NAME%;
insert into test values (%NAME%, %ID%, %DESG%);
```

Db2 commands supported by CLPPlus

The CLPPlus interface supports a subset of Db2 commands for database and database manager administration, tuning, and maintenance.

The following Db2 commands are supported in the CLPPlus interface:

- **GET DATABASE CONFIGURATION**
- **GET DATABASE MANAGER CONFIGURATION**
- **UPDATE DATABASE CONFIGURATION**
- **UPDATE DATABASE MANAGER CONFIGURATION**
- **RESET DATABASE CONFIGURATION**
- **RESET DATABASE MANAGER CONFIGURATION**
- **LIST PACKAGES**
- **IMPORT**
- **EXPORT**
- **LOAD**
- **REORG**, supported when connected to Db2 and Db2 for z/OS.
- **RUNSTATS**, supported when connected to Db2 and Db2 for z/OS.
- **REORGCHK**, for more information see the related reference.
- Limited support for **CREATE DATABASE**, also supported when connected to IBM Informix. For more information about restrictions see the related reference.
- **DROP DATABASE**, also supported when connected to IBM Informix.

Note: For Db2, the **CREATE DATABASE** and **DROP DATABASE** commands fail if connected to a remote database manager instance. You must be connected to a local database manager instance

Note: IMPORT, EXPORT and LOAD commands have a restriction that processed files must be on the server

CREATE DATABASE in CLPPlus

The CLPPlus interface provides limited support for the **CREATE DATABASE** command. The **CREATE DATABASE** command can be used to create database in the Db2 and IBM Informix environments.

Restrictions

For the Db2 environment, the **CREATE DATABASE** command fails if you are connected to a remote database manager instance. You must be connected to a local database manager instance. You can specify the following parameters for the **CREATE DATABASE** command in the CLPPlus interface:

- The required *database-name* variable
- The optional **CODESET** parameter
- The optional **TERRITORY** parameter

- The optional **PAGESIZE** parameter

  Note: When you create a Db2 database without the **pagesize** parameter, the default page size for the new database is 32 K.

For IBM Informix, **CREATE DATABASE** from the CLPPlus interface requires connection credentials, specifically to the **sysmaster** database. You are prompted for connection credentials if a connection does not exist. For more information about the **CREATE DATABASE** command in IBM Informix, see http://www.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.sqls.doc/ids_sqs_0368.htm

Examples

The following command creates a database that is named `testdb`:

```
create database testdb;
```

The following command creates a database that is named `testdb` with a page size of 4 K in the Db2 environment:

```
create database testdb pagesize 4K;
```

The following command creates a database that is named `testdb` in the Db2 environment. The code page set is defined as `UTF-8`, and the territory is defined as `US`.

```
create db testdb using codeset UTF-8 territory US;
```

The following command creates a database that is named `udttest` in the IBM Informix environment. No previous connection exists, so you are prompted to provide the connection information.

```
SQL> create database udttest;

Enter DATABASE NAME [sysmaster]:
Enter HOSTNAME [localhost]: 9.130.34.100
Enter PORT [50000]: 9089
Enter ID:  informix
Enter password: **********

DB250000I: The command completed successfully.
```

In the following example, the first command connects to an IBM Informix database. The second command creates a database that is named `udttest`.

```
SQL> connect Informix/informix123@9.130.34.100:9089/stores

Database Connection Information :
---------------------------------
Hostname = 9.130.34.100
Database server = IDS/NT32  IFX11700
SQL authorization ID = informix
Local database alias = stores
Port = 9089

SQL> create database udttest with log mode ansi ;
DB250000I: The command completed successfully.
```

CLPPlus restrictions

The CLPPlus interface has certain connection, command, and statement restrictions.

The CLPPlus interface can establish database connections with the following Db2 database product:

- IBM Db2 for IBM i
- Db2 Express-C

The CLPPlus interface can establish database connections with the following Db2 database products but with the following restrictions:

- Db2 Version 9.8 Fix Pack 1 or higher. You must apply Fix Pack 1 to V9.8 before connectivity is supported.

The CLPPlus interface has the following restrictions on PL/SQL support:

- PL/SQL functions and triggers cannot be created in a partitioned database environment.
- The NCLOB data type is not supported for use in PL/SQL statements or in PL/SQL contexts when the database is not defined as a Unicode database. In Unicode databases, the NCLOB data type is mapped to a Db2DBCLOB data type.
- The XMLTYPE data type is not supported.
- TYPE declaration is not supported in a function, procedure, trigger, or anonymous block.
- The FOR EACH STATEMENT option is not supported for PL/SQL triggers.

Table 1. CLPPlus limitations across different data servers.

CLPPlus Feature	Db2	Db2 for z/OS	IBM Informix
Variable REFCURSOR	Yes	No	No
SERVEROUTPUT	Yes	No	No
EXECUTE	Yes	No	No
LIST PACKAGES	Yes	Yes	No
SHOW ERRORS	Yes	No	No
UPDATE/GET/ RESET DB CFG	Yes	No	No
UPDATE/GET/ RESET DBM CFG	Yes	No	No
EXPORT	Yes	No	No
IMPORT	Yes	No	No
LOAD	Yes	No	No

The **EDIT** command is supported in the CLPPlus window mode. The command is not supported in the CLPPlus non-window mode.

CLPPlus troubleshooting hints and tips

List of general CLPPlus startup issues and solutions.

Table 2. Starting CLPPlus: Issues and solutions

Issue	Explanation and solution
The following message is displayed when you try to start the CLPPlus session: `CLPPlus requires Java 1.5 or higher to execute. Please ensure Java is in your PATH.` The CLPPlus session fails to start.	Ensure that you have Java 1.5 or later installed. The IBM database server products andIBM Data Server Client product install a required Java product during the Db2 installation process. However, other IBM data server client products do not install a Java product. When a required Java product is not installed as part of the Db2 product installation, the CLPPlus interface looks for a Java product in the path that is specified by the **JAVA_HOME** and **PATH** environment variables. If you installed an IBM data server client product other than the IBM Data Server Client product, download and install a Java JRE or SDK, Version 1.5 or later. Set the **JAVA_HOME** environment variable to point to the Java installation directory. Add the Java `bin` directory to the **PATH** environment variable setting.
The following message is displayed when you try to start the CLPPlus session: `Could not find db2jcc.jar. Please ensure that your installation completed successfully. If the problem persists, please locate and add db2jcc.jar to your CLASSPATH.` The CLPPlus session fails to start.	The CLPPlus interface requires the Java universal drivers in the `db2jcc.jar` file. When the CLPPlus interface cannot find the `db2jcc.jar` file in the **CLASSPATH** environment variable setting or in the *installation directory*/`java` directory, startup fails. Ensure that the installation was completed successfully. Add the absolute path of the `db2jcc.jar` file to the **CLASSPATH** environment variable setting.
The following message is displayed when you try to start the CLPPlus session: `Could not find clpplus.jar. Please ensure that your installation completed successfully. If the problem persists, please locate and add clpplus.jar to your CLASSPATH.` The CLPPlus session fails to start.	The CLPPlus interface requires the `clpplus.jar` file that is included with the product. When the CLPPlus interface cannot find the `clpplus.jar` file in the **CLASSPATH** environment variable setting or in the installation directory, startup fails. Ensure that the installation was completed successfully. Add the absolute path of the `db2jcc.jar` file to the **CLASSPATH** environment variable setting.

Table 2. Starting CLPPlus: Issues and solutions (continued)

Issue	Explanation and solution
The following message is displayed when you try to start the CLPPlus session: `Could not find jline-0.9.93.jar. Please ensure that your installation completed successfully. If the problem persists, please locate and add jline-0.9.93.jar to your CLASSPATH.` The CLPPlus session fails to start.	The CLPPlus interface requires the `jline-0.9.93.jar` file that is included with the product. When the CLPPlus interface cannot find the `jline-0.9.93.jar` file in the **`CLASSPATH`** environment variable setting or in the installation directory, the CLPPlus session fails to start. Ensure that the installation was completed successfully. Add the absolute path of the `jline-0.9.93.jar` file to the **`CLASSPATH`** environment variable setting.

CLPPlus traces and record logging

CLPPlus provides mechanisms for file traces and record logging. CLPPlus supports logging or traces from the CLPPlus client layer and JDBC driver layer.

The IBM Data Server Driver for JDBC and SQLJ and IBM Data Server Driver for ODBC and CLI offer comprehensive tracing facilities. These facilities have been extended to CLPPlus. Trace facilities generate text log files whenever an application accesses a specified driver (CLPPLus Client layer or JDBC Driver layer) using the SET command. These log files provide detailed information about the CLPPlus Client and JDBC:

- functions called by an application
- function contents; including input and output parameters passed to and received from
- function return codes and any error or warning messages generated.

To configure the CLPPlus trace facilities, issue the **`SET`** command from a CLPPlus command prompt. To enable client layer or driver layer traces, set the **`LOGMODE`** parameter:

```
CLPPlus> SET LOGMODE logmode-value
```

where *logmode-value* indicates whether to perform tracing and for which layer. The default value is NONE, which indicates no tracing is done. Other valid values are CLPPLUS, which traces the client layer, JCC, which traces the JDBC layer, and BOTH, which traces both the client and JDBC layers.

To perform more detailed JDBC tracing, set *logmode-value* to `JCC` or `BOTH`, and specify the **`JCCLOGMODE`** parameter:

```
SET LOGMODE JCC
SET JCCLOGMODE jcclogmode_value
```

where *jcclogmode_value* indicates the features to be traced and logged. For more information about valid *jcclogmode_value* settings, see "SET" on page 89.

CLPPlus commands

The CLPPlus feature includes many commands which provide extensive user control, customization, and personalization.

Note: Unless otherwise specified, CLPPlus command names and parameters are not case-sensitive; you can specify uppercase or lowercase letters.

.

The **.** CLPPlus command is similar to a No Operation Performed (NOOP or NOP) machine language command. It is ignored when entered on its own with no other CLPPlus command.

The **.** CLPPlus command can also be used to skip the currently entered command and move to the next SQL> prompt. This can be done by entering the command on its own on a new line in the current block of code. This can help you cancel a command when you enter an incorrect entry. A cancelled command is available in the history and can be accessed and edited.

Invocation

You must run this command from the CLPPlus interface or from within a CLPPlus script file.

Authorization

None

Required connection

None

Command syntax

```
▸▸─.──────────────────────────────────────────────────────────▸◂
```

Example

In the following example, the **.** command is used to cancel the current command.

```
SQL> begin
       2 dbms_output.putline('wrong put_line');
       3 .
SQL>
```

!

The **!** CLPPlus command is a synonym to the **HOST** CLPPlus command. It will run an operating system command.

Invocation

You must run this command in the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
▶▶─!─os_command──────────────────────────────────────────────────────▶◀
```

Command parameters

os_command
: Specifies an operating system command.

Example

In the following example, the **!** command is issued with the **dir** operating system command.

```
SQL> ! dir

Volume in drive C has no label.
Volume Serial Number is 6806-ABBD

Directory of C:\USER_Backup\Office\Project\FP3\FP3\src

06/05/2010  22:18  <DIR>    .
06/05/2010  22:18  <DIR>    ..
06/05/2010  22:35           405.classpath
06/05/2010  17:20  <DIR>    com
               1 File(s)           798 bytes
               3 Dir(s)  33,397,190,656 bytes free
```

/

The / CLPPlus command reruns the last command run in the current CLPPlus session.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
▶▶─/─────────────────────────────────────────────────────────────────▶◀
```

Example

The following example shows the / command rerunning the last **SELECT** statement run within CLPPlus:

```
    SQL> SELECT AVG(salary) FROM employee WHERE workdept = 'E21'
                                              1
    ------------------------------------------
           47086.66666666666666666666666666667
    SQL> /
```

```
                                       1
----------------------------------------
      47086.6666666666666666666666666667
```

@

The **@** CLPPlus command is an alias for the **START** CLPPlus command. It can be used to run a CLPPlus script file.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──@──┬──────┬──script-file─────────────────────────────────────────────►◄
       ├─path─┤
       └─URL──┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the script file that contains SQL statements and commands to run. If no path is specified, the current directory is used.

URL
: Specifies the URL to the script file that contains SQL statements and commands to run. The URL must start with `http://` or `https://`.

script-file
: Specifies the script file name that contains SQL statements and commands to run.

Example

A script called employee_count.sql contains the following lines:

```
ACCEPT dept_id PROMPT "Enter Department ID code : "
SELECT COUNT(*) FROM employee WHERE workdept = &dept_id;
```

The **@** command can be used to run the script as follows:

```
SQL> @ employee_count
Enter Department ID code : 'E21'

Original statement:SELECT AVG(salary) FROM employee WHERE workdept = &dept_id
New statement with substitutions:SELECT AVG(salary) FROM employee WHERE workdept = 'E21'
                                       1
----------------------------------------
      47086.6666666666666666666666666667
```

@@

The **@@** CLPPlus command is an alias for the **START** CLPPlus command. It can be used only from within a CLPPlus script file to call and run another CLPPlus script file.

Invocation

You must run this command from within a CLPPlus script.

Authorization

None

Required connection

None

Command syntax

```
►►──@@──┬────────┬──script-file──────────────────────────────────────────►◄
        ├─path───┤
        └─URL────┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the script file that contains SQL statements and commands to run. If no path is specified, the current directory is used.

URL
: Specifies the URL to the script file that contains SQL statements and commands to run. The URL must start with `http://` or `https://`.

script-file
: Specifies the script file name that contains SQL statements and commands to run.

Example

A script called dept_details.sql calls another script called employee_count.sql. The contents of dept_details.sql follows:

```
ACCEPT dept_id PROMPT "Enter Department ID code : "
@@ employee_count &dept_id
```

ACCEPT

The **ACCEPT** CLPPlus command creates a variable with a specified name. Values can be assigned to this variable interactively in the CLPPlus interface or through a parameter read as part of a script that is run. The **ACCEPT** command is useful for storing values that you commonly use in SQL statements or in the SQL buffer.

Output of the command is by default displayed to the standard output of the CLPPlus interface.

Invocation

You must run this command from the CLPPlus interface or from within a CLPPlus script file.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─ACCEPT─┬──variable-name──┬──────────────────────┬──────────────────────►
    └─ACC────┘                 └─FORMAT──format-string─┘

►──┬───────────────────────────┬──┬─────────────┬──┬──────┬──────────────────►◄
   └─DEFAULT──default-value────┘  └─PROMPT──text─┘  └─HIDE─┘
```

Command parameters

variable-name
: Defines the variable name. You cannot use special symbols and characters such as the forward slash (/) or at sign (@).

  When you issue the **ACCEPT** command, you are prompted for the value of *variable-name*.

FORMAT ***format-string***
: Defines the format assigned to the variable. The value you attempt to assign to the variable must follow the format outlined.

  For a character variable, the value of *format-string* is A*n*, where *n* specifies the number of characters that can be used to display the variable. The data wraps if it is wider than the specified width.

  For numeric variables, the value of *format-string* can be one or more of the following characters:

  $ Displays a leading dollar sign.

  , Displays a comma at the indicated position.

  . Displays a decimal point at the indicated position.

  0 Displays a zero at the indicated position.

  9 Displays a significant digit at the indicated position.

  If loss of significant digits occurs due to overflow of the format settings, the # character is displayed.

DEFAULT ***default-value***
: The default value defined with this option is assigned to the variable when a user hits the ENTER key and does not provide any value when prompted.

PROMPT ***text***
: The value defined with this option is displayed in the prompt when the **ACCEPT** command is entered.

HIDE
When **HIDE** is specified, the value entered by the user is not echoed on the console.

Example

In the following example, the **ACCEPT** command creates a variable named `my_name` and prompts for a value. The value `John Smith` is stored in this variable. The **DEFINE** command displays the value of the variable.

```
SQL> ACCEPT my_name
Enter value for my_name: John Smith
SQL> DEFINE my_name
DEFINE my_name = "John Smith"
```

The following example shows all the options used in the **ACCEPT** command.

```
SQL> ACCEPT lname FORMAT A10 DEFAULT 'Joy' PROMPT 'Enter Last Name [Joy]:' HIDE
```

The **FORMAT** option specifies that the value for `lname` is alphanumeric and 10 characters in length. The **DEFAULT** used if the user does not provide a value when prompted and instead hits the ENTER key is `JOY`. The prompt at the command line when the **ACCEPT** command is issued is as defined: `Enter Last Name [JOY]:`. In this case the default is included as part of the prompt. The **HIDE** option does not echo what the user enters as the value for `lname` on the console.

The following example shows the **ACCEPT** command being used in a CLPPlus script file and the different methods in which a value can be assigned to the defined variable. Consider the following script named `average_salary.sql` which finds the average salary of an employee in the given department:

```
ACCEPT dept_id PROMPT "Enter Department ID code : "
SELECT AVG(salary) FROM employee WHERE workdept = &dept_id;
```

The script can be called in two different ways, with and without arguments.

When called with arguments, the variable is assigned the value of the argument passed at the time of invocation:

```
SQL> start average_salary 'E21'

Original statement:SELECT AVG(salary) FROM employee WHERE workdept = &dept_id
New statement with substitutions:SELECT AVG(salary) FROM employee WHERE workdept = 'E21'
                                          1
------------------------------------------
       47086.66666666666666666666666666667
```

When called without arguments, the user interaction is required to assign a value to the variable:

```
SQL> start average_salary
Enter Department ID code : 'E21'

Original statement:SELECT AVG(salary) FROM employee WHERE workdept = &dept_id
New statement with substitutions:SELECT AVG(salary) FROM employee WHERE workdept = 'E21'
                                           1
 ------------------------------------------
        47086.66666666666666666666666666667
```

APPEND

The **APPEND** CLPPLus command adds text to the end of the current line in the SQL buffer. You can use this command to build commands and statements in the SQL buffer.

Invocation

This is a line editor command and can be used to build commands in the SQL buffer.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─APPEND─┬──text-string──────────────────────────────────────────►◄
    └─A──────┘
```

Command parameters

text-string
: Specifies a string of characters to append. The string can include spaces and special characters. The case of the string is preserved.

