Dialect Database Gateway 5.4a

Programmer's Reference Guide

Williams℠ COMMUNICATIONS

Dialect Database Gateway 5.4a Programmer's Reference Guide

Part No.: DGG-360769-34PG
Product: Dialect Database Gateway 5.4a
Document: Programmer's Reference Guide
Date: April 2000
Revised for InterVoice IVR Clients

Trademarks

Williams Communications Solutions, LLC has made every effort to supply trademark information about company names, products and services mentioned in this document.

- Dialect is a trademark of Williams Communications Solutions, LLC.
- Nortel, Meridian, and Symposium are trademarks of Northern Telecomm.
- Intel and Pentium are registered trademarks of Intel Corporation.
- OS/2 is a trademark of IBM Corporation.
- SCO is a registered trademark of The Santa Cruz Operation, Inc.
- Solaris and Java are trademarks of SunSoft Microsystems, Inc.
- Windows, Windows NT, Visual Basic, ActiveX, and Microsoft are either trademarks or registered trademarks of Microsoft Corporation.
- UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

All other brand and product names are trademarks or registered trademarks of their respective companies.

Notice

This document, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of this license or a nondisclosure agreement. Williams Communications Solutions, LLC, makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties, salability, merchantability or fitness for a particular purpose. Williams Communications Solutions, LLC, also reserves the right to revise the software and this document and make changes to its content, at any time, without obligation to notify any person or entity of such revision or changes.

Table of Contents

Introduction

Welcome to DBG

Welcome to Dialect Database Gateway or DBG, a powerful client/server gateway system providing transparent, cross-platform access to multiple database engines.

With support for 32-bit Microsoft® Windows® platforms and SCO® UNIX® systems, DBG simplifies access to data and provides split-second system response to ensure that calling applications receive replies to query requests in the event of network or database failures.

Open System Architecture

Dialect Database Gateway hosts two components: a server module and a client module.

The DBG Server (Server) is a Windows NT® service that communicates SQL (Structure Query Language) statements directly to database engines that follow the Open Database Connectivity (ODBC) standard.

Client applications running on Windows platforms and SCO UNIX use the DBG Application Programming Interface (API) to send and receive requests for information through the DBG Server. The DBG API can also (separately) support OS/2®, Solaris™, Java™, and ActiveX® clients.

Channel Requests Using Named Pipes

DBG makes it easy for client applications to access information by using "pipes" to channel requests and connect to specific databases.

A pipe is a logical, named entity that connects to a database engine (such as Oracle, Sybase, Microsoft Access, DB/2, etc.) via a single ODBC driver, provided by the vendor of the database. You assign pipe names and multiple pipe connections using the DBG Configuration Tool. DBG takes care of the connectivity to the database and effectively insulates calling applications from connectivity details. In addition, if the pipe was defined with multiple connections (logins), DBG can also execute multiple requests to the same pipe, in parallel.

Client applications identify the pipe name and construct a query statement that is passed to the DBG Server via an API call. Along with the statement, the application passes a pipe name to which the statement is to be sent. The Server software then directs the statement to the named pipe, which is processed by the vendor's ODBC driver residing at the other end of the pipe. (See Figure 1.)

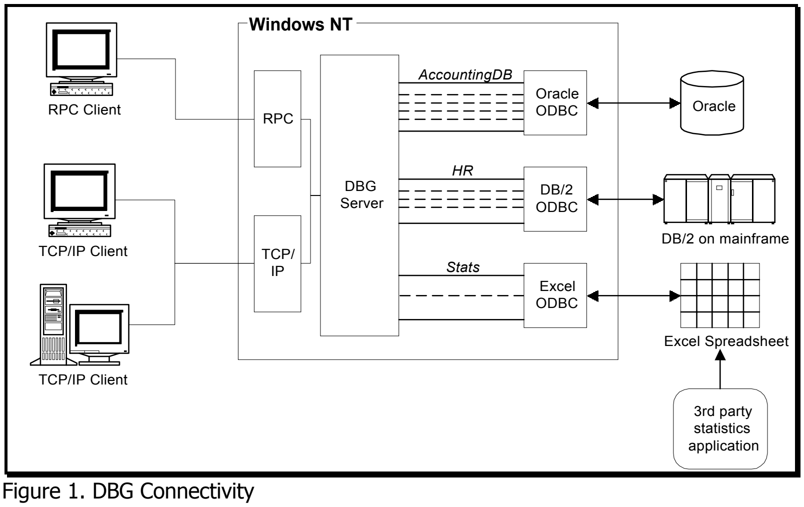


Figure 1. DBG Connectivity

Database Support

DBG was designed to support any database that provides a 32-bit ODBC driver for Windows NT.