Examples

In the following example, the **APPEND** command appends the string `this text is appended.` to the end of the current line in the SQL buffer:

```
APPEND this text is appended.
```

The following example shows how you can use the **APPEND** command to build a SELECT statement in the SQL buffer. Two spaces are placed between the **APPEND** command and the WHERE clause to separate `DEPT` and `WHERE` by one space in the SQL buffer.

```
SQL> APPEND SELECT * FROM DEPT
SQL> LIST
  1* SELECT * FROM DEPT
SQL> APPEND  WHERE DEPTNO = 10
SQL> LIST
  1* SELECT * FROM DEPT WHERE DEPTNO = 10
```

The **LIST** command displays the contents of the SQL buffer as the SQL statement is being built.

BREAK

The **BREAK** CLPPlus command inserts a page break or blank lines at the specified point in a result set.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──BREAK──ON──column-name──┬─────────────────────┬──────────────────►◄
                            └─SKIP────────────────┘
                              ├─PAGE───────────┤
                              └─number-of-lines─┘
```

Command parameters

column-name
: Specifies the column used to determine a break.

SKIP PAGE | *number-of-lines*
: Where *number-of-lines* is an integer.

When **SKIP PAGE** is appended to the command the output breaks and continues on the next page. When **SKIP** *number-of-lines* is appended to the command, the output breaks and blanks lines equal to the *number-of-lines* specified are inserted in the result set.

Example

In the following example, when the SELECT statement is invoked and the value of WORKDEPT changes from one row to another, the **BREAK** command is invoked and the action specified is performed. In this case, since **SKIP PAGE** was specified, the next row will be printed on the next page skipping the remainder of the current page.

```
SQL> BREAK ON WORKDEPT SKIP PAGE;
SQL> SELECT * FROM EMPLOYEE ORDER BY WORKDEPT;
```

In the following example, in addition to the behavior of the preceding example, every time the value in the JOB column changes, 2 blank lines are printed on the display.

```
SQL> BREAK ON WORKDEPT SKIP PAGE;
SQL> BREAK ON JOB SKIP 2;
SQL> SELECT * FROM EMPLOYEE ORDER BY WORKDEPT, JOB;
```

BTITLE

The **BTITLE** CLPPlus command inserts text at the bottom of each page displayed.

Invocation

You must run this command from the CLPPlus interface.

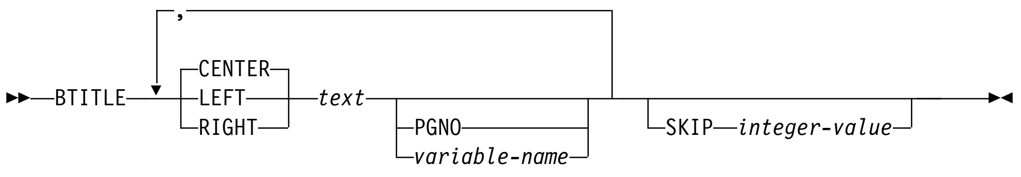
Authorization

None

Required connection

None

Command syntax



Command parameters

text
: Specifies the text to be displayed.

CENTER
: Specifies the display will center justify the text on each page. If neither **CENTER**, **LEFT**, or **RIGHT** is specified, center justification is the default behavior.

LEFT
: Specifies the display will left justify the text on each page.

RIGHT
: Specifies the display will right justify the text on each page.

PGNO
: Specifies the current page number.

variable-name
: Specifies a user defined variable that will follow the *text* field.

SKIP *integer-value*
: The *integer-value* value specifies the number of blank lines displayed before the bottom title.

Example

In the following example, the `DEPT:` (with the variable contents), `CONFIDENTIAL`, and `Page No:` (with the current page number) is displayed across the bottom of every page. Three blank lines follows the bottom title.

```
SQL> BREAK ON workdept SKIP PAGE;
SQL> COLUMN workdept OLD_VALUE old_dept;
SQL> BTITLE LEFT 'DEPT: ' old_dept, CENTER 'CONFIDENTIAL, RIGHT 'Page No: ' PGNO SKIP 3;
```

In the following example, the `Page No:` title (with the current page number) is displayed across the bottom of every page with right justification. Two blank lines follow the bottom title.

```
SQL> BTITLE RIGHT 'Page No: ' PGNO SKIP 2;
```

CALL

The **CALL** CLPPlus command calls a stored procedure.

Invocation

You can use the **CALL** command to call a stored procedure with array parameters when the stored procedure is on a Db2 server.

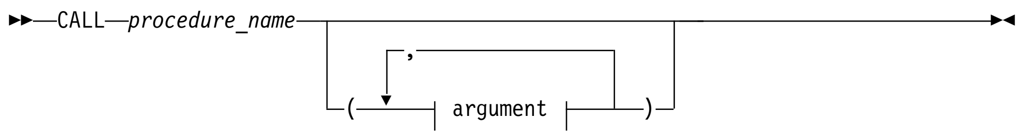
Authorization

None

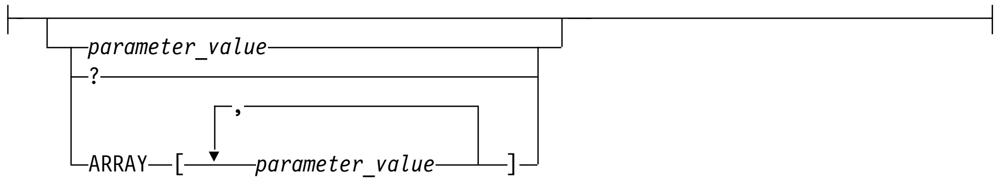
Required connection

None

Command syntax



argument:



Command parameters

procedure_name
: Specifies the procedure to call. If multiple stored procedures with the same name are present, the specific procedure to invoke is chosen by using procedure resolution. Procedure resolution involves identifying the target procedure by its schema, the procedure name, and the number of parameters.

parameter_value
: Specifies the argument value. Enclose all text values in single quotation marks.

? Specifies the **OUT** parameter placeholder.

ARRAY[*parameter_value1*, *parameter_value2*, ...]
: Specifies the array parameter values. Enclose all text values in single quotation marks.

Examples

In the following example, the SUM stored procedure with **IN** and **OUT** parameters is created:

```
CREATE PROCEDURE SUM
          (IN int1 integer, IN int2 integer, OUT int3 integer)
           BEGIN
            SET int3 = int1+int2;
           END;
           /
```

The SUM stored procedure is invoked by using the **CALL** command:

```
call sum(5,10,?);
/
```

The sample **CALL** command returns the following output:

```
Value of output parameters
--------------------------------
INT3 = 15

DB250000I: The command completed successfully.
```

In the following example, the names user type and the find_student stored procedure with array parameters are created:

```
CREATE TYPE names AS VARCHAR(20) ARRAY[50];

CREATE PROCEDURE find_student(IN students_in names,IN alphabet VARCHAR(1), OUT students_out names)
          BEGIN
            DECLARE i,j,max INTEGER;
        SET i = 1;
        SET j = 1;
        SET students_out = NULL;
        SET max = CARDINALITY(students_in);
        WHILE i <= max DO
        if substr(students_in[i], 1, 1) = alphabet THEN
            SET students_out[j] = students_in[i];
            SET j = j+1;
        END IF;
            SET i = i+1;
         END WHILE;
           END;
          /
```

The find_student stored procedure is invoked to find the list of students whose names start with the character A:

```
CALL find_student(ARRAY['Alice','Bob','Derk','Peter','Alan','Clark'],'A',?);
/
```

The sample **CALL** command returns the following output:

```
Value of output parameters
--------------------------------
STUDENTS_OUT : ARRAY

Values
---------------
'Alice'
'Alan'

DB250000I: The command completed successfully.
```

CHANGE

The **CHANGE** CLPPlus command modifies specified content in the SQL buffer. If you set the buffer reader to a specific line in the buffer, the command modifies only the specified content in that line.

The **CHANGE** or **C** token is optional for the complete command when specifying which line in the buffer you are changing. You can issue the command by specifying only the line number in the buffer you want to change along with the new text.

Invocation

This is a line-editor command used to modify fields in the SQL buffer.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─CHANGE─┬──┬──────────────────────┬──┬──────────────────────┬──┬─────┬─────────────►◄
    ├─C──────┤  └─/──search-string──/─┘  └─replacement-string─┘  └─/─┘
    └─n──────┘
```

Command parameters

n Specifies the line number in the buffer that will be changed. The *n* command only takes the *replacement-string* variable.

search-string
Defines the text in the SQL buffer to be replaced or deleted. If the buffer contains more than one line of text, specify the line to be modified by entering the line number at the prompt before you run the **CHANGE** command.

If the text to search for contains an asterisk (*), enclose the asterisk in single quotation marks.

replacement-string
Specifies the replacement text or specifies that text is to be removed. If you specify a value for *replacement-string*, the first occurrence of the value of *search-string* is replaced with the value of *replacement-string*. If you do not specify a value for *replacement-string*, the first occurrence of the value of *search-string* is removed.

If the replacement text contains an asterisk (*), you do not need to enclose an asterisk (*) in single quotation marks.

Examples

In the following example, the **LIST** command displays the contents of the buffer. At the SQL prompt, 3 is entered to move the buffer reader to the start of the third line in the buffer. The third line becomes the new current line, as indicated by an asterisk. The **CHANGE** command then replaces the occurrence of the string 20 with the string 30. The LIST command then displays the modified text within the buffer.

```
SQL> LIST
  1  SELECT EMPNO, ENAME, JOB, SAL, COMM
  2  FROM EMP
  3  WHERE DEPTNO = 20
  4* ORDER by EMPNO
SQL> 3
  3* WHERE deptno = 20
SQL> CHANGE /20/30/
  3* WHERE DEPTNO = 30
SQL> LIST
  1  SELECT EMPNO, ENAME, JOB, SAL, COMM
  2  FROM EMP
  3* WHERE DEPTNO = 30
  4  ORDER by EMPNO
```

In the following example, the buffer contains the following single statement:

```
SQL> SELECT EMPNO FROM EMPLOYEE
```

To change the statement so that EMPNO is replaced with *, 1 is entered to move the buffer reader to the start of the first line in the buffer. The following **CHANGE** command is issued:

```
SQL> CHANGE /empno/'*'/
```

The output of the command is as follows:

```
  1* SELECT * FROM EMPLOYEE
```

The command output displays the line number followed by the new content for that line.

You can use the **CHANGE** command to specify the line number in the buffer you want to change, and what value you want to change it to.

```
SQL> SELECT *
  2  FROM
  3  EMPLOKEE ;
ERROR near line 1:
SQL0204N  "SCHEMA.EMPLOKEE" is an undefined name.

SQL> LIST
 1  SELECT *
 2  FROM
 3* EMPLOKEE

SQL> 3 EMPLOYEE
 3* EMPLOYEE

SQL> LIST
 1  SELECT *
 2  FROM
 3* EMPLOYEE

SQL> /
```

CLEAR

The **CLEAR** CLPPlus command removes the contents of the SQL buffer, deletes all column definitions set by the **COLUMN** command, or clears the screen.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
                 ┌─SCREEN──────────────────┐
                 ├─SCR─────────────────────┤
►►──┬─CLEAR─┬────┼─────────────────────────┼──────────────────────────────►◄
    └─CL────┘    ├─BREAKS──┬─────────────┬─┤
                 │         └─ON──column──┘ │
                 ├─BRE──┬─────────────┬────┤
                 │      └─ON──column──┘    │
                 ├─BUFFER──────────────────┤
                 ├─BUF─────────────────────┤
                 ├─SQL─────────────────────┤
                 ├─COLUMNS─────────────────┤
                 ├─COL─────────────────────┤
                 ├─COMPUTES─┬─────────────┬┤
                 │          └─ON──column──┘│
                 ├─COMP──┬─────────────┬───┤
                 │       └─ON──column──┘   │
                 ├─VARIABLES───────────────┤
                 └─VAR─────────────────────┘
```

Command parameters

SCREEN | SCR
Removes all SQL commands, data currently displayed, and CLPPlus messages from the screen. When the **CLEAR** command is entered with no options, the default behavior clears the screen.

BREAKS | BRE ON *column*
Clears all breaks when no column is specified. When a column is specified, the break associated with that column is cleared, all other breaks are left intact.

BUFFER | BUF and SQL
Deletes all text in the SQL buffer. You must specify both the **BUFFER** parameter (or the **BUF** parameter) and the **SQL** parameter.

COLUMNS | COL
Removes column definitions in the SQL buffer.

COMPUTES | COMP ON *column*
Clears all compute definitions when no column is specified. When a column is specified, the compute definition associated with that column is cleared, all other compute definitions are left intact.

VARIABLES | VAR
Clears all defined bind variables.

COLUMN

The **COLUMN** CLPPlus command specifies character and numeric output formats for columns in a table. Formats set by the **COLUMN** command remain in effect only for the duration of the current session. You can change or clear format settings for the same column more than once in the current session.

When you issue **COLUMN** for a specified column in a database, format settings are by default displayed using the standard output of the CLPPlus interface.

Invocation

This command must be executed from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►—COLUMN—┬─────────────────────────────────────────────────────────┬——————————►◄
          └column——┬CLEAR | CLE—————————————————————————————————────┤
                   │     ┌───────────────────────────────────────┐  │
                   │     ▼                                       │  │
                   └─────┬ALIAS—alias-name────────────────────┬──┴──┘
                         ├FORMAT | FOR—spec───────────────────┤
                         ├HEADING | HEA—text──────────────────┤
                         ├FOLD_BEFORE─────────────────────────┤
                         ├FOLD_AFTER──────────────────────────┤
                         ├LIKE—source-column──────────────────┤
                         ├NEWLINE─────────────────────────────┤
                         ├NEW_VALUE—variable-name─────────────┤
                         ├NULL—text───────────────────────────┤
                         ├OLD_VALUE—variable-name─────────────┤
                         ├PRINT | NOPRINT─────────────────────┤
                         ├WRAPPED | TRUNCATED─────────────────┤
                         ├JUSTIFY [ LEFT | RIGHT | CENTER ]───┤
                         ├OFF─────────────────────────────────┤
                         └ON──────────────────────────────────┘
```

Command parameters

column
Specifies the name of a table column to which formats are applied. If you do not specify any parameters after *column*, default format settings apply. If you do not specify any parameters after *column*, and you have set parameters for the same column previously in this session, the last set of parameters assigned to the column are again used.

ALIAS *alias-name*
Specifies and alias name for *column*. The maximum length of *alias-name* is 255 characters. The alias can be used in other commands such as: **COMMAND**, **COMPUTE**, and **COLUMN**.

CLEAR | CLE
Changes all formatting settings for the specified column to their default values. If you specify **CLEAR**, you cannot specify any parameters other than *column*.

FORMAT | FOR *spec*
Specifies the formats to apply to the specified column. There are two types of columns: character columns and numeric columns.

For a character column, the value of *spec* is A*n*, where *n* specifies the number of characters that can be used to display the column. The data wraps if it is wider than the specified width.

For numeric columns, the value of *spec* can be one or more of the following characters:

$ Displays a leading dollar sign.

, Displays a comma at the indicated position.

. Displays a decimal point at the indicated position.

0 Displays a zero at the indicated position.

9 Displays a significant digit at the indicated position.

If loss of significant digits occurs due to overflow of the format settings, the # character will be displayed.

HEADING | HEA *text*
Specifies a heading for the specified column.

FOLD_BEFORE
Before printing the values for the specified column, new line feed and carriage return are provided for each row.

FOLD_AFTER
After printing the values for the specified column, new line feed and carriage return are provided for each row.

LIKE *source-column*
The format and display attributes of *source-column* are applied to *column*.

NEWLINE
A synonym of **FOLD_AFTER**. After printing the values for the specified column, new line feed and carriage return are provided for each row.

NEW_VALUE *variable_name*
Defines a variable that can hold the new value for a break column defined using the **BREAK** command. The variable can be used with page top title **TTITLE** command. The break column must be defined with the **SKIP PAGE** action.

The **NEW_VALUE** command can also be used in all places within the current session. Similar to a substitution variable. Whenever you define a **NEW_VALUE** variable for a column, CLPPlus creates a substitution variable with the specified variable name. This variable is updated with the column value on each column break.

NULL *text*
When the value for the specified column is NULL, the value specified for *text* is printed. The maximum length of *text* is 255 characters.

OLD_VALUE *variable_name*
Defines a variable that can hold the old value for a break column defined using the **BREAK** command. The variable can be used with page bottom title **BTITLE** command. The break column must be defined with the **SKIP PAGE** action.

The **OLD_VALUE** command can also be used in all places within the current session. Similar to a substitution variable. Whenever you define a **OLD_VALUE** variable for a column, CLPPlus creates a substitution variable with the specified variable name. This variable is updated with the column value on each column break.

PRINT | NOPRINT
Specifies whether console printing of a specified column is enabled or disabled.

WRAPPED | TRUNCATED
Specifies if column data is wrapped or truncated in the CLPPlus output if it exceeds the specified format.

JUSTIFY [LEFT | RIGHT | CENTER]
Specifies column justification to be either LEFT, RIGHT, or CENTER.

OFF
Changes the formatting options to the default values. The values that you previously specified for the column in the session are saved and still available for use later in the session.

ON
Changes the formatting options to the values applied to the specified column the last time that you ran **COLUMN**.

Examples

In the following example, the **SET PAGESIZE** command sets the maximum page length to 9999, and the **COLUMN** command changes the display width of the JOB column to five characters. The SELECT statement then prints specified columns in the table.

```
SQL> SET PAGESIZE 9999
SQL> COLUMN JOB FORMAT A5
SQL> COLUMN JOB
COLUMN     JOB     ON
FORMAT     A5
WRAPPED
SQL> SELECT EMPNO, ENAME, JOB FROM EMP;

EMPNO   ENAME       JOB
------  ----------  -----
  7369  SMITH       CLERK
  7499  ALLEN       SALES
                    MAN
  7521  WARD        SALES
                    MAN
  7566  JONES       MANAG
                    ER
  7654  MARTING     SALES
                    MAN
  7698  BLAKE       MANAG
                    ER
  7782  CLARK       MANAG
                    ER
  7788  SCOTT       ANALY
                    ST
  7839  KING        PRESI
                    DENT
  7844  TURNER      SALES
                    MAN
  7876  ADAMS       CLERK
  7900  JAMES       CLERK
  7902  FORD        ANALY
                    ST
  7934  MILLER      CLERK

14 rows received.
```

In the following example, the **COLUMN** command applies a numeric format to the SAL column:

```
SQL> COLUMN SAL FORMAT $99,999.00
SQL> COLUMN
COLUMN   JOB  ON
FORMAT   A5
WRAPPED
```

```
COLUMN   SAL  ON
FORMAT   $99,999.00
WRAPPED
SQL> SELECT EMPNO, ENAME, JOB, SAL FROM EMP;

EMPNO ENAME      JOB           SAL
----- ---------- ----- -----------
 7369 SMITH      CLERK     $800.00
 7499 ALLEN      SALES   $1,600.00
                 MAN

 7521 WARD       SALES   $1,250.00
                 MAN

 7566 JONES      MANAG   $2,975.00
                 ER

 7654 MARTIN     SALES   $1,250.00
                 MAN

 7698 BLAKE      MANAG   $2,850.00
                 ER

 7782 CLARK      MANAG   $2,450.00
                 ER

 7788 SCOTT      ANALY   $3,000.00
                 ST

 7839 KING       PRESI   $5,000.00
                 DENT

 7844 TURNER     SALES   $1,500.00
                 MAN

 7876 ADAMS      CLERK   $1,100.00
 7900 JAMES      CLERK     $950.00
 7902 FORD       ANALY   $3,000.00
                 ST

 7934 MILLER     CLERK   $1,300.00

14 rows retrieved.
```

In the following example, the improved **NEW_VALUE** parameter behavior is shown. The new **OLD_VALUE** behavior is identical:

```
SQL> break on empno skip 1
SQL> column empno new_value highest_sal
SQL> select empno from employee order by salary;

EMPNO
------
200340
******

000290
******

200330
******

000310
******

      ...
      ...
```

```
000070
******

000030
******

000010
******

SQL>DEFINE
DEFINE HIGHEST_SAL = 000010

SQL> select EMPNO, FIRSTNME, MIDINIT, LASTNAME from employee where empno=&highest_sal;
EMPNO  FIRSTNME     MIDINIT LASTNAME
------ ------------ ------- ---------------
000010 CHRISTINE    I       HAAS
```

COMPUTE

The **COMPUTE** CLPPlus command executes a specified function on the aggregate values of a defined column. The command works in conjunction with the **BREAK** CLPPlus command.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──COMPUTE──┬─SUM───┬──LABEL──text──OF──column1──ON──column2──────────────────►◄
             ├─MAX───┤
             ├─MIN───┤
             ├─AVG───┤
             ├─COUNT─┤
             ├─NUM───┤
             ├─STD───┤
             └─VAR───┘
```

Command parameters

SUM
: The SUM function adds the aggregate values on the column specified.

MIN
: The MIN function returns the smallest of the aggregate values on the column specified.

MAX
: The MAX function returns the largest of the aggregate values on the column specified.

AVG
: The AVG function returns the average of the aggregate values on the column specified.

COUNT
: The **COUNT** function counts the number of non-null values on the column specified.

NUM
: The NUM function returns the number of aggregate rows processed on the column specified.

STD
: The STD function returns the standard deviation of the aggregate values on the column specified.

VAR
: The VAR function returns the variance of the aggregate values on the column specified.

LABEL *text*
: Defines the text label that precedes the output of the function specified.

column1
: Specifies the column on which the function is executed.

column2
: Specifies the column on which the **BREAK** command is executed against.

Example

The following example highlights the usage of the **COMPUTE** command in conjunction with the **BREAK** command.

```
SQL> BREAK ON WORKDEPT SKIP 2;
SQL> COMPUTE AVG LABEL "Average" OF SALARY ON WORKDEPT;
SQL> COMPUTE MAX LABEL "Maximum" OF SALARY ON WORKDEPT;
SQL> SELECT WORKDEPT, EMPNO, SALARY FROM EMPLOYEE ORDER BY WORKDEPT;
```

Here is the output of the commands in the example.

```
WORKDEPT   EMPNO   SALARY
--------   -----   ------
A01        00100   75000.00
A01        00101   80000.00
A01        00102   70000.00
********           ------
Average            75000.00
Maximum            80000.00


A02        00103   80000.00
A02        00104   90000.00
A02        00105   85000.00
********           ------
Average            85000.00
Maximum            90000.00
```

CONNECT

The **CONNECT** CLPPlus command changes the user ID connected to a database, connects to a different database, or does both. The command also displays the result of the change.

Invocation

This command must be run from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►─CONNECT─┬─connection_identifier─┬──────────────────────────────────────────►◄
           ├─/─────────────────────┤
           └─@─dsn_alias───────────┘
```

connection_identifier:

```
├─user─┬─────────────┬─┬────────────┬─┬────────┬─┬────────────┬─────────────┤
       └─/─password──┘ ├─@hostname──┤ └─:─port─┘ └─/─database─┘
                       └─@variable──┘
```

with dsn_alias:

```
├─@─dsn_alias─┬──────────────────────────────────────────┬──────────────────┤
              ├─using(accesstoken accesstoken-string)────┤
              └─using(apikey apikey-string)──────────────┘
```

Command parameters

Do not include spaces between any of the parameters.

accesstoken
: Specifies the accesstoken which is received from obtained from IAM (Identity Access Management) service to connect to the database.

apikey
: Specifies the apikey which is received from obtained from IAM (Identity Access Management) service to connect to the database.

user
: Specifies the user ID to connect to the database. With **dsn_alias**, userid can also be in email id format, such as ibmid : abc@xy.ibm.com for Identity Access Management based Authentication. In the case of **apikey** and **accesstoken**, The userID is not specified with **dsn_alias**.

password
: Specifies the password that corresponds to the user ID.

hostname
: Specifies the name of the computer on which the database is located. For example, for a computer that is named ascender, specify `@ascender`.

variable
: Specifies the name of a variable, which contains CLPPlus related information.

This variable can define connection string type information and must be defined in a file that is called `login.sql`, which is read by the CLPPlus interface during start-up.

port
: Specifies the port number that receives connections on the computer where the database server is installed. The default is 50000.

database
: Specifies the database name to which the connection is made. The default is SAMPLE.

/
: Specifies the current operating system login user ID is used to connect to the database.

dsn_alias
: Specifies the database connection information is read from the IBM data server driver configuration file (`db2dsdriver.cfg`) with alias name *dsn_alias*. If *dsn_alias* is not found in the IBM data server driver configuration file, the string *dsn_alias* is used as a database name and all other connection parameters are obtained interactively. If you configure an LDAP directory server in the specified IBM data server driver configuration file, the following steps apply:

1. The database connection information is read from the DSN entry with alias name *dsn_alias* in the IBM data server driver configuration file.
2. If *dsn_alias* is not found in the IBM data server driver configuration file, the configured LDAP Directory server is searched for an entry with the name *dsn_alias*.
3. If *dsn_alias* is not found on the LDAP Directory server, *dsn_alias* is used as a database name and all other connection parameters are obtained interactively.

Note: CLPPlus supports connection attempts to databases with DSN aliases and the authentication mechanism that is defined in the IBM data server driver configuration. If you do not define an authentication mechanism in the IBM data server driver configuration file, a default authentication mechanism is used by CLPPlus. The default authentication mechanism is equivalent to that of JCC.

Examples

In the following example, the database connection is changed to database Db2 on the local host at port 5445 with user name smith:

```
SQL> CONNECT smith/mypassword@localhost:5445/db2
Connected to CLPlus 1.1.0.10 (localhost:5445/db2) AS smith
```

In the same session, the connection is changed to the user name CLPPlus. For this connection, localhost, 5444, and Db2 are maintained as the host, port, and database values.

```
SQL> CONNECT CLPPlus/password
Connected to CLPPlus 1.1.0.10 (localhost:5444/db2) AS CLPPlus
```

The following example attempts to connect to a database by first locating an IBM data server driver configuration file. If one is found, the default_dsn is read for the host, port, and database values. The current logon ID is used in the connection attempt. If no IBM data server driver configuration file is found, all required parameters are requested interactively.

```
SQL> CONNECT /
```

The following example attempts to connect to a database by fetching the parameters from the data_dsn alias in the IBM data server driver configuration file. The db2admin user ID is used in the connection. Any parameters that cannot be read are requested interactively.

```
SQL> CONNECT db2admin@data_dsn
```

In the following example, the `login.sql` file contains a variable definition that is used to attempt a connection to a database. The `login.sql` file contains `define connStr = localhost:50000/sample` and can be used in the **`CONNECT`** CLPPlus command as follows:

```
SQL> CONNECT db2admin@connStr
```

A connection by the db2admin user is attempted on the sample database on the localhost, which has a listener port number of 50000.

COPY

The **`COPY`** CLPPlus command copies data from a source database and table to a target database and table.

Invocation

You must run this command from the CLPPlus interface or from within a CLPPlus script file.

Authorization

None

Required connection

None

Command Syntax

```
►►──COPY──FROM──srcdb──TO──destdb──┬──────────┬──────────────────────────►
                                   ├─APPEND───┤
                                   ├─CREATE───┤
                                   ├─INSERT───┤
                                   └─REPLACE──┘

►──┬──────────────────────────────────┬─USING──query────────────────────►◄
   └─dest_table──[(col1, col2, ...)]──┘
```

srcdb, destdb:

```
├──┬─connection_identifier─┬──────────────────────────────────────────────┤
   ├─/─────────────────────┤
   └─@──dsn_alias──────────┘
```

connection_identifier:

```
├──user──┬──────────────┬──┬─────────┬──┬─────────┬──┬─────────────┬──────┤
         └─/──password──┘  └─@──host─┘  └─:──port─┘  └─/──database─┘
```

Command parameters

FROM *srcdb*
: Defines the connection details and database name from which the data is copied.

  Note: Either one, or both, of the FROM or TO parameters must be specified in the **COPY** command. If FROM is not specified, and TO is specified, the database you are currently connected to, if a connection exists, is used for the source database.

TO *destdb*
: Defines the connection details and database name, which the data is copied into.

  Note: Either one, or both, of the FROM or TO parameters must be specified in the **COPY** command. If TO is not specified, and FROM is specified, the database you are currently connected to, if a connection exists, is used for the target database.

APPEND
: Inserts data into the *dest_table*. If *dest_table* does not exist, you must specify the destination table definition with the *dest_table* and *[(col1, col2, ...)]* variables.

CREATE
: Creates *dest_table* and inserts the data. You must specify the destination table definition with the *dest_table* and *[(col1, col2, ...)]* variables. If *dest_table* exists, an error is returned.

INSERT
: Inserts data into the *dest_table*. If *dest_table* does not exist, an error is returned.

REPLACE
: You must specify the destination table definition with the *dest_table* and *[(col1, col2, ...)]* variables. The *dest_table* is dropped, re-created, and then data is inserted.

dest_table
: Target database table into which data is inserted.

query
: The SQL query that is used to get the data from the source database.

user
: Specifies the user ID to connect to the database.

password
: Specifies the password that corresponds to the user ID.

hostname
: Specifies the name of the computer on which the database is located. For example, for a computer that is named ascender, specify `@ascender`.

port
: Specifies the port number that receives connections on the computer where the database server is installed. The default is 50000.

database
: Specifies the database name to which the connection is made. The default is SAMPLE.

dsn_alias
: Specifies that the database connection information is read from the IBM data

server driver configuration file (`db2dsdriver.cfg`) from the dsn with alias name *dsn_alias*. If the specified *dsn_alias* is not found in the IBM data server driver configuration file, the string *dsn_alias* is used as a database name and all other connection parameters are obtained interactively.

Examples

The following command copies the rows in the emp table in the db1 database and appends them into the emp table in the db2 database.

```
COPY FROM u1@db1 TO u2@db2 APPEND emp USING SELECT * FROM emp;
```

The following command copies the rows in the emp table in the db1 database and appends them into the emp table in the db2 database. Since the target table does not exist in the database that is named db2, you must specify the table definition in the command.

```
COPY FROM u1@db1 TO u2@db2 APPEND emp (EmpId integer, name varchar(20)) USING SELECT * FROM emp;
```

The following command copies the rows in the emp table in the db1 database, creates the emp table in the db2 database, and inserts the rows into the newly defined table in db2.

```
COPY FROM u1@db1 TO u2@db2 CREATE emp (EmpId integer, name varchar(20)) USING SELECT * FROM emp;
```

The following command copies the rows in the emp table in the db1 database and inserts them into the emp table in the db2 database since the target table exists.

```
COPY FROM u1@db1 TO u2@db2 INSERT emp USING SELECT * FROM emp;
```

The following command copies the rows in the emp table in the db1 database, re-creates the emp table in the db2 database, and replaces the rows.

```
COPY FROM u1@db1 TO u2@db2 REPLACE emp (EmpId integer, name varchar(20)) USING SELECT * FROM emp;
```

DEFINE

The **DEFINE** CLPPlus command creates a user variable, also called a substitution variable, and specifies its value. The command also displays the value of one or more user variables.

Invocation

The **DEFINE** command must be executed from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
>>──┬─DEFINE─┬──┬──────────┬──┬──────┬─────────────────────────────><
    └─DEF────┘  └─variable─┘  └─text─┘
```

Command parameters

variable
: Specifies the name of a variable. If you specify *variable* without *text*, the name of the variable and its value are displayed. If you do not specify *variable*, the names of all variables and their values are displayed.

text
: Specifies text to assign to the variable specified by *variable*. If the text contains spaces, you must enclose it in double or single quotation marks. If the text does not contain spaces, quotation marks are optional.

Example

In the following example, the **DEFINE** command defines the Db2, DEPT, and NAME variables and then displays the values of all variables:

```
SQL> DEFINE DEPT = 20
SQL> DEFINE NAME = 'John Smith'
SQL> DEFINE DB2 = 'localhost:5445/sample'
SQL> DEFINE
DEFINE DB2 = "localhost:5445/sample"
DEFINE DEPT = "20"
DEFINE NAME = "John Smith"
```

DEFINE_EDITOR CLPPlus command

In Db2 Cancun Release 10.5.0.4, the CLPPlus interface introduces support for the **DEFINE_EDITOR** command. You can specify the editor that you want to use with the **EDIT** command during a CLPPlus session with the **DEFINE_EDITOR** command.

Invocation

The **DEFINE_EDITOR** command must be run from the CLPPlus interface.

Authorization

None.

Required connection

None.

Command syntax

```
>>--DEFINE_EDITOR--+---+--+-------------+-------------------------------><
                   '-=-'  '-editor_name-'
```

Command parameters

editor_name
: Specifies the name of the editor for the current CLPPlus session.

Examples

Example to use the `Vim` editor
: You can set the editor to `Vim` with the following command:

```
SQL> DEFINE_EDITOR=vim
```

Example to use the `Notepad++` editor
: You can set the editor to `Notepad++` with the following command:

```
SQL> DEFINE_EDITOR= C:\Program Files (x86)\Notepad++\notepad++.exe
```

Example to display the current editor setting
You can check the current editor setting by issuing the **DEFINE_EDITOR** command without the assignment operator and an editor name.

```
SQL> DEFINE_EDITOR

define_editor = C:\Program Files (x86)\Notepad++\notepad++.exe
```

Example to clear the editor setting
You can clear the editor setting by entering the command with the assignment operator without an editor name.

```
SQL> DEFINE_EDITOR=
```

DEL

The **DEL** command deletes one or more lines from the SQL buffer.

Invocation

This command must be executed from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
         ┌─*──────┐
►►──DEL──┼────────┼──────────────────────────────────────────────►◄
         ├─n──────┤
         ├─n m────┤
         ├─n *────┤
         ├─n LAST─┤
         ├─n L────┤
         ├─* n────┤
         ├─* LAST─┤
         └─* L────┘
```

Command parameters

You can use the parameters to specify the start and end of a range of lines to delete from the SQL buffer. If you do not specify any parameters, the current line is deleted.

n Specifies a line number.

n m
Specifies two line numbers, where the value of *m* is greater than the value of *n*.

***** Indicates the current line.

LAST | L
Indicates the last line in the SQL buffer.