Automatic Failure Recovery

In the event of a network and/or database failure, DBG continuously attempts to reconnect to its databases. While disconnected, DBG automatically queues requests from applications until connection (to the named pipe) is restored.

Background Processing

DBG's powerful background processing capabilities include waiting for a pipe to deliver information (if the statement requires records to be returned), even in the event of a network or database failure. If either one fails, DBG writes all transient data to disk, reads all transient data into memory, and then submits the statement for execution again when the network or database returns.

DBG also includes facilities for limiting the number of records returned from a database, even if the database itself does not support this feature.

About This Guide

This *Dialect Database Gateway 5.4a Programmer's Reference Guide* is written for developers who write applications that need simpler access to one or more database engines. It contains configuration information for DBG and detailed descriptions of the system's API routines. This *Programmer's Reference Guide* also provides guidelines for using the DBG API routines and programming examples.

Organization

Besides this "**Introduction,**" this *Programmer's Reference Guide* contains the following chapters:

- "**Installation and Distribution**"—describes the requirements for DBG, how to install the system's Server and client software components, and identifies the distribution files required for running an application that uses the DBG API.
- "**DBG Server Administration**"—describes how to configure and maintain the DBG Server.
- "**Client API Programming**"—describes the DBG API function calls and provides examples in Visual Basic® and Visual C++™.
- "**Appendix A. Data Structures and Constants**"—describes common data structures and constants for the system.
- "**Appendix B. Condition Values**"—lists possible condition values and errors returned by the system.
- "**Appendix C. The mivrdbg User Function**"—describes the user function and its function calls for SCO UNIX clients.
- "**Appendix D. DBG UserDLL for InterVoice IVR**"—describes the user function for InterVoice IVR and its function calls for InterVoice IVR clients.
- "**Appendix E. Licensing**"—describes the software licensing mechanism used by Dialect Database Gateway.

Assumptions

This *Programmer's Reference Guide* assumes that its readers are familiar with application development tools, such as Visual Basic® or C++®.

Conventions Used

This *Programmer's Reference Guide* uses the following conventions:

- Parameters, settings, and variables look like this: *callID*.

- Literals, error codes, and source code examples look like this: `callID`, **`DBGCConnect`**.
- Referenced properties, events, and methods look like this: **MakeCall**.
- Words used for emphasis and Windows conventions look like this: **OK**, **Start**.

Technical Support

Technical support for Dialect Database Gateway is pursuant to your contract or purchase agreement with Williams Communications Solutions. Please use the section below to record your contract or purchase agreement information.

Contract No.: ______________________________

Date Purchased: ______________________________

Sales Representative: ______________________________

Telephone: ______________________________

License Key Number: ______________________________

Installation and Distribution

This chapter describes the:

- Dialect Database Gateway box contents.
- Licensing services offered by the Williams' License Manager.
- Requirements for installing DBG.
- Installation procedures for the DBG Server and the DBG client APIs.
- Required distribution files for an application using the DBG API.

Database Gateway Box Contents

The Database Gateway box contains:

- A Dialect Suite CD-ROM.
- A hardware license key (also known as a dongle), or a diskette containing the new hardware license key image.
- *License Manager Installation Guide*
- *Dialect Database Gateway 5.4a Programmer's Reference Guide* (this manual).

The Dialect Suite CD-ROM contains the Database Gateway Server software and installation kits for the Database Gateway API software. It also contains this guide in Adobe Acrobat Portable Document Format (PDF) and a text file, README.TXT, which covers compatibility issues, late-breaking news, usage tips, and information about Database Gateway.

Requirements

This section describes what you need to install Database Gateway, such as installation prerequisites, hardware and software requirements, and related information.

Installation Prerequisites

Before installing Database Gateway, make sure that you read and understand the following requirements:

- If you have a hardware license key, you must install it, along with the Williams License Manager software ***before*** you install Database Gateway. For hardware license key and License Manager installation instructions, refer to the *License Manager Installation Guide*.
- If you do not have a hardware license key, want to evaluate the software, or plan to use a temporary license keycode, contact your Williams Communications Support representative.

Hardware and Software Requirements

Make sure you have the following requirements:

Component	Requirements
Server	• Processor and memory as required by the 32-bit Windows operating system installed on the computer. • Windows NT Server 4.0 with Service Pack 4 or higher. • 20 MB free disk space (minimum installation). • 32 MB of RAM (minimum installation). • ODBC Level 2 (or higher).