Example

In the following example, the fifth line, containing column SAL, and the sixth line, containing column COMM, are deleted from the SELECT statement in the SQL buffer:

```
SQL> LIST
  1  SELECT
  2    EMPNO
  3   ,ENAME
  4   ,JOB
  5   ,SAL
  6   ,COMM
  7   ,DEPTNO
  8* FROM EMP
SQL> DEL 5 6
SQL> LIST
  1  SELECT
  2    EMPNO
  3   ,ENAME
  4   ,JOB
  5   ,DEPTNO
  6* FROM EMP
```

The contents of line 7 becomes the contents of line 5, and the contents of line 8 becomes the contents of line 6. The contents of the current line have not changed, but the current line, marked with an asterisk, has changed from line 8 to line 6.

DESCRIBE

The **DESCRIBE** CLPPlus command displays a list of columns and their data types and lengths for a table view; a list of parameters for a procedure or function; or a list of procedures and functions and their parameters for a package.

The **DESCRIBE** CLPPlus command allows you to specify type of database object you want to describe. If you do not specify the type of object you want to describe, then all objects that are found with the given name and schema are described. The default schema is *CURRENT SCHEMA*.

The **DESCRIBE** CLPPlus command supports temporal tables. Temporal tables are new for Db2 for z/OS Version 10.

Invocation

You can run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──┬─DESCRIBE─┬──┬─────────────┬──┬──────────┬─object──────────────────────►◄
    └─DESC─────┘  ├─TABLE───────┤  └─schema.──┘
                  ├─VIEW────────┤
                  ├─ALIAS───────┤
                  ├─PROCEDURE───┤
                  ├─FUNCTION────┤
                  ├─SYNONYM─────┤
                  ├─PACKAGE─────┤
                  └─MODULE──────┘
```

Command parameters

TABLE
Specifies that the type of database object to be described is a table.

VIEW
Specifies that the type of database object to be described is a view.

ALIAS
Specifies that the type of database object to be described is a alias.

PROCEDURE
Specifies that the type of database object to be described is a procedure.

FUNCTION
Specifies that the type of database object to be described is a function.

SYNONYM
Specifies that the type of database object to be described is a synonym.

PACKAGE
Specifies that the type of database object to be described is a package.

MODULE
Specifies that the type of database object to be described is a module.

schema
Specifies the name of the schema containing an object to be described. The default schema is *CURRENT SCHEMA*.

object
Specifies the name of a table, view, procedure, function, or package to be described.

Example

In the following example, the **DESCRIBE** command is run to get details on the table named **ABCD**.

```
SQL> DESCRIBE TABLE ABCD ;

TABLE - ABCD


Name                 Data Type       Type schema       Length    Scale Nulls Hidden
-------------------- --------------- --------------- -------- -------- ----- --------
ROLL                 INTEGER         SYSIBM                 4        0 Y     N
NAME                 CHARACTER       SYSIBM                10        0 Y     P
```

In the following example, the **DESCRIBE** command is run to get details on a bitemporal table.

```
SQL > create table policy
(
  policy_id int NOT NULL,
  coverage int NOT NULL  IMPLICITLY HIDDEN,
  bus_start date NOT NULL,
  bus_end date NOT NULL,
  system_start TIMESTAMP(12) generated always as row begin NOT NULL,
  system_end TIMESTAMP(12) generated always as row end NOT NULL,
  trans_start   generated always as transaction start ID,
  period BUSINESS_TIME(bus_start, bus_end),
  period SYSTEM_TIME (system_start, system_end)
);

DB250000I: The command completed successfully.

SQL> create table policy_hist LIKE policy;

DB250000I: The command completed successfully.

SQL> ALTER TABLE policy ADD VERSIONING USE HISTORY TABLE policy_hist;

DB250000I: The command completed successfully.

SQL> describe policy
TABLE - POLICY
********************************************************************************

Name                Data Type     Type schema   Length   Scale   Nulls Hidden
------------------- ------------- ------------- -------- ------- ----- --------
POLICY_ID           INTEGER       SYSIBM               4       0 N     Not
COVERAGE            INTEGER       SYSIBM               4       0 N     Implicit
BUS_START           TIMESTAMP     SYSIBM               7       0 N     Not
BUS_END             TIMESTAMP     SYSIBM               7       0 N     Not
SYSTEM_START        TIMESTAMP     SYSIBM              13      12 N     Not
SYSTEM_END          TIMESTAMP     SYSIBM              13      12 N     Not
TRANS_START         TIMESTAMP     SYSIBM              13      12 Y     Not


Temporal Type : Bitemporal

Table is versioned and has the following periods
---------------------------------------------------------

Name                 Type Begin Column    End Column
-------------------- ---- --------------- ---------------
SYSTEM_TIME          S    SYSTEM_START    SYSTEM_END
BUSINESS_TIME        A    BUS_START       BUS_END

********************************************************************************
```

In the following example, the **DESCRIBE** command is run to get details on a table with system period, but not versioned.

```
SQL> create table demo_nontemp
(
   policy_id int NOT NULL,
   coverage int NOT NULL  IMPLICITLY HIDDEN,
   system_start TIMESTAMP(12) generated always as row begin NOT NULL,
   system_end TIMESTAMP(12) generated always as row end NOT NULL,
   trans_start   generated always as transaction start ID,
   period SYSTEM_TIME (system_start, system_end)
);

DB250000I: The command completed successfully.

SQL> describe demo_nontemp
```

```
TABLE - TEMPTAB
********************************************************************************

Name                Data Type     Type schema   Length   Scale   Nulls Hidden
------------------- ------------- ------------- -------- ------- ----- --------
POLICY_ID           INTEGER       SYSIBM               4       0 N     Not
COVERAGE            INTEGER       SYSIBM               4       0 N     Implicit
SYSTEM_START        TIMESTAMP     SYSIBM              13      12 N     Not
SYSTEM_END          TIMESTAMP     SYSIBM              13      12 N     Not
TRANS_START         TIMESTAMP     SYSIBM              13      12 Y     Not


Table has the following periods
----------------------------------------------------------

Name                 Type Begin Column    End Column
-------------------- ---- --------------- ---------------
SYSTEM_TIME          S    SYSTEM_START    SYSTEM_END

********************************************************************************
```

In the following example, the **DESCRIBE** command is run to get details on a application period temporal table.

```
SQL> create table demo_app
     (
      policy_id int NOT NULL,
     coverage int NOT NULL  IMPLICITLY HIDDEN,
     bus_start date NOT NULL,
     bus_end date NOT NULL,
      period BUSINESS_TIME(bus_start, bus_end));


DB250000I: The command completed successfully.

SQL> describe demo_app

TABLE - DEMO_APP
********************************************************************************

Name                Data Type     Type schema   Length   Scale   Nulls Hidden
------------------- ------------- ------------- -------- ------- ----- --------
POLICY_ID           INTEGER       SYSIBM               4       0 N     Not
COVERAGE            INTEGER       SYSIBM               4       0 N     Implicit
BUS_START           TIMESTAMP     SYSIBM               7       0 N     Not
BUS_END             TIMESTAMP     SYSIBM               7       0 N     Not


Temporal Type : Application period temporal

Table has the following periods
----------------------------------------------------------

Name                 Type Begin Column    End Column
-------------------- ---- --------------- ---------------
BUSINESS_TIME        A    BUS_START       BUS_END

********************************************************************************
```

In the following example, the **DESCRIBE** command is run to get details on a system period temporal table.

```
SQL> create table demo_sys
(
   policy_id int NOT NULL,
   coverage int NOT NULL  IMPLICITLY HIDDEN,
   system_start TIMESTAMP(12) generated always as row begin NOT NULL,
   system_end TIMESTAMP(12) generated always as row end NOT NULL,
   trans_start   generated always as transaction start ID,
   period SYSTEM_TIME (system_start, system_end)
);

DB250000I: The command completed successfully.

SQL> create table demo_sys_history like demo_sys ;

DB250000I: The command completed successfully.

SQL> ALTER TABLE DEMO_SYS ADD VERSIONING USE HISTORY TABLE DEMO_SYS_HISTORY;

DB250000I: The command completed successfully.

SQL> desc demo_sys

TABLE - DEMO_SYS
********************************************************************************

Name                Data Type     Type schema   Length   Scale   Nulls Hidden
------------------- ------------- ------------- -------- ------- ----- --------
POLICY_ID           INTEGER       SYSIBM               4       0 N     Not
COVERAGE            INTEGER       SYSIBM               4       0 N     Implicit
SYSTEM_START        TIMESTAMP     SYSIBM              13      12 N     Not
SYSTEM_END          TIMESTAMP     SYSIBM              13      12 N     Not
TRANS_START         TIMESTAMP     SYSIBM              13      12 Y     Not


Temporal Type : System period temporal

Table is versioned and has the following periods
---------------------------------------------------------

Name                 Type Begin Column    End Column
-------------------- ---- --------------- ---------------
SYSTEM_TIME          S    SYSTEM_START    SYSTEM_END

********************************************************************************
```

DISCONNECT

The **DISCONNECT** CLPPlus command closes the current database connection but does not exit the CLPPlus session.

Invocation

You can run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►─┬─DISCONNECT─┬──────────────────────────────────────────────────────────►◄
   └─DISC───────┘
```

EDIT

The **EDIT** CLPPlus command starts an external editor to make large changes to the contents of a file or the SQL buffer.

CLPPlus reads the **EDITOR** and **PATH** system environment variables to establish which external editor is used when the **EDIT** command is started. Any editor of your choice can be specified in the **EDITOR** system environment variable provided it is installed on the system. The location of the binary file for the external editor specified in the **EDITOR** system environment variable location must be included in your **PATH** system environment variable. If these variables are not set or not set properly, the default editor used on Windows operating systems is Notepad. On UNIX and Linux operating systems, it is vi.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─EDIT─┬──┬──────┬──┬──────────┬──────────────────────────────►◄
    └─ED───┘  └─path─┘  └─filename─┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the file specified by the *filename* variable. If no path is specified, the current directory is used.

filename
: Specifies the name of the file to open. If you do not specify a file extension, the `.sql` extension is used. If you do not specify the *filename* parameter, the contents of the SQL buffer are brought into the editor.

The **EDIT** command is supported in the CLPPlus window mode. The command is not supported in the CLPPlus non-window mode.

EXPORT CLPPlus using external tables

Use the EXPORT CLPPlus command to export an external table file to a local server location, a remote client, a Swift Object Store, or an AWS S3 object store.

Invocation

You must run the **EXPORT** command from the CLPPlus interface.

Authorization

None.

Required connection

You must be connected to a database.

Restrictions

The options such as **DATAOBJECT**, which are not supported with transient external tables, cannot be specified in the **OPTIONS** clause of the **EXPORT** command.

Command syntax

```
►►──EXPORT──EXTERNAL──TO──'──filename──'──┬──────────────────────────┬────────────►
                                          └─OPTIONS──(options-string)─┘

►──┬──────────────────┬────────────────────────────────────────────────────────────►◄
   └─select-statement─┘
```

Command parameters

EXTERNAL
: Specifies that the **EXPORT** command uses external table operations.

OPTIONS ***options-string***
: Specifies the options that control the processing of the export operation. These are described in CREATE EXTERNAL TABLE.

TO ***filename***
: Specifies the name of the file to which data is to be exported. If the file already exists, the contents of the file are overwritten, not appended to. The name must be specified in single quotes.

Select-statement
: Specifies the **SELECT** statement that is to return the data that is to be exported.

Example

The following command exports the content of the Employees table on the server to the Employees.txt file in Swift Object Store:

```
SQL> Export external to 'Employees.txt'
     options(s3('s3.amazonaws.com', 'AKIA99999999999999999', '783nGlH12345678910', 'db2.s3.qa.us-eas
     "DELIMITER ',' LOGDIR '/home/user') select * from employees;
```

The following command exports the content of the Employees table on the server to the Employees.txt file in an AWS S3 object store:

```
SQL> Export external to 'Employees.txt'
     options(swift('default', 'IBMOS28999999999', 'b107aa9172c70f8df16', 'db2_dev')
     DELIMITER ',' LOGDIR '/home/user/') select * from employees;
```

The following command exports the content of the Employees table on the server to the Employees.txt file on the client location:

```
SQL> Export external to 'C:\Employees.txt'
     options('maxerrors 20 REMOTESOURCE 'JDBC' LOGDIR '/home/user') select * from employees;
```

The following command exports the content of the Employees table on the server to the Employees.txt file on the server location:

```
SQL> Export external to '/home/user/Employees.txt'
     options(maxerrors 20) select * from employees;
```

EXECUTE

The **EXECUTE** CPPPlus command runs a procedure in the currently connected database. It is also used to define variables and run single-line PL/SQL statements. For a Db2 database connection, when you run this command, a Db2CALL statement is issued.

Invocation

You must run this command from the CLPPlus interface.

Authorization

You must ensure that the user ID that is used to run the **EXECUTE** CPPPlus command has one of the following privileges:

- EXECUTE privilege on the procedure
- DATAACCESS authority

If a matching procedure exists that the authorization ID of the statement is not authorized to run, an error is returned (SQLSTATE 42501).

Required connection

You must be connected to a database.

Command syntax

```
>>-+-EXECUTE-+--+-procedure-name--+-----------------------------+-+--+-;-+----------><
   '-EXEC----'  |                 |   .-,------------.          | |  '-/-'
                |                 |   V              |          | |
                |                 '-(----| argument |-+--)------' |
                +-PL/SQL-statement--------------------------------+
                '-variable-definition-----------------------------'
```

argument:

```
|--+----------------+--+-expression-+--------------------------------------------|
   '-parameter-name-'  +-DEFAULT----+
                       '-NULL-------'
```

Command parameters

procedure-name
: Specifies the procedure to call. The procedure must be cataloged. For Db2 data servers, you can qualify the name with a schema name, a module name, or both a schema name and a module name. The procedure to run is chosen by the procedure resolution algorithm. For Db2databases, the execution of the **EXECUTE** command, including procedure resolution, is the same as the Db2CALL statement.

PL/SQL-statement
: Specifies the PL/SQL statement to run.

variable-definition
: Specifies the definition of a variable.

argument

parameter-name
For Db2 data servers only, specifies the name of the parameter to which a value is assigned. If you assign a value to a parameter by name, all subsequent value assignments must also be by name.

All named parameters that you use to run the procedure must exist in the procedure definition.

Parameter names must be unique.

You cannot use named parameters to run uncataloged procedures.

value
Specifies the value that is associated with a parameter. The *n*th unnamed value corresponds to the *n*th parameter defined in the **CREATE PROCEDURE** statement for the procedure. Named values correspond to the same named parameter, regardless of the order in which you specify them.

expression
Passes a user-specified list of parameter values to the procedure that is called.

NULL
Passes NULL as the parameter value.

DEFAULT
If a default value is defined in the **CREATE PROCEDURE** statement, the specified default is passed as the parameter value. If no default value is specified, the NULL value is passed as the parameter value.

For Db2 databases, you must specify a value for each parameter that is not defined to have a default value (SQLSTATE 428HF). Also, for Db2 databases, each value must be compatible with the corresponding parameter in the procedure definition, as follows:

- IN parameter
  - The value must be assignable to the parameter.
  - The assignment of a string argument uses the storage assignment rules.
- OUT parameter
  - The value must be a single variable or parameter marker (SQLSTATE 42886).
  - The value must be assignable to the parameter.
  - The assignment of a string value uses the retrieval assignment rules.

  Note: You cannot display the following output data type values in the CLPPlus interface: row, array, associative array, and Boolean.
- INOUT parameter
  - The value must be a single variable or parameter marker (SQLSTATE 42886).
  - The value must be assignable to the parameter.
  - The assignment of a string value uses the storage assignment rules on invocation and the retrieval assignment rules on return.

Examples

1. The **CREATE PROCEDURE** statement creates a procedure that is called save_tester_details_PROC. The **EXECUTE** command runs this procedure.

```
>   SQL> CREATE PROCEDURE save_tester_details_PROC
           (tno, IN integer, tname IN varchar, tadd IN varchar)
           AS
            BEGIN
             INSERT INTO tester1 VALUES
             (tno, tname, tadd);
            END;
    /

>   The SQL command completed successfully.

>   SQL> EXECUTE save_tester_details_PROC(1, 'John Smith', 'Address1');
>   DB250000I: The SQL command completed successfully.
```

2. The **EXECUTE** command spans multiple lines and the block terminator / is used to submit the command for processing. The block terminator / must be used at the end of a command, which spans multiple lines.

```
SQL> exec dbms_output.put_line('test serveroutput')
2   /
test serveroutput
DB250000I: The command completed successfully.
```

3. The **EXECUTE** command runs a single PL/SQL statement.

```
SQL> Exec BEGIN dbms_output.put_line('TEST EXEC'); END
     2
     /
DB250000I: The command completed successfully.
```

4. The **EXECUTE** command defines a variable.

```
SQL> Variable bindvar varchar(20)
SQL> Execute :bindvar := 'value' ;
```

EXIT

The **EXIT** CLPPlus command ends the CLPPlus session and returns control to the operating system. This command is synonymous with the **QUIT** command.

Invocation

You must run this command from the CLPPlus interface.

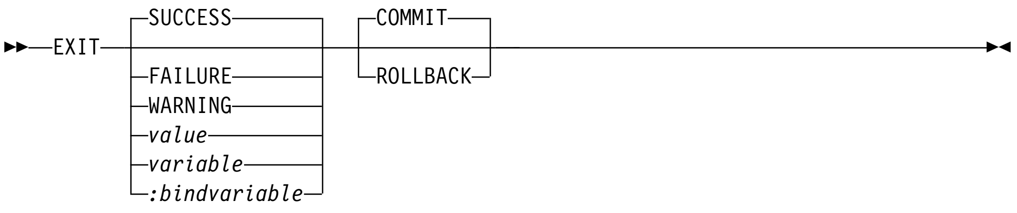
Authorization

None

Required connection

None

Command syntax



Command parameters

SUCCESS
Returns an operating system-dependant return code that indicates success.

FAILURE
Returns an operating system-dependant return code that indicates failure.

WARNING
Returns an operating system-dependant return code that indicates a warning.

value
Specifies a variable that is created by the **DEFINE** command whose value is returned as the return code.

variable
Specifies a substitution variable value that is created by the **DEFINE** command whose value is returned as the return code.

:bindvariable
Specifies a Bind variable value that is created by the **DEFINE** command whose value is returned as the return code.

COMMIT
Specifies that uncommitted updates are committed when the CLPPlus session ends.

ROLLBACK
Specifies that uncommitted updates are rolled back when the CLPPlus session ends.

Examples

In the following example, the **EXIT** command is issued with the **SUCCESS** parameter.

```
SQL> exit success
```

You can review whether the return code indicates success by running the **echo %errorlevel%** command.

```
echo %errorlevel%
0
```

In the following example, the **EXIT** command is issued with the **WARNING** and **COMMIT** parameters.

```
SQL> exit warning commit
```

You can review the return code for by running the **echo %errorlevel%** command.

```
echo %errorlevel%
2
```

In the following examples, the substitution variable *exit_value* is defined, set to 5, and used in the **EXIT** command.

```
SQL> variable exit_value integer
DB250000I: The command completed successfully.
SQL> exec :exit_value:=5 ;
DB250000I: The command completed successfully.
SQL> exit :exit_value rollback
```

You can review the return code for by running the **echo %errorlevel%** command.

```
echo %errorlevel%
5
```

EXPLAIN PLAN

The **EXPLAIN PLAN** CLPPlus command retrieves explain plan information for any single SQL statement.

The **EXPLAIN PLAN** CLPPlus command is supported on Db2 for z/OS and IBM Informix.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Restrictions

Support on IBM Informix has these restrictions:
- Only SELECT statements are supported.
- You must create and specify a default sbspace name for the **SBSPACENAME** configuration parameter in the `ONCONFIG` file. This sbspace is used for creating BLOB objects when an explain plan is created.
- To retrieve statistics data from an Informix server, your user ID must have the DBA privilege on the Informix database. Only user IDs with this privilege have access to statistics data.

Syntax diagram

```
►►—EXPLAIN—PLAN—FOR—SQL-statement————————————————————————————————►◄
```

Command parameters

SQL-statement
The SQL statement on which explain information is retrieved. For IBM Informix only **SELECT** statements are supported.

Example

```
SQL> explain plan for select * from emp where bonus > 1000 and salary>10000;

    ID TYPE           OBJECT_SCHEMA       OBJECT_NAME        PREDICATE_TEXT
------ -------------- ------------------- ------------------ ----------------------
     1 RETURN
     2 TBSCAN         MANSHANB            EMPLOYEE           (10000 < Q1.SALARY)
     2 TBSCAN         MANSHANB            EMPLOYEE           (1000 < Q1.BONUS)
```

GET

The **GET** CLPPlus command loads the contents of a text file into the CLPPlus SQL buffer.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
                               ┌─LIS────┐
                               ├─LIST───┤
►►─GET──┬──────┬──filename─────┤        ├──────────────────────────────────►◄
        └─path─┘               ├─NOLIST─┤
                               └─NOL────┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the file specified by the *filename* variable. If no path is specified, the current directory is used.

filename
: Specifies the name of the file to load into the SQL buffer. If you do not specify a file extension, the `.sql` extension is used.

LIST | LIS
: Displays the contents of the SQL buffer after the file is loaded.

NOLIST | NOL
: Prevents the contents of the SQL buffer from being displayed after the file is loaded.

HELP

The **HELP** command displays an index of topics for CLPPlus or it displays help for a specific CLPPlus topic.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─HELP─┬──┬────────┬──────────────────────────────────────────►◄
    └─?────┘  ├─INDEX──┤
              └─topic──┘
```

Command parameters

INDEX
Displays an index all CLPPlus help topics.

topic
Displays help for a specific CLPPlus subject, for example, **ACCEPT**.

HOST

The **HOST** CLPPlus command runs an operating-system command in the CLPPlus interface.

Invocation

You must run this command in the CLPPlus interface.

Authorization

None

Required connection

None.

Command syntax

```
►►──┬─HOST─┬──os_command──────────────────────────────────────────►◄
    └─HO───┘
```

Command parameters

os_command
Specifies an operating-system command.

IMPORT CLPPlus command

The **IMPORT** CLPPlus command is supported from a remote CLPPlus client where the import file is processed on the same client.

Invocation

You must run the **IMPORT** command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Restrictions

- The parameters that are available for the **IMPORT** command in the CLPPlus interface are a subset of the parameters that are available for the **IMPORT** command in the CLP interface.
- Data can be imported only from a delimited file. The delimited file can be of any file type, such as: `.del`, `.ixf`, or `.txt`. The **,** character is the default delimiter. You can set the delimiter to another character by using the **SET** CLPPlus command.
- The target database table cannot be a system table, a declared temporary table, or a summary table.

Syntax diagram

```
►►—IMPORT—FROM—filename—┬──────────────────────────────────────┬—INSERT—INTO—table_name———————►◄
                        ├─┬COMMITCOUNT—n─┬─────────────────────┤
                        │ └COMMIT—n──────┘                     │
                        ├─┬RESTARTCOUNT—n—|—SKIPCOUNT—n─┬──────┤
                        │ └RESTART—n—|—SKIP—n───────────┘      │
                        └─┬ROWCOUNT—n─┬────────────────────────┘
                          └ROW—n──────┘
```

Command parameters

`filename`
: Specifies the file that contains the import data. You can specify a path as part of the *filename* variable. The path can be absolute or relative to the current directory. If the path is omitted, the current directory is searched.

COMMITCOUNT | COMMIT *n*
: When specified, **IMPORT** commits data after every *n* records are read and imported.

RESTARTCOUNT | RESTART *n*
: When specified, **IMPORT** starts at record *n+1*. The first n records are skipped. This option is functionally equivalent to **SKIPCOUNT**. **RESTARTCOUNT** and **SKIPCOUNT** are mutually exclusive.

SKIPCOUNT | SKIP *n*
: When specified, **IMPORT** starts at record *n+1*. The first n records are skipped. This option is functionally equivalent to **RESTARTCOUNT**. **RESTARTCOUNT** and **SKIPCOUNT** are mutually exclusive.

ROWCOUNT | ROW *n*
: When specified, *n* physical records from the beginning of *filename* are imported. When **ROWCOUNT** is specified with **RESTARTCOUNT** or **SKIPCOUNT**, **IMPORT** reads *n* rows from *filename* starting from the record that is defined by **RESTARTCOUNT** or **SKIPCOUNT**.

table_name
: Specifies the target database table for the **IMPORT** operation. This table cannot be a system table, a declared temporary table or a summary table. If not fully qualified with a schema, the default schema is the current ID.

Examples

The following **IMPORT** command reads the first 100 rows of the `c:\data.txt` file and inserts the data into the `db2admin.emptab` table:

```
import from c:\data.txt rowcount 100 insert into db2admin.emptab;
```

The following **IMPORT** command starts reading data at row 11 of the `data.txt` file and inserts the data into the `emptab` table:

```
import from data.txt skip 10 insert into emptab;
```

The following **IMPORT** command starts reading data at row 11 of the `data.txt` file, one directory up in the directory tree relative to the current directory. The command inserts the data into the `emptab` table.

```
import from ./data.txt restart 10 insert into emptab;
```

INPUT

The **INPUT** line-editor command adds a line of text to the SQL buffer after the current line.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─INPUT─┬──text──────────────────────────────────────────────►◄
    └─I─────┘
```

Command parameters

text
Specifies the text to be inserted into the SQL buffer.

Example

In the following example, the sequence of **INPUT** commands constructs a SELECT statement, which is displayed by the **LIST** command:

```
SQL> INPUT SELECT empno, ename, job, sal, comm
SQL> INPUT FROM emp
SQL> INPUT WHERE deptno = 20
SQL> INPUT ORDER BY empno
SQL> LIST
  1  SELECT empno, ename, job, sal, comm
  2  FROM emp
  3  WHERE deptno = 20
  4* ORDER BY empno
```

LIST

The **LIST** line-editor command displays all of the lines or a range of lines in the SQL buffer.

Invocation

You must run this command from the CLPPlus interface.

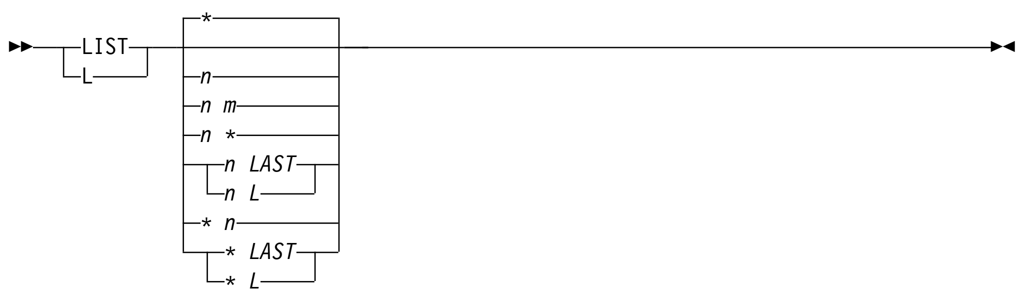
Authorization

None

Required connection

None

Command syntax



Command parameters

n Displays the specified line in the SQL buffer.

n m
Displays the specified range of lines in the SQL buffer, where the value of *m* is greater than the value of *n*.

***** Displays the current line in the SQL buffer.

LAST | L
Displays the last line in the SQL buffer.

Example

In the following example, the **L** command used to show line buffer contents.

```
SQL> LIST
 1  SELECT
 2  *
 3  FROM
 4* EMPLOYEE
SQL> L2
 2* *
SQL> L4
 4* EMPLOYEE
SQL> L5
DB250400E: The buffer line number is invalid.  Valid values can be between '1' and '4'.
SQL>
```

In the following example, a variant of the **LIST** command is used to show line buffer contents.

```
SQL> begin
2 dbms_output.put_line('list command ');
3 end;
4 /
```

```
DB250000I: The command completed successfully.

SQL> list
1  begin
2  dbms_output.put_line('list command ');
3* end

SQL> 2
2* dbms_output.put_line('list command ');

SQL>
```

Note in the previous example how the number 2 is entered on its own in. The result is the contents of line buffer 2 is displayed. You can enter any valid line number in the buffer to display the contents of the line.

LOAD CLPPlus command by using external tables

The LOAD CLPPlus command that uses external tables is supported for loading a file to a target Db2 server.

Invocation

You must run the LOAD command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──LOAD──EXTERNAL──FROM──┬─filename─┬──┬───────────────────────┬──────────────►
                          └─pipename─┘  └─NOT LOGGED INITIALLY─┘

►──┬──────────────────────────────────┬─────────────────────────────────────────►
   └─OPTIONS──(──options-string──)─┘

►─INTO──table-name──┬──────────────────────────────┬────────────────────────────►◄
                    └─(──insert-column-names──)─┘
```

Command parameters

EXTERNAL
Specifies that the LOAD command uses external table operations.

FROM *filename* or *pipename*
Specifies the file or pipe that contains the data that is being loaded.

NOT LOGGED INITIALLY
Data that is loaded into the table by a LOAD operation that is in the same unit of work is not logged.

OPTIONS(*options-string*)
Specifies the external table options that control the processing of the LOAD operation. Refer to the target server's external table options for details.

INTO ***table-name***
: Specifies the database table into which the data is to be loaded.

insert-column-names
: Specifies the comma-separated table columns into which the data is to be loaded.

Usage notes

LOG and BAD files contain information about records that are successfully loaded, rejected, or skipped. The location of these files depends on the location of the server:

Remote server (REMOTESOURCE 'LOCAL' is not specified)
: LOGDIR specifies the directory to which `~.log` and `~.bad` files are generated. If `log` and `bad` files are generated, the server returns the full path to store them by including the specified LOGDIR in the `log` and `bad` file path that is returned to the driver.

  If the LOGDIR option is not defined, the JDBC driver's output directory is used to store the `log` and `bad` files. This default output directory is the operating system default temp directory or the configured *db2.jcc.outputDirectory* directory.

Local server (REMOTESOURCE 'LOCAL' is specified)
: LOGDIR specifies the directory to which `~.log` and `~.bad` files are generated. When used with SWIFT or S3, the files are located in the object store where the paths of the `log` and `bad` files are relative to the top of the bucket or container.

ERROR_LOG can be used as a synonym for the LOGDIR option. If the LOGDIR option is not defined, then the default location is the location of the data files.

Output of LOAD CLPPlus command by using external tables

- If all rows of data are loaded successfully, the result of LOAD command returns information similar to the following output:

  ```
  Total number of rows loaded: 10
  DB250000I: The command completed successfully.
  ```
- If all rows of data are not loaded successfully, the result of LOAD command returns information similar to the following output:

  ```
  SQL5108W Loading of data to a Hadoop table or processing of data in an externaltable completed.
  Number of rows processed: "2". Number of source records: "3".
  If the source was a file, number of skipped lines: "0".
  Number of rejected source records: "1". Job or file identifier:
  "SAMPLE.REGRES1.SYSTET46273.019412".

  DB250000I: The command completed successfully.
  ```
-

Draft comment

I've omitted the bullet about "invalid syntax returns an error". That's obvious

If the server or JCC driver returns an error message, then the result of LOAD command displays the same error message in the CLP.

Examples

Example 1

In this example, the data file is on a client that is connected to the database. The data file is loaded remotely.

The target table is DB2TBL1, whose columns are ID [type INTEGER] and NAME [type varchar(50)].

The path and name of the data file is `C:\data\et.txt` and has the following content:

```
1, 'Name0'
2, 'Name1'
3, 'Name2'
```

1. Start CLPPlus.
2. Connect to the target database.
3. Run the LOAD command:

   ```
   LOAD EXTERNAL FROM C:\data\et.txt
      OPTIONS (DELIMITER ',' SOCKETBUFSIZE 30000 LOGDIR 'C:\data\' MAXERRORS 20 REMOTESOU
      INTO DB2TBL1;
   ```

   Note: REMOTESOURCE 'JDBC' is a mandatory option for a remote load.
4. Data is successfully loaded in target table with following output message:

   ```
   Total number of rows loaded: 3

   DB250000I: The command completed successfully.
   ```

Example 2

In this example, the data file exists on the target server and is loaded locally.

The target table is DB2TBL1, whose columns are ID [type INTEGER] and NAME [type VARCHAR(50)].

The path and name of the data file is `/home/regres1/et/et1.txt` and has the following content:

```
1, 'Name0'
2, 'Name1'
"3", 'Name2'
```

1. Start CLPPlus.
2. Connect to the target database.
3. Run the LOAD command:

   ```
   LOAD EXTERNAL FROM /home/regres1/et/et1.txt
      OPTIONS (DELIMITER ',' SOCKETBUFSIZE 30000 LOGDIR /home/regres1/et/' MAXERRORS 20)
      INTO DB2TBL1;
   ```

   Note: The keyword REMOTESOURCE cannot be used for a local load.
4. Partial data is successfully loaded in target table with following output message:

   ```
   SQL5108W Loading of data to a Hadoop table or processing of data in an external table
   Number of rows processed: "2". Number of source records: "3".
   If the source was a file, number of skipped lines: "0".
   Number of rejectedsource records: "1".
   Job or file identifier:
   "SAMPLE.REGRES1.SYSTET46273.019412".

   DB250000I: The command completed successfully.
   ```

Example 3

In this example, the data file that is in the AWS S3 object store.

The target table is DB2TBL1, whose columns are ID [type INTEGER] and NAME [type VARCHAR(50)].

The path and name of the data file is `C:\data\et.txt` and has the following content:

```
1, 'Name0'
2, 'Name1'
3, 'Name2'
```

1. Start CLPPlus.
2. Connect to the target database.
3. Run the LOAD command:

   Draft comment

   The preamble of this example mentions ain input file C:\data\et.txt. However, the path is the LOAD command refers to /home/regres1/et/et1.txt. Is this correct? Or is one of these filepaths incorrect?

   ```
   LOAD EXTERNAL FROM /home/regres1/et/et1.txt
      OPTIONS (S3 (ENDPOINT, AUTHKEY1, AUTHKEY2, BUCKET),
         DELIMITER ',' SOCKETBUFSIZE 30000 LOGDIR  /home/regres1/et/' MAXERRORS 20)
      INTO DB2TBL1;
   ```

   Note: The keyword REMOTESOURCE cannot be used with AWS S3 object store.
4. Data is successfully loaded in target table with following output message:

   ```
   Total number of rows loaded: 3

   DB250000I: The command completed successfully.
   ```

Example 4

In this example, the data file is in the Swift Object Store.

The target table is DB2TBL1, whose columns are ID [type INTEGER] and NAME [type VARCHAR(50)].

The path and name of the data file is `C:\data\et.txt` and has the following content:

```
1, 'Name0'
2, 'Name1'
3, 'Name2'
```

1. Start CLPPlus.
2. Connect to the target database.
3. Run the LOAD command:

   Draft comment

   The preamble of this example mentions ain input file C:\data\et.txt. However, the path is the LOAD command refers to /home/regres1/et/et1.txt. Is this correct? Or is one of these filepaths incorrect?