Client API	• Processor and memory as required by the 32-bit operating system installed on the computer. • Windows 95/98, Windows NT 4.0, or SCO UNIX. (If you're interested in integrating DBG software with other clients, such as Windows 3.11, Windows for Workgroups, OS/2, Java, Solaris, and so on, contact your Williams representative.) • A BSD (Berkeley Software Distribution) Sockets compliant TCP/IP (Transmission Control Protocol/Internet Protocol) stack on all platforms. (DBG supports WinSock (Windows Sockets) and WinSock2 on capable Windows platforms.)

Relevant Information

In addition to the standard TCP/IP interface, DBG supports Remote Procedure Calls (RPC) on Win32® (Windows NT, Windows 95/98). Support also extends to local DBG clients running on the same computer as the DBG Server. (The DBG client makes use of TCP/IP or Named Pipes as the wire-line protocol for RPC.)

On SCO UNIX for Meridian™ OPEN and Integrated IVR 2.0 and Symposium™ OPEN IVR 4.0, access to the DBG Server is available through an interface (mivrdbg) implemented as a user function. (See "Appendix C. The mivrdbg User Function" for details.)

Installing Database Gateway

This section describes how to install both the server software and the client API for DBG.

Installing the DBG Server

Make sure that you have met the hardware and software requirements before installing the DBG Server. Also, make sure that you know the name of the computer on which you installed the License Manager and the network protocol that the DBG Server will be using to connect to that computer.

> **Note:** You need not know the name of the computer on which you installed the License Manager and the network protocol if you plan to use a temporary license keycode.

▶ **To install the DBG Server:**

1. Run Windows NT and login as an administrator.

   You must have administrative privileges to install the DBG Server software. If you do not, please contact your system administrator.

2. Choose from the following:

   - If you're installing from compact disc, insert the Dialect CD-ROM in the CD-ROM drive. If the Williams' Dialect Software Suite web page opens in your browser, select **Database Gateway 5.4a** under "Dialect Suite," and then select **Setup**. When the File Download dialog box displays, select "Open it" or "Run the program from its current location," click **OK** and then proceed to step 5.

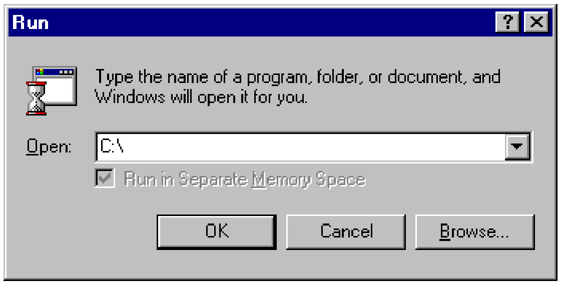
     If the Williams' Dialect Software Suite web page does not open in your browser, proceed to step 3.

   - If you're installing from a 3.5-inch disk, insert the DBG Server disk into drive A or B.

   - If you're installing from a single directory, ensure that all the installation files are in the same directory.

3. Click the **Start** button and then click **Run**.

   The Run dialog box displays.



Run
?
Type the name of a program, folder, or document, and Windows will open it for you.
Open:
C:\
☑ Run in Separate Memory Space
OK
Cancel
Browse...

Figure 2. Run dialog box

4. Depending on the type of medium you're using, choose from the following options:

   - If you're installing from CD-ROM, type `E:\DBGATEWY\SERVER\INSTALL\SETUP.EXE` (where `E` represents the drive letter of your CD-ROM) and click **OK**.
   - If you're installing from 3.5-inch disk, type `A:\SETUP.EXE` (where `A` represents the drive letter of your floppy disk drive) and click **OK**.
   - If you're installing from a single directory, type `A:\DDD\SETUP` (where `A` represents the drive letter, and `DDD` is the name of the directory where the installation files are located) and click **OK**.

5. Follow the on-screen instructions.

   The setup program installs and configures the DBG Server as a Windows NT Service (background process). The DBG (service) will automatically start each time the operating system reboots.

   > **Note:** The setup program's "Welcome" screen now includes options for installing the Data Repository Software Development Kit (SDK) and an extension module for InterVoice-Brite. For more information about the latter option, read "Appendix D. DBG UserDLL for InterVoice IVR" beginning on page 129.