```
LOAD EXTERNAL FROM /home/regres1/et/et1.txt
   OPTIONS (SWIFT (ENDPOINT, AUTHKEY1, AUTHKEY2, CONTAINER),
      DELIMITER ',' SOCKETBUFSIZE 30000 LOGDIR  /home/regres1/et/' MAXERRORS 20)
   INTO DB2TBL1;
```

Note: The keyword REMOTESOURCE cannot be used with Swift Object Storage.

4. Data is successfully loaded in target table with following output message:

```
Total number of rows loaded: 3

DB250000I: The command completed successfully.
```

Example 5

This example demonstrates the use of the NOT LOGGED INITIALLY option.

The target table is DB2TBL1, whose columns are ID [type INTEGER] and NAME [type VARCHAR(50)].

The path and name of the data file is `C:\data\et.txt` and has the following content:

```
1, 'Name0'
2, 'Name1'
3, 'Name2'
```

1. Start CLPPlus.
2. Connect to the target database.
3. Run the LOAD command:

Draft comment

The preamble of this example mentions ain input file C:\data\et.txt. However, the path is the LOAD command refers to /home/regres1/et/et1.txt. Is this correct? Or is one of these filepaths incorrect?

```
LOAD EXTERNAL FROM 'C:\data\et1.txt' NOT LOGGED INITIALLY
   OPTIONS (DELIMITER ',' LOGDIR 'C:\data\' MAXERRORS 20 SOCKETBUFSIZE 30000 REMOTESOU
   INTO DB2TBL1(ID, NAME);
```

4. Data is successfully loaded in target table with following output message:

```
Total number of rows loaded: 3

DB250000I: The command completed successfully.
```

LOAD CLPPlus for z/OS

The LOAD CLPPlus command for z/OS is supported for remotely loading data against Db2 for z/OS servers.

Invocation

You must run the **LOAD** command from the CLPPlus interface.

Authorization

Both of the following authorities:

- Ownership of the table

- **LOAD** privilege for the database

Required connection

You must be connected to a database.

Command syntax

```
►►—LOAD—"<filename>"—LOADSTMT—"<loadstmt>"—UTILITYID—"<utilityid>"—————————►

►—VERBOSE——┬—ON——┬——————————————————————————————————————————————————————————►◄
           └—OFF—┘
```

Command parameters

filename
: File from which data is to be loaded. The parameter value has to be enclosed in double quotes.

LOADSTMT
: Load statement. User can directly specify the **LOADSTMT** in the command, or write the **LOADSTMT** in a file and specify the complete path of the file. This parameter value has to be enclosed in double quotes. Refer to Syntax and options of the LOAD control statement for a detailed definition on **LOADSTMT**

UTILITYID
: Specifies the utility-id for **LOAD** operation. This is an optional parameter. This parameter value has to be enclosed in double quotes. If the driver returns java.sql.SQLException, CLPPlus will display the same. If the driver returns `java.sql.SQLException`, CLPPlus will display the same.

VERBOSE
: Specifies whether CLPPlus should display the **LOAD** result on console. This is an optional parameter. The default value for this parameter is `ON`. Unless the user specifies verbose as `OFF`, CLPPlus will display the **LOAD** result on the console. If the CLPPlus tracing is enabled, **verbose** will be put into the trace file irrespective of the value specified for **verbose**.

Example

The following **LOAD** command loads the data from `block.cust.del` into Db2 for z/OS serve:

```
SQL> load file "C:\Users\IBM_ADMIN\Desktop\CLPPlus_Deliver\zFastLoadTesting\block.cust.del"
     loadstmt "TEMPLATE SORTIN DSN &JO..&ST..SORTIN.T&TIME. UNIT SYSVIO SPACE(10,10) CYL DISP(NEW,DE
     TEMPLATE SORTOUT DSN &JO..&ST..SORTOUT.T&TIME. UNIT SYSVIO SPACE(10,10) CYL DISP(NEW,DELETE,DEL
     LOAD DATA INDDN SYSCLIEN WORKDDN(SORTIN,SORTOUT)
     REPLACE PREFORMAT LOG(NO) REUSE NOCOPYPEND FORMAT DELIMITED EBCDIC INTO TABLE ADMF001.CUSTOMER_

DB250000I: The command completed successfully.
Return Code : 0
```

The following **LOAD** command loads the data from `block.cust.del` into Db2 for z/OS server. **LOADSTMT** is read from a file `loadStatement.txt`, and **VERBOSE** is `ON`:

```
SQL> Load file "C:\Users\IBM_ADMIN\Desktop\CLPPlus_Deliver\zFastLoadTesting\block.cust.del" loadstmt

DB250000I: The command completed successfully.
1DSNU000I    110 22:04:44.76 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = ZLOAD11110342195
 DSNU1045I   110 22:04:44.80 DSNUGTIS - PROCESSING SYSIN AS UNICODE UTF-8
0DSNU050I    110 22:04:44.80 DSNUGUTC -  TEMPLATE SORTIN DSN &JO..&ST..SORTIN.T&TIME. UNIT SYSVIO SP
```

```
 DISP(NEW, DELETE, DELETE)
 DSNU1035I   110 22:04:44.80 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    110 22:04:44.81 DSNUGUTC -  TEMPLATE SORTOUT DSN &JO..&ST..SORTOUT.T&TIME. UNIT SYSVI
 CYL DISP(NEW, DELETE, DELETE)
 DSNU1035I   110 22:04:44.81 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
0DSNU050I    110 22:04:44.81 DSNUGUTC -  LOAD DATA INDDN SYSCLIEN WORKDDN(SORTIN, SORTOUT) REPLACE
  REUSE NOCOPYPEND FORMAT DELIMITED EBCDIC
 DSNU650I  -DB2A 110 22:04:44.81 DSNURWI -  INTO TABLE ADMF001.CUSTOMER_LOCAL NUMRECS 30000
 DSNU1038I   110 22:04:44.89 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SORTIN
                       DDNAME=SYS00007
                       DSN=DB2AUTL1.DB2AUTL1.SORTIN.T050444
 DSNU1038I   110 22:04:44.90 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=SORTOUT
                       DDNAME=SYS00008
                       DSN=DB2AUTL1.DB2AUTL1.SORTOUT.T050444
 DSNU350I  -DB2A 110 22:04:44.97 DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
 DSNU3340I   110 22:04:44.98 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
 DSNU3342I   110 22:04:45.01 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 1, NUMBER OF ACTIVE SORT TA
 DSNU395I    110 22:04:45.01 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 2
 DSNU3345I   110 22:04:45.01 DSNURPIB - MAXIMUM UTILITY PARALLELISM IS 3 BASED ON NUMBER OF PARTIT
 DSNU397I    110 22:04:45.01 DSNURPIB - NUMBER OF TASKS CONSTRAINED BY CPUS TO 3
 DSNU304I  -DB2A 110 22:05:29.65 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=30000 FOR
 ADMF001.CUSTOMER_LOCAL
 DSNU1147I -DB2A 110 22:05:29.65 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOA
 TABLESPACE DSNDB04.TCUST000
 DSNU302I    110 22:05:29.65 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCES
 DSNU300I    110 22:05:29.65 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:44
 DSNU3350I   110 22:05:29.68 DSNUGSRP - SORT TASK SW01: 30000 RECORDS SORTED, ESTIMATED 30000, VAR
 DSNU3352I   110 22:05:29.70 DSNUGSRP - SORT TASK SW01: USED DFSORT
 DSNU3354I   110 22:05:29.70 DSNUGSRP - SORT TASK SW01: MEMORY BELOW THE BAR: OPTIMAL 6 MB, USED 6
 DSNU394I  -DB2A 110 22:05:29.74 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=30000 FOR IN
 DSNU391I    110 22:05:29.76 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 1
 DSNU392I    110 22:05:29.76 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
 DSNU3357I   110 22:05:29.79 DSNUGUTC - MAXIMUM SORT AMOUNT ESTIMATION VARIATION WAS 0 PERCENT
 DSNU3355I   110 22:05:29.79 DSNUGUTC - TOTAL SORT MEMORY BELOW THE BAR: OPTIMAL 6 MB, USED 6 MB
 DSNU010I    110 22:05:29.80 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

Return Code : 0
```

PAUSE

The **PAUSE** CLPPlus command suspends CLPPlus processing, optionally displays a message, and waits to resume processing until the ENTER key is pressed.

Invocation

You must run this command in the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──┬─PAUSE─┬──optional-text──────────────────────────────────────►◄
    └─PAU───┘
```

Command parameters

optional-text
: Specifies a message. The message is a string of characters that can include spaces and special characters. If you use quotation marks around the message, it is included in the command output or statement output. The character case specified as **optional-text** is maintained as entered.

PRINT

The **PRINT** CLPPLus command displays the value of a bind variable. Bind variables are used in place of literal values. If you issue a SELECT statement multiple times, you can use bind variables to reduce the number of literal values.

Invocation

You can run this command in the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►──┬─PRINT─┬──bind-variable-name──────────────────────────────────►◄
    └─PRI───┘
```

Command parameters

bind-variable-name
: Specifies a string of characters that can include spaces and special characters. If you do not specify *bind-variable-name*, the values of all bind variables are printed.

  If *bind-variable-name* is of the datatype **REFCURSOR**, the result set pointed to by *bind-variable-name* will be read in its entirety and all the rows will be printed following the report formatting specified in the current CLPPlus session.

PROMPT

The **PROMPT** CLPPlus command prints a line to the display.

Output of the **PROMPT** command is displayed in CLPPlus output.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─PROMPT─┬──optional-text──────────────────────────────────────────►◄
    ├─PRO────┤
    └─pro────┘
```

Command parameters

optional-text
: Specifies a message. The message is a string of characters that can include spaces and special characters. If you use quotation marks around the message, it is included in the command output or statement output. The case of letters is maintained.

QUIT

The **QUIT** CLPPlus command ends the CLPPlus session and returns control to the operating system. This command is synonymous with the **EXIT** command.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
           ┌─SUCCESS──────┐  ┌─COMMIT───┐
►►──QUIT───┼──────────────┼──┼──────────┼──────────────────────────────►◄
           ├─FAILURE──────┤  └─ROLLBACK─┘
           ├─WARNING──────┤
           ├─value────────┤
           └─sub-variable─┘
```

Command parameters

SUCCESS
: Returns an operating system-dependant return code indicating success.

FAILURE
: Returns an operating system-dependant return code indicating failure.

WARNING
: Returns an operating system-dependant return code indicating a warning.

value
: Specifies a variable created by the **DEFINE** command whose value is returned as the return code.

sub-variable
: Specifies a substitution variable that can be used to return information.

COMMIT
: Specifies that uncommitted updates are committed when the CLPPlus session ends.

ROLLBACK
: Specifies that uncommitted updates are rolled back when the CLPPlus session ends.

REMARK

The **REMARK** CLPPlus command allows for the coding of a comment in a script. This is similar to a comment line coded with two dashes.

Invocation

You must run this command from the CLPPlus interface. It can only be included in CLPPlus scripts.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─REMARK─┬──optional-text──────────────────────────────────►◄
    └─REM────┘
```

Command parameters

optional-text
: Specifies a string of characters that can include spaces and special characters. You can use the convention shown in the following example.

```
/*
 * This is a three-line comment
*/
```

REORGCHK

The **REORGCHK** CLPPlus command calculates statistics on the database to determine whether tables or indexes, or both, must be reorganized or cleaned up.

Invocation

You must run this command from the CLPPlus interface.

Authorization

You must have SELECT privilege on the catalog tables. You must have EXECUTE privilege on the REORGCHK_IX_STATS procedure.

Restriction

Support for this command is limited to Db2 servers.

Required connection

You must be connected to a database.

Syntax diagram

```
►►──REORGCHK──ON──┬─TABLE──┬─USER───────┬─┬──────────────────────────────────►◄
                  │        ├─SYSTEM─────┤ │
                  │        ├─ALL────────┤ │
                  │        └─table-name─┘ │
                  └─SCHEMA──schema-name───┘
```

Command parameters

TABLE [USER | SYSTEM | ALL | *table-name*]
: Where USER checks the tables that are owned by the authorization ID at runtime, SYSTEM checks the system tables, ALL checks all user and system tables, and *table-name* specifies which table to check. When you use *table-name*, the fully qualified name or alias must be used, for example, schema.tablename. The schema is the user name under which the table was created. If the table specified is a system catalog table, the schema is SYSIBM. For typed tables, the specified table name must be the name of the root table of the hierarchy.

SCHEMA *schema-name*
: Checks all the tables that are created under the specified schema.

Example

1. This example performs **REORGCHK** on system tables.
   ```
   SQL> reorgchk on table system
   ```
2. This example performs **REORGCHK** on table EMPLOYEE under schema manshanb.
   ```
   SQL> reorgchk on table manshanb.EMPLOYEE
   ```
3. This example performs **REORGCHK** on all user and system tables.
   ```
   SQL> reorgchk on table all
   ```
4. This example performs **REORGCHK** the tables that are owned by the runtime authorization ID.
   ```
   SQL> reorgchk on table user
   ```
5. This example performs **REORGCHK** on schema named manshanb.
   ```
   SQL> reorgchk on schema manshanb
   ```

REPFOOTER

The **REPFOOTER** CLPPlus command prints a report footer at the end of a report.

Invocation

You must run this command from the CLPPlus interface.

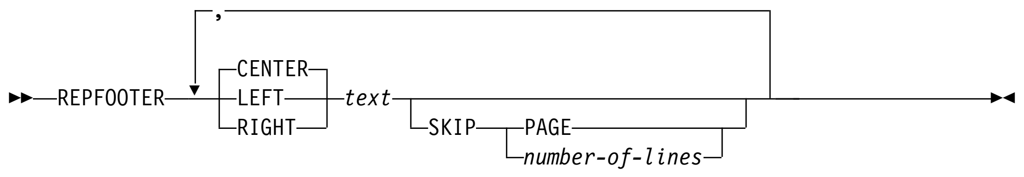
Authorization

None

Required connection

None

Command syntax



Command parameters

text
: Specifies the text displayed at the end of a report.

`CENTER`
: Specifies the display will center justify the report footer. If neither **`CENTER`**, **`LEFT`**, or **`RIGHT`** is specified, center justification is the default behavior.

`LEFT`
: Specifies the display will left justify the text for the report footer.

`RIGHT`
: Specifies the display will right justify the text for the report footer.

`SKIP`

  `PAGE`

  Specifies the report footer is displayed on the next new page.

  number-of-lines

  Specifies the number of lines to skip.

Example

In the following example, the report footer `END SALARY REPORT` is displayed with center justification at the end of the report on the next new page.

```
SQL> REPFOOTER CENTER 'END SALARY REPORT' SKIP PAGE;
```

In the following example, the report footer `Company Name` is displayed, two lines are skipped, then `End of Report` is displayed all with center justification on the next new page.

```
SQL> REPFOOTER CENTER "Company Name" SKIP 2, CENTER "End of Report" SKIP PAGE
```

REPHEADER

The **REPHEADER** CLPPlus command prints a report heading once at the beginning of a report.

Invocation

You must run this command from the CLPPlus interface.

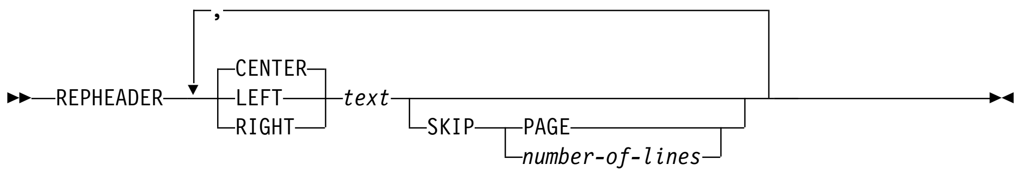
Authorization

None

Required connection

None

Command syntax



Command parameters

text
: Specifies the text displayed for the report header.

CENTER
: Specifies the display will center justify the report header. If neither **CENTER**, **LEFT**, or **RIGHT** is specified, center justification is the default behavior.

LEFT
: Specifies the display will left justify the text for the report header.

RIGHT
: Specifies the display will right justify the text for the report header.

SKIP

PAGE

Specifies the report header is displayed and the report data starts on the next new page.

number-of-lines

Specifies the number of lines to skip.

Example

In the following example, the report header SALARY REPORT is displayed with left justification and the report data starts on the next new page.

```
SQL> REPHEADER LEFT 'SALARY REPORT' SKIP PAGE;
```

In the following example, the report header COMPANY NAME is displayed, two lines are skipped, and then SALARY REPORT is displayed all with center justification. The report data starts on the next new page.

```
SQL> REPHEADER CENTER 'COMPANY NAME' SKIP 2, CENTER 'SALARY REPORT' SKIP PAGE;
```

RUN

The **RUN** CLPPlus command runs a SQL query or a PL/SQL command that is stored in the SQL buffer.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

You must be connected to a database.

Command syntax

```
►►—RUN——————————————————————————————————————————————————————————————►◄
```

Example

In the following example, the contents of the SQL buffer is populated with the `SELECT EMPNP FROM EMP` statement.

```
SQL> APPEND SELECT EMPNP FROM EMP
```

The **RUN** command issues the statement in the SQL buffer:

```
SQL> run
  1* SELECT EMPNO FROM EMP
```

The output is as follows:

```
EMPNO
-------
000010
000020
...
...
```

SAVE

The **SAVE** line-editor command stores the contents of the SQL buffer in a new or existing file.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─SAVE─┬──┬──────┬─filename──┬─────────────┬──────────────────────────────►◄
    └─SAV──┘  └─path─┘           ├─┬─CREATE─┬──┤
                                 │ └─CRE────┘  │
                                 ├─┬─REPLACE─┬─┤
                                 │ └─REP─────┘ │
                                 └─┬─APPEND─┬──┘
                                   └─APP────┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the file used. If no path is specified, the current directory is used.

filename
: Specifies the name of a file to which the buffer contents are written. If you do not provide a file extension, the `.sql` extension is used.