Installing the DBG API

This section describes how to install the DBG client API on 32-bit Windows platforms and on SCO UNIX. For the latter, and Meridian™ OPEN IVR, Meridian Integrated IVR 2.0, and Symposium™ OPEN IVR 4.0, you must use the UNIX tar utility to copy and extract the client API onto the IVR. The client API is archived in a compressed format, which can either be downloaded or installed from a 3.5-inch floppy disk.

For installation information in other environments, please contact your Williams Communications Solutions representative.

▶ **To install the DBG APIs on Windows platforms:**

1. Run Windows.
2. Choose from the following:
   - If you're installing from compact disc, insert the Dialect CD-ROM in the CD-ROM drive. If the Williams' Dialect Software Suite web page opens in your browser, select **Database Gateway 5.4a** under "Dialect Suite." Next, under "Installation," click **Win32 Client Setup**. When the File Download dialog box displays, select "Open it" or "Run the program from its current location," click **OK**, and proceed to step 5.

     If the Williams' Dialect Software Suite web page does not open in your browser, proceed to step 3.
   - If you're installing the client APIs from a 3.5-inch disk, insert the appropriate DBG client API disk into drive A or B.
   - If you're installing from a single directory, ensure that all the installation files are in the same directory.

3. In Windows 95/98 and Windows NT 4.0, click the **Start** button, and then click **Run**. (Windows 95/98 and Windows NT 4.0 automatically launch the installation routine; skip to step 5 now.)

   The Run dialog box displays. (See Figure 2 on page 11.)

4. Depending on the type of medium you're using, choose from the following options:

   - If you're installing from the CD-ROM, select the subdirectory for the client platform on which you will be installing the client API, and then type: `E:\DBGATEWY\CLIENT\INSTALL\SETUP.EXE` (where `E` represents the drive letter of your CD-ROM, `CLIENT` represents the client platform, and `INSTALL` represents the subdirectory in which the client executable resides) and click **OK**.
   - If you're installing from a 3.5-inch client API disk, type `A:\SETUP.EXE` (where `A` represents the drive letter of your floppy disk drive) and click **OK**.
   - If you're installing from a single directory, type `C:\DDD\SETUP.EXE` (where `C` represents the drive letter, and `DDD` is the name of the directory where the installation files are located) and choose **OK.**

5. Follow the on-screen instructions.

▶ To install the client APIs on SCO UNIX for the Meridian Integrated IVR 2.0, or Meridian OPEN IVR 2.0, or Symposium OPEN IVR 4.0:

1. Run SCO UNIX and log in for the appropriate IVR:

   - For a Symposium OPEN or Meridian OPEN IVR, login as `nortel`.
   - For a Meridian Integrated IVR, login as `vad`.

   For more information about these logins, contact your system administrator or refer to your SCO UNIX documentation.

2. Check whether the appropriate .tar file already exists on the IVR's hard drive. Type:

   ```
   find / -name xxx_dg54.tar.Z -print
   ```

   where `xxx_dg54.tar.Z` represents one of the following tar files, and then press **Enter**:

   - `mo2_dg54.tar.Z` for the Meridian Open IVR 2.x
   - `mi2_dg54.tar.Z` for the Meridian Integrated IVR 2.x
   - `so4_dg54.tar.Z` for the Symposium Open IVR 4.0

   If the tar file exists, you need to decompress and extract the client API files; skip to step 5. If the tar file does not exist on the IVR's hard drive, continue with step 3.

3. Insert the 3.5 inch DBG client API disk into the floppy drive.

4. Copy the appropriate `xxx_dg54.tar.Z` file into the correct location. Type:

   ```
   tar –xvf /dev/rfd0135ds18 xxx_dg54.tar.Z
   ```

   where `xxx_dg54.tar.Z` represents the compressed tar file for the appropriate IVR and press **Enter**.

5. Decompress the tar file. Type:

   ```
   compress –d xxx_dg54.tar.Z
   ```

   where `xxx_dg54.tar.Z` represents the compressed tar file, and press **Enter**.

6. Extract the DBG client API files. Type:

   ```
   tar -xvf xxx_dg54.tar
   ```

   where `xxx_dg54.tar` represents the tar file, and press **Enter**.

7. Remove the 3.5-inch DBG client API disk from the drive.

8. Create the mivrdbg.ini file to identify the Windows NT server hosting DBG (see "Configuration" on page 109).

Application Distribution

Applications developed in association with the DBG APIs require certain C run-time .DLL files to ensure that they work properly. Under the terms of your DBG software license agreement, you may reproduce and distribute the C run-time .DLL file shown in Table 1.