CREATE | CRE
: Creates the specified file if it does not exist.

REPLACE | REP
: Overwrites the specified file.

APPEND | APP
: Adds the contents of the SQL buffer to the end of the specified file.

SET

The **SET** CLPPlus command controls a session-level variable for the CLPPlus interface.

Important:
- For each invocation of the **SET** command, you can specify only one parameter.
- In a batch script, you can issue several **SET** commands in a series.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►─SET─┬─────────────────────────┬─┬──────────────┬─┬──────────────────┬─────────────────────►
       └─ARRAYSIZE─integer-value─┘ ├─AUTOCOMMIT─┤ ├─ON───────────────┤
                                   └─AUTO───────┘ ├─OFF──────────────┤
                                                  ├─OFFALL───────────┤
                                                  ├─IMMEDIATE────────┤
                                                  └─statement_count──┘

►─┬─────────────┬─┬──────────────┬─┬───────────┬─┬──────────────┬─┬────────────────────────┬───►
  ├─AUTOTRACE─┤ ├─ON───────────┤ ├─EXPLAIN─┤ ├─STATISTICS─┤ └─BGCOLOR─┬─color─┬─┘
  └─AUTOT─────┘ ├─OFF──────────┤ └─EXP─────┘ └─STAT───────┘           └─r,g,b─┘
                └─┬─TRACEONLY─┤
                  └─TRACE─────┘

►─┬──────────────────────┬─┬──────────────────────────┬─┬──────────────────────┬─┬─────────────────┬──►
  └─COLOR─┬─color─┬──────┘ └─COLSEP─column-separator─┘ └─DUPLICATES─┬─ON──┬─┘ └─ECHO─┬─ON──┬─┘
          └─r,g,b─┘                                                 └─OFF─┘          └─OFF─┘

►─┬───────────────────────────┬─┬───────────────────────────────────┬─┬──────────────────────┬──►
  └─ENCODING─encoding-format─┘ └─┬─ENVARSUBSTITUTION─┬─┬─ON──┬─┘ └─ESCAPE─┬─ON─────────┤
                                 └─ENVARSUBST────────┘ └─OFF─┘             ├─OFF────────┤
                                                                           └─character──┘

►─┬──────────────────────────────────┬─┬──────────────────────┬─┬─────────────────────────────┬──►
  └─┬─FEEDBACK─┬─┬─ON─────────────┤ └─┬─FLUSH─┬─┬─ON──┬─┘ └─FONT─name─,─style─,─size─┘
    └─FEED─────┘ ├─OFF────────────┤   └─FLU───┘ └─OFF─┘
                 └─row-threshold──┘

►─┬───────────────────────────┬─┬──────────────┬─┬──────────────────────────┬──────────────────►
  └─┬─HEADING─┬─┬─ON──┬──────┘ └─┬─HEADSEP─┬─┘ └─LOGMODE─logmode-value─┘
    ├─HEAD────┤ └─OFF─┘            └─HEADS───┘
    └─HEA─────┘

►─┬────────────────────────────────┬─┬──────────────────────────────┬─┬──────────────────────┬──►
  └─JCCLOGMODE─jcclogmode-value─┘ └─┬─LINESIZE─┬─width-of-line─┘ └─LOCALE─locale_name─┘
                                    └─LIN──────┘

►─┬────────────────────┬─┬──────────────────────┬─┬──────────────────────────────┬────────────►
  └─LOGPATH─log-path─┘ └─LONG─integer-value─┘ └─┬─NEWPAGE─┬─lines-per-page─┘
                                                 └─NEWP────┘

►─┬──────────────────────┬─┬───────────────────────────┬─┬───────────────────────────────┬────►
  └─NULL─null-string─┘ └─NUMFORMAT─format-string─┘ └─NUMWIDTH─┬─OFF────────────┬─┘
                                                              └─integer-value──┘

►─┬───────────────────────────────┬─┬──────────────────────────────┬─┬────────────────────────┬─►
  └─┬─PAGESIZE─┬─lines-per-page─┘ └─┬─SQLCASE─┬─┬─┬─MIXED─┬─┬─┘ └─┬─PAUSE─┬─┬─ON──┬─┘
    └─PAGES────┘                    └─SQLC────┘ │ └─MIX───┘ │       └─PAU───┘ └─OFF─┘
                                                ├─┬─UPPER─┬─┤
                                                │ └─UP────┘ │
                                                └─┬─LOWER─┬─┘
                                                  └─LO────┘

►─┬───────────────────────────────────────┬─┬──────────────────────┬─┬────────────────────────────┬─►
  └─PRESERVEWHITESPACE─┬─TRUE──┬─┘ └─RECSEP─┬─ON──┬─┘ └─RECSEPCHAR─character─┘
                       └─FALSE─┘                └─OFF─┘
```

```
►─┬─────────────────────────────────────────────────────────────────────────────────┬──────────────►
  └─SERVEROUTPUT──┬─ON──┬──┬──────────────────────────┬──┬──────────────────────────┬─┘
                  └─OFF─┘  │         ┌─UNLIMITED─┐    │  └─FORMAT──┬─WRAPPED──────┤
                           └─SIZE────┴─n─────────┴────┘            ├─WORD_WRAPPED─┤
                                                                   └─TRUNCATED────┘

►─┬─────────────────────────────┬─┬──────────────────────────┬─┬─────────────────────────┬──────────►
  ├─┬─SQLPROMPT─┬──prompt───────┘ └─┬─TERMOUT─┬──┬─ON──┬─────┘ └─┬─TIMING─┬──┬─ON──┬─────┘
    └─SQLP──────┘                   └─TERM────┘  └─OFF─┘           └─TIMI───┘  └─OFF─┘

►─┬──────────────────────────┬─┬────────────────────────────┬─┬──────────────────────┬─┬─────────────────────┬─►
  └─┬─TRIMOUT─┬──┬─ON──┬─────┘ └─┬─TRIMSPOOL─┬──┬─ON──┬─────┘ └─UNDERLINE──┬─ON──┬──┘ └─VERBOSE──┬─ON──┬───┘
    └─TRIMO───┘  └─OFF─┘           └─TRIMS─────┘  └─OFF─┘                   └─OFF─┘               └─OFF─┘

►─┬─────────────────────────┬─┬──────────────────┬──────────────────────────────────────────────────►◄
  └─┬─VERIFY─┬──┬─ON──┬─────┘ └─WRAP──┬─ON──┬────┘
    └─VER────┘  └─OFF─┘               └─OFF─┘
```

Command parameters

ARRAYSIZE ***integer-value***

Defines the number of rows that are fetched at a time from the server. You can use this parameter to tune query performance. Valid values are `1 - 10000`. The default value is `10`.

AUTOCOMMIT | AUTO

Controls the commit behavior of SQL statements in CLPPlus. CLPPlus always automatically commits DDL statements.

ON | IMMEDIATE

Enables automatic commitment of SQL statements.

OFF

Disables automatic commitment of SQL statements except for DDL statements.

OFFALL

Disables automatic commitment of SQL statements.

AUTOTRACE | AUTOT

Controls the display of explain plans and statistics information for SQL statements in a CLPPlus session.

The **AUTOTRACE** parameter is supported on Db2 for z/OS. The **AUTOTRACE** parameter is also supported on IBM Informix, with these restrictions:

- The `EXPLAIN` option is supported only for SELECT statements.
- If you specify the `EXPLAIN`, option, you must create and specify a default sbspace name for the **SBSPACENAME** configuration parameter in the `ONCONFIG` file. This sbspace name is used for creating BLOBs when an explain plan is created.
- To retrieve statistics from an Informix server, you must have the Informix privilege or an equivalent privilege.

ON Enables AUTOTRACE. If you set the **AUTOTRACE** parameter to `ON`, CLPPlus continues to display the explain information until the session ends or until you set the **AUTOTRACE** parameter to `OFF`.

OFF

Disables AUTOTRACE.

TRACEONLY | TRACE
Disables the display of query execution output.

EXPLAIN | EXP
Enables the display of the explain plan.

STATISTICS | STAT
Enables the display of the statistics for statements.

BGCOLOR *color|r,g,b*
Sets the background color in window mode.

color
Valid values for the *color* variable are as follows:

- `BLACK|black`
- `BLUE|blue`
- `CYAN|cyan`
- `DARK_GRAY|darkGray`
- `GRAY|gray`
- `GREEN|green`
- `LIGHT_GRAY|lightGray`
- `MAGENTA|magenta`
- `ORANGE|orange`
- `PINK|pink`
- `RED|red`
- `WHITE|white`
- `YELLOW|yellow`

r,g,b
Sets an opaque RGB color with the specified red, green, and blue values. The valid range for the red, green, and blue values is 0 - 255. Any invalid value is treated as 255.

COLOR *color|r,g,b*
Sets the font color in window mode.

color
Valid values for the *color* variable are as follows:

- `BLACK|black`
- `BLUE|blue`
- `CYAN|cyan`
- `DARK_GRAY|darkGray`
- `GRAY|gray`
- `GREEN|green`
- `LIGHT_GRAY|lightGray`
- `MAGENTA|magenta`
- `ORANGE|orange`
- `PINK|pink`
- `RED|red`
- `WHITE|white`
- `YELLOW|yellow`

r,g,b
Sets an opaque RGB color with the specified red, green, and blue values. The valid range for the red, green, and blue values is 0 - 255. Any invalid value is treated as 255.

COLSEP *column-separator*
Places the specified delimiter between columns in a table. The delimiter must be a character, which can be a special character or a space (the default value).

DUPLICATES
Controls the printing of duplicate column values for the break columns that you specify for the **BREAK** command.

ON Enables the printing of duplicate column values.

OFF
Disables the printing of duplicate column values.

ECHO
Controls whether all commands are displayed in the standard output of the CLPPlus interface.

ON Enables the display of commands.

OFF
Disables the display of commands.

ENCODING *encoding-format*
Controls the encoding format that is used in a CLPPlus session. You can set the encoding format in the window mode and non-window mode CLPPlus consoles. The default value is UTF-8.

The **SET ENCODING** command is available in Db2 Version 10.5 Fix Pack 5 and later fix packs.

When you set the encoding format in the non-window mode CLPPlus console, only the batch file that is read into the console and the output that is written to the spooled file use the specified encoding format. The non-window mode CLPPlus console might not process interactive command input or send output to standard output in the specified encoding.

The following list contains the valid encoding format values:

```
Big5, Big5-HKSCS, CESU-8, EUC-JP, EUC-KR, GB18030,
GB2312, GBK, hp-roman8, IBM-Thai, IBM00858,
IBM00924, IBM01140, IBM01141, IBM01142, IBM01143,
IBM01144, IBM01145, IBM01146, IBM01147, IBM01148,
IBM01149, IBM037, IBM1026, IBM1047, IBM273, IBM277,
IBM278, IBM280, IBM284, IBM285, IBM290, IBM297,
IBM420, IBM424, IBM437, IBM500, IBM775, IBM850,
IBM852, IBM855, IBM857, IBM860, IBM861, IBM862,
IBM863, IBM864, IBM865, IBM866, IBM868, IBM869,
IBM870, IBM871, IBM918, ISO-2022-CN, ISO-2022-JP,
ISO-2022-JP-2, ISO-2022-KR, ISO-8859-1, ISO-8859-10,
ISO-8859-13, ISO-8859-14, ISO-8859-15, ISO-8859-16,
ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5,
ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9,
JIS_X0201, JIS_X0212-1990, KOI8-R, KOI8-U, KZ-1048,
PTCP154, Shift_JIS, TIS-620, US-ASCII, UTF-16,
UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE,
UTF-8, windows-1250, windows-1251, windows-1252,
windows-1253, windows-1254, windows-1255,
windows-1256, windows-1257, windows-1258,
windows-31j, windows-874, x-Big5-HKSCS-2001,
x-Big5-Solaris, x-compound-text, x-EUC-TW,
x-EUC_CN, x-EUC_JP_LINUX, x-eucJP-Open, x-IBM-udcJP,
x-IBM1006, x-IBM1025, x-IBM1027, x-IBM1041,
x-IBM1043, x-IBM1046, x-IBM1046S, x-IBM1047_LF,
x-IBM1088, x-IBM1097, x-IBM1098, x-IBM1112,
x-IBM1114, x-IBM1115, x-IBM1122, x-IBM1123,
x-IBM1124, x-IBM1141_LF, x-IBM1153, x-IBM1166,
x-IBM1351, x-IBM1362, x-IBM1363, x-IBM1363C,
x-IBM1364, x-IBM1370, x-IBM1371, x-IBM1380,
x-IBM1381, x-IBM1382, x-IBM1383, x-IBM1385,
x-IBM1386, x-IBM1388, x-IBM1390, x-IBM1390A,
x-IBM1399, x-IBM1399A, x-IBM16684, x-IBM16684A,
x-IBM29626, x-IBM29626C, x-IBM300, x-IBM300A,
x-IBM301, x-IBM33722, x-IBM33722A, x-IBM33722C,
x-IBM420S, x-IBM4933, x-IBM720, x-IBM737,
x-IBM808, x-IBM833, x-IBM834, x-IBM835, x-IBM836,
x-IBM837, x-IBM856, x-IBM859, x-IBM864S, x-IBM867,
x-IBM874, x-IBM875, x-IBM897, x-IBM921, x-IBM922,
x-IBM924_LF, x-IBM927, x-IBM930, x-IBM930A, x-IBM933,
x-IBM935, x-IBM937, x-IBM939, x-IBM939A, x-IBM942,
x-IBM942C, x-IBM943, x-IBM943C, x-IBM947, x-IBM948,
x-IBM949, x-IBM949C, x-IBM950, x-IBM951, x-IBM954,
x-IBM954C, x-IBM964, x-IBM970, x-IBM971, x-ISCII91,
x-ISO-2022-CN-CNS, x-ISO-2022-CN-GB, x-iso-8859-11,
x-ISO-8859-6S, x-JIS0208, x-JISAutoDetect, x-Johab,
x-KOI8_RU, x-KSC5601, x-MacArabic,
x-MacCentralEurope, x-MacCroatian, x-MacCyrillic,
x-MacDingbat, x-MacGreek, x-MacHebrew, x-MacIceland,
x-MacRoman, x-MacRomania, x-MacSymbol, x-MacThai,
x-MacTurkish, x-MacUkraine, x-MS932_0213,
x-MS950-HKSCS, x-MS950-HKSCS-XP, x-mswin-936,
x-mswin-936A, x-PCK, x-SJIS_0213, x-UTF-16LE-BOM,
X-UTF-32BE-BOM, X-UTF-32LE-BOM, x-UTF_8J,
x-windows-1256S, x-windows-50220, x-windows-50221,
x-windows-949, x-windows-950, x-windows-iso2022jp
```

ENVVARSUBSTITUTION | ENVVARSUBST
Controls whether the CLPPlus interface supports environment variable substitution.

ON Enables environment variable substitution. This is the default value. CLPPlus treats all text that is prefixed by the dollar sign ($) character and text that is wrapped in percent sign (%) characters as environment variables and substitutes them with the associated values.

OFF
Disables environment variable substitution.

ESCAPE
Controls whether an escape character is set for use in the CLPPlus interface.

ON Enables the default escape character "\".

OFF
Disables the currently defined escape character.

character
Enables the escape character with the value of *character*.

FEEDBACK | FEED
Controls the display of interactive information after you issue an SQL statement.

ON Enables the display of interactive information. This is the default action.

OFF
Disables the display of interactive information.

row-threshold
Specifies the minimum number of rows that are required to enable feedback.

FLUSH| FLU
Controls whether the output buffer is accessible to external programs. The **FLUSH** parameter is still active while the output buffer is being appended to. The process creates extra processing requirements when you call statements or run commands, however.

ON Makes the buffer accessible to external programs.

OFF
Prevents the buffer from being available to external programs.

FONT
Sets the font name, style, and size in window mode.

name
Specifies the font name to set. Possible values are as follows:
- `monospaced`
- `sansserif`
- `serif`
- A valid system font name

style
Specifies the font style. Possible values are as follows:

0 Plain text.

1 Bold text

2 Italic text.

3 Bold and italic text.

size
Specifies the font size. The accepted value is an integer.

HEADING | HEA
Determines whether column headings are displayed for SELECT statements.

ON Enables the display of column headings.

OFF
Disables the display of column headings.

HEADSEP | HEADS
Sets the heading separator character that is used by the **COLUMN HEADING** command. The default character is a vertical bar (|).

LOGMODE *logmode-value*
Controls tracing and logging for the CLPPlus client layer and JDBC driver layer (JCC). You can specify one of the following values for the *logmode-value* variable:

CLPPLUS
Performs tracing and logging for the CLPPlus client layer only.

JCC
Performs tracing and logging for the JDBC client layer only.

BOTH
Performs tracing and logging for the CLPPlus client layer and JDBC client layer.

NONE
Disables all tracing and logging.

JCCLOGMODE *jcclogmode-value*
Specifies what JCC client layer features are traced, logged, or both. You can specify one of the following values for the *jcclogmode-value* variable. To specify a value for the *jcclogmode-value* variable, you must set the **LOGMODE** parameter to 1 or 2.

0 (TRACE_NONE)

1 (TRACE_CONNECTION_CALLS)

2 (TRACE_STATEMENT_CALLS)

4 (TRACE_RESULT_SET_CALLS)

16 (TRACE_DRIVER_CONFIGURATION)

32 (TRACE_CONNECTS)

64 (TRACE_DRDA_FLOWS)

128
(TRACE_RESULT_SET_META_DATA)

256
(TRACE_PARAMETER_META_DATA)

512
(TRACE_DIAGNOSTICS)

1024
(TRACE_SQLJ)

2048
(TRACE_XA_CALLS)

-1 (TRACE_ALL). By default, the -1 setting is used, meaning that all layers are traced.

LINESIZE | LIN *width-of-line*
Specifies the width of a line in characters. The valid range is 1 - 32767. The default value is 80.

LOCALE*locale_name*
Sets the name of the message locale to use in the CLPPlus environment.

LOGPATH*log-path*
Sets the path of a file that is used to keep log records of traces that use the settings of the LOGMODE and JCCLOGMODE parameters.

LONG *integer-value*
Defines the number of characters that are displayed for large text objects such as CLOB and XML. The default value is 50. The valid range is 1 - 2147483647.

NEWPAGE | NEWP *lines-per-page*
Controls how many blank lines are printed after a page break. The value is an integer of 0 - 100. By default, the value is 1, which indicates that one blank line is printed after a page break. A value of 0 causes a form feed to be printed at the start of each new page.

NULL ***null-string***
: Sets the string of characters that is displayed for a null value in a column in the output buffer. The string can include spaces and special characters. By default, the string is set to a space. The use of quotation marks around the string has no affect on its value. The case of letters is maintained.

NUMFORMAT ***format-string***
: Sets the default format string that is used for displaying numbers. The supported formats are the same as those for **COLUMN FORMAT** *format-string*.

NUMWIDTH
: Sets the default width that is used to display numbers. The default value is `OFF`.

PAGESIZE | PAGES ***lines_per_page***
: Sets the number of printed lines that fit on a page. The default is `25`. Valid values are `0` and `2 - 50000`.

PRESERVEWHITESPACE

TRUE
: Retains any indentation for all SQL and PL/SQL syntax and blocks. Spacing is retained in both window and non-window mode. This behavior applies whether input is read from a file or interactively in CLPPlus. `TRUE` is the default setting.

FALSE
: Trims spaces in all SQL and PL/SQL syntax and blocks.

SQLCASE | SQLC
: Controls whether the characters in SQL statements that are transmitted to the server are converted to uppercase or lowercase letters.

MIXED | MIX
: Specifies that a string of characters can contain uppercase and lowercase letters.

UPPER | UP
: Specifies that a string of characters can contain only uppercase letters.

LOWER | LO
: Specifies that a string of characters can contain only lowercase letters.

PAUSE | PAU
: Determines whether to stop a process before each page break. If the output cannot be displayed in one page, you are prompted with the message `Hit ENTER to continue` before each page break.

ON Pauses the display of output.

OFF
: Displays the output without pausing.

RECSEP
: Specifies whether the record-separating character that you set by using the **RECSEPCHAR** parameter is displayed after each record in the result set is printed.

ON Prints the record-separating character following each record in the result set.

OFF
: Does not print the record-separating character.

RECSEPCHAR *character*
: Specifies a single record-separating character that is used with the **RECSEP** parameter.

SERVEROUTPUT
: Specifies whether output messages from server-side procedures are retrieved and displayed on the client console.

  ON Specifies that server-side procedure output messages are retrieved and displayed.

  OFF
  Specifies that server-side procedure output messages are not retrieved and displayed.

  SIZE
  Specifies the number of characters that are displayed on the screen. Possible values are as follows:

    UNLIMITED
    The default value.

    n Specifies a particular number of characters, where *n* must be a positive integer.

  FORMAT
  Specifies the format style that is used to display server output in the console.

    TRUNCATED
    Truncates text that exceeds the line size.

    WORD_WRAPPED
    Enables text to overflow to the next line and does not split words across lines.

    WRAPPED
    Enables text to overflow to the next line as needed.

SQLPROMPT | SQLP *prompt*
: Specifies the prompt in the CLPPlus interface. By default, the prompt is SQL>. The prompt must be a string, which can include special characters and spaces. The use of quotation marks around the string has no affect on its value; the case of letters is maintained.

TERMOUT | TERM
: Determines whether output is displayed in the standard output of the CLPPlus interface.

  ON Displays the output on the screen.

  OFF
  Does not display output.

TIMING | TIMI
: Controls whether elapsed time is displayed for each SQL statement after it is issued.

  ON Specifies that elapsed time is displayed.

  OFF
  Specifies that elapsed time is not displayed.

TRIMOUT | TRIMO
Controls whether trailing blank spaces are removed from the output before it is written to the console.

ON Specifies that trailing blank spaces are removed.

OFF
Specifies that trailing blank spaces are not removed. This is the default.

TRIMSPOOL | TRIMS
Controls whether trailing blank spaces are removed from the spool output before it is written to the spool file.

ON Specifies that trailing blank spaces are removed.

OFF
Specifies that trailing blank spaces are not removed. This is the default.

UNDERLINE
Specifies whether column headings are underlined.

ON Specifies that column headings are underlined.

OFF
Specifies that column headings are not underlined.

USECURRENTDIRLOGPATH
Controls the value of LOGPATH.

ON LOGPATH is set to <CurrentDirectory>/clpplus.log unless the user has explicitly set the value of LOGPATH.

OFF
Has no effect on LOGPATH.

VERBOSE
Determines whether all CLPPlus messages are printed to the console.

ON Specifies that all CLPPlus messages are printed to the console.

OFF
Specifies that only a subset of messages is printed to the console. This is the default value.

VERIFY | VER
Determines whether the old and new values of an SQL statement are displayed when a substitution variable is encountered.

ON Specifies that the old and new values are displayed.

OFF
Specifies that the old and new values are not displayed.

WRAP
Sets the default alignment that is used when displaying column values.

ON Specifies that column values that exceed the column width are wrapped.

OFF
Specifies that column values that exceed the column width are truncated.

SPOOL

You can use the **SPOOL** CLPPlus command to log CLPPlus command and its output to a file. The output file uses UTF-8 encoding.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
►►──┬─SPOOL─┬──┬──────┬──┬─────────────┬─────────────────────────────►◄
    └─SP────┘  └─path─┘  ├─output_file─┤
                         └─OFF─────────┘
```

Command parameters

path
: Specifies the path, either absolute or relative, to the output file. If you do not specify a path, the current directory is used.

output_file
: Causes the CLPPlus command and its output to be logged in the file that is specified by the *output_file* variable instead of being displayed in the CLPPlus standard output.

OFF
: Causes the CLPPlus command and its output to be displayed in the CLPPLus standard output, which is the default behavior.

SHOW

The **SHOW** CLPPlus command displays the current settings of session-level variables in the CLPPlus interface or errors returned from server-side procedures. The settings are controlled using the **SET** command.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
>>--+-SHOW-+--{--+------------------------+--}-------------------------------><
    '-SHO--'     +-setting----------------+
                 +-ALL--------------------+
                 +-ERRORS-----------------+
                 '----procedure-name------'
```

Command parameters

setting
: Displays the name and setting of the specified session-level variable.

ALL
: Displays the names and settings of all session-level variables.

ERRORS
: Displays the errors for all server-side procedures run in the current CLPPlus session.

procedure-name
: When appended to the **SHOW ERRORS** command, shows only the errors for *procedure-name*.

START

The **START** CLPPlus command runs a CLPPlus script file.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
>>--+-START-+--+--------+--script-file-------------------------------------><
    '-STA---'  +-path---+
               '-URL----'
```

Command parameters

path
: Specifies the path, either absolute or relative, to the script file that contains SQL statements and commands to run. If no path is specified, the current directory is used.

URL
: Specifies the URL to the script file that contains SQL statements and commands to run. The URL must start with `http://` or `https://`.

script-file
: Specifies the script file name that contains SQL statements and commands to run.

Example

This example shows CLPPlus starting a script named demo.sql found on the computer with IP address 9.124.159.144.

```
SQL> Start http://9.124.159.144/demo.sql
```

TTITLE

The **TTITLE** CLPPlus command inserts text at the top of each page displayed.

Invocation

You must run this command from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
                ┌─,──────────────────────────────────────┐
                ▼  ┌─CENTER─┐                            │
>>──TTITLE──────┴──┼─LEFT───┼──text──┬──────────────────┬─┴──┬─────────────────────┬──────><
                   └─RIGHT──┘        ├─PGNO─────────────┤    └─SKIP─integer-value──┘
                                     └─variable-name────┘
```

Command parameters

text
: Specifies the text to be displayed.

CENTER
: Specifies the display will center justify the text on each page. If neither **CENTER**, **LEFT**, or **RIGHT** is specified, center justification is the default behavior.

LEFT
: Specifies the display will left justify the text on each page.

RIGHT
: Specifies the display will right justify the text on each page.

PGNO
: Specifies the current page number.

variable-name
: Specifies a user defined variable that will follow the *text* field.

SKIP *integer-value*
: The *integer-value* value specifies the number of blank lines displayed after the top title.

Example

In the following example, the `DEPT:` (with the variable contents), `CONFIDENTIAL`, and `Page No:` (with the current page number) is displayed across the top of every page. One blank line follows the top title.

```
SQL> BREAK ON workdept SKIP PAGE;
SQL> COLUMN workdept NEW_VALUE new_dept;
SQL> TTITLE LEFT 'DEPT: ' new_dept, CENTER 'CONFIDENTIAL', RIGHT 'Page No: ' PGNO SKIP 1;
```

In the following example, the `Page No:` title (with the current page number) is displayed across the top of every page with right justification. Two blank lines follow the top title.

```
SQL> TTITLE RIGHT 'Page No: ' PGNO SKIP 2;
```

UNDEFINE

The **UNDEFINE** CLPPlus command clears and deletes a variable created by the **DEFINE** CLPPlus command.

Invocation

You must run this command from the CLPPlus interface.

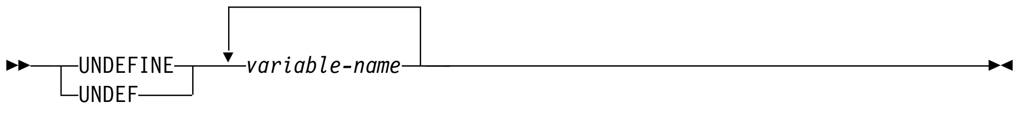
Authorization

None

Required connection

You must be connected to a database.

Command syntax



Command parameters

variable-name
: Specifies the name of the variable to clear and delete.

WHENEVER OSERROR

The **WHENEVER OSERROR** CLPPlus command specifies the action the CLPPlus interface performs when an operating system error occurs. You can use this command to trap errors and control the CLPPlus interface behavior by performing specified actions like **EXIT** or **CONTINUE**.

Invocation

The **WHENEVER OSERROR** command must be issued from the CLPPlus interface.

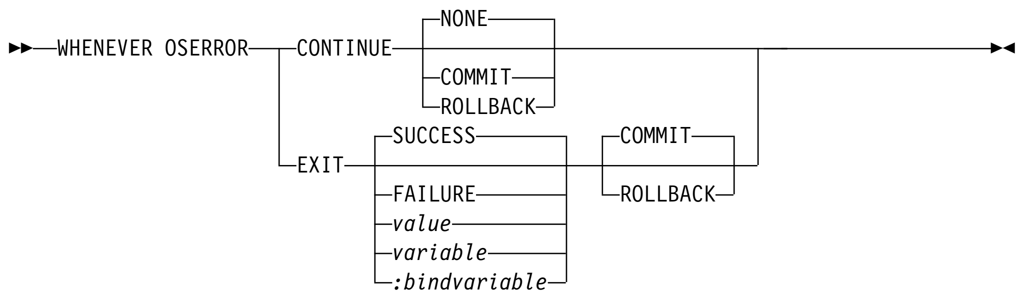
Authorization

None

Required connection

None

Command syntax



Command parameters

CONTINUE
: Directs the CLPPlus interface to continue with a specified action when an SQL or PL/SQL error is encountered.

NONE
: The default value that is used in the **WHENEVER OSERROR CONTINUE** command. No action on the block of SQL generating an error is taken.

COMMIT
: When **COMMIT** is specified in the **WHENEVER OSERROR CONTINUE** command, any possible work that is done by the current SQL block is committed.

ROLLBACK
: When **ROLLBACK** is specified in the **WHENEVER OSERROR CONTINUE** command, all work in the current SQL block is rolled back.

EXIT
: Directs the CLPPlus interface to exit once an operating system error is encountered. The functionality of this option is the same as the stand-alone **EXIT** command.

SUCCESS
: Returns an operating system-dependant return code that indicates success. This is the first default **EXIT** parameter.

FAILURE
: Returns an operating system-dependant return code that indicates a failure.

value
: Specifies a variable that is created by the **DEFINE** command whose value is returned as the return code.

variable
: Specifies a substitution variable value that is created by the **DEFINE** command whose value is returned as the return code.

:bindvariable
: Specifies a Bind variable value that is created by the **DEFINE** command whose value is returned as the return code.

COMMIT
Specifies that uncommitted updates are committed when the CLPPlus session ends. This is the second default **EXIT** parameter.

ROLLBACK
Specifies that uncommitted updates are rolled back when the CLPPlus session ends.

Examples

The following example shows the command behavior when **EXIT** and an exit error value are specified.

```
SQL> whenever oserror exit -1
SQL> get c:\nonexistingfile.sql
DB250204E: An attempt to locate a file 'c:\\nonexistingfile.sql' failed. The co
mmand cannot be processed.
```

You can review the return code for by running the **echo %errorlevel%** command.

```
echo %errorlevel%
-1
```

The following example shows the command behavior when **CONTINUE** is specified.

```
SQL> whenever oserror continue
SQL> get c:\nonexistingfile.sql
DB250204E: An attempt to locate a file 'c:\\nonexistingfile.sql' failed. The co
mmand cannot be processed.
SQL>
```

WHENEVER SQLERROR

The **WHENEVER SQLERROR** CLPPlus command specifies the action CLPPlus performs when an SQL error occurs in SQL or PL/SQL. This command allows you to trap errors and control CLPPlus behavior by performing specified actions like **EXIT** or **CONTINUE**.

Invocation

This command must be executed from the CLPPlus interface.

Authorization

None

Required connection

None

Command syntax

```
                                  ┌─NONE─────┐
►►──WHENEVER SQLERROR──┬─CONTINUE─┼──────────┼───────────────────────┬──────────►◄
                       │          ├─COMMIT───┤                       │
                       │          └─ROLLBACK─┘                       │
                       │      ┌─SUCCESS───────┐  ┌─COMMIT───┐        │
                       └─EXIT─┼───────────────┼──┼──────────┼────────┘
                              ├─FAILURE───────┤  └─ROLLBACK─┘
                              ├─WARNING───────┤
                              ├─value─────────┤
                              ├─variable──────┤
                              └─:bindvariable─┘
```

Command parameters

CONTINUE
Directs CLPPlus to continue with a specified action when an SQL or PL/SQL error is encountered.

NONE
The default value used in the **WHENEVER SQLERROR CONTINUE** command. No action on the block of SQL generating an error is taken.

COMMIT
When **COMMIT** is specified in the **WHENEVER SQLERROR CONTINUE** command, any possible work done by the current SQL block is committed.

ROLLBACK
When **ROLLBACK** is specified in the **WHENEVER SQLERROR CONTINUE** command, all work in the current SQL block is rolled back.

EXIT
Directs CLPPlus to exit once an SQL or PL/SQL error is encountered. The functionality of this option is the same as the stand-alone **EXIT** command.

SUCCESS
Returns an operating system-dependant return code indicating success. The is the first default **EXIT** parameter.

FAILURE
Returns an operating system-dependant return code indicating failure.

WARNING
Returns an operating system-dependant return code indicating a warning.

value
Specifies a variable created by the **DEFINE** command whose value is returned as the return code.

variable
Specifies a substitution variable value created by the **DEFINE** command whose value is returned as the return code.

:bindvariable
Specifies a Bind variable value created by the **DEFINE** command whose value is returned as the return code.

COMMIT
Specifies that uncommitted updates are committed when the CLPPlus session ends. The is the second default **EXIT** parameter.

ROLLBACK
Specifies that uncommitted updates are rolled back when the CLPPlus session ends.

Examples

The following example shows the **WHENEVER SQLERROR CONTINUE** command behavior. The CLPPlus prompt is returned and CLPPlus is still available for use.

```
SQL> whenever sqlerror continue
SQL> select * from nonexistingtable;
SQL0204N "SCHEMA.NONEXISTINGTABLE" is an undefined name.
SQL>
```

You can also commit, rollback, or take no action whenever an SQL error occurs.

```
SQL> whenever sqlerror continue commit
SQL>
```

```
SQL> whenever sqlerror continue rollback
SQL>
```

```
SQL> whenever sqlerror continue none
SQL>
```

The following examples use the **EXIT** option to exit the CLPPlus application.

```
SQL> whenever sqlerror exit
SQL> select * from nonexistingtable;
SQL0204N "SCHEMA.NONEXISTINGTABLE" is an undefined name.

C:\>
```

The following specify the error code returned during exit. This behavior is identical to the **EXIT** CLPPlus command.

```
SQL> whenever sqlerror exit success
SQL> whenever sqlerror exit failure
SQL> select * from nonexistingtable;
SQL0204N "SCHEMA.NONEXISTINGTABLE" is an undefined name.

C:\echo %errorlevel%
1
```

```
SQL> define exit_value=6
SQL> whenever sqlerror exit exit_value
SQL> select * from nonexistingtable;
SQL0204N "SCHEMA.NONEXISTINGTABLE" is an undefined name.

C:\echo %errorlevel%
6
```

Similar to the EXIT CLPPlus command, you can specify whether to commit or rollback while exiting the CLPPlus application.

```
SQL> whenever sqlerror exit 2 commit

SQL> whenever sqlerror exit 2 rollback
```

Index

Printed in USA