> **Note:** You may not distribute DBGS.LIC (the DBG license file).

Table 1. Distribution File

Platform	Filename
32-bit Windows	DBGAPI32.DLL

Distributed applications also require a separate Williams License Manager and a hardware key to properly function. (See "Appendix E. Licensing" on page 149 for more information.)

DBG Server Administration

Part of installing DBG is configuring the Server component and maintaining its configuration. This chapter introduces the DBG Configuration Tool (Configuration Tool), which was automatically installed when you installed the Server software; it will help you manage the DBG Server (service). This chapter also explains how to:

- Start and close the Configuration Tool.
- Configure service, licensing, pipe, and data handling options for the DBG service and start, stop, and pause the service.

Working with the Configuration Tool

The default settings for the DBG Server are taken from the Windows NT Registry.

After installing DBG, there are several configuration steps to be performed before the DBG Server is ready to provide transparent, cross-platform access to multiple database engines. You can use the Configuration Tool to perform all configuration tasks, including:

- Administer the service on a local or networked computer.
- Start, stop, and modify the service.
- Designate tracing options to aid in debugging.
- Enter keycode options to validate each license of the software and to ensure technical support.
- Define and modify pipe information to external databases.

- Define error handling instructions for broken pipes that may be encountered by the Server.

Configuration Tool Prerequisites

Before using the Configuration Tool, make sure that you:

- Have administrative privileges on the host machine on which the DBG Server software resides. (See your system administrator to determine your privileges.)
- Are familiar with 32-bit ODBC Level-2 (or higher) drivers for Windows NT and vendor/driver-specific configurations.
- Know the user IDs and passwords to the databases that DBG is to access.
- Are familiar with creating SQL statements.
- Are familiar with Domain Name Services (DNS) protocol and system data sources.

Starting and Closing the DBG Configuration Tool

This section describes how to start and close the Configuration Tool.

▶ **To start the Configuration Tool:**

1. Run Windows and login as an administrator.

   You must have administrative privileges to use the DBG Configuration Tool. If you do not, please contact your system administrator.

2. In Windows NT 4.0, click the **Start** button, and then click **Programs | Dialect | Database Gateway Server | Configuration**.

   The DBG Configuration dialog box displays. (See Figure 3.)

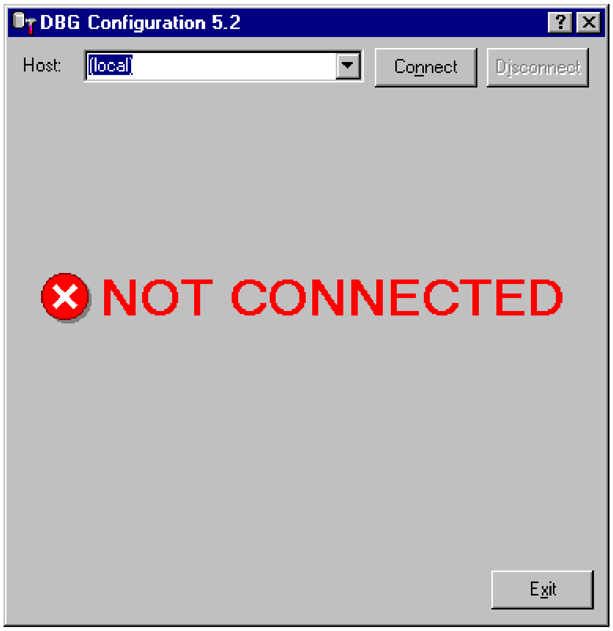


Figure 3. DBG Configuration dialog box (upon startup)

▶ To close the Configuration Tool:

- Click **Exit.**

  or

- Press **Alt**+**F4.**

  or

- Click the application's control-menu box.

Establishing a Connection to the Server

Before you can configure the DBG service for your organization's needs, you must establish a connection to the Server.

▶ To establish a connection to the DBG Server:

1. Start the Configuration Tool (as described in "Starting and Closing the DBG Configuration Tool" on page 18).

   The DBG Configuration dialog box displays. (See Figure 3 on page 19.) The bold caption "Not Connected" indicates that you have yet to connect to the DBG Server. This caption disappears from view once that connection is made.

2. In the **Host** box, enter the name of the computer on which the DBG Server software resides or select it from the drop-down list.

   The default is (local)—the local computer. The Host drop-down list will only contain host names if a previous session successfully connected to a host.

Even if the host is invalid or currently unavailable, the Configuration Tool will attempt to establish a connection to the remote computer.

3. Click **Connect** to establish a session.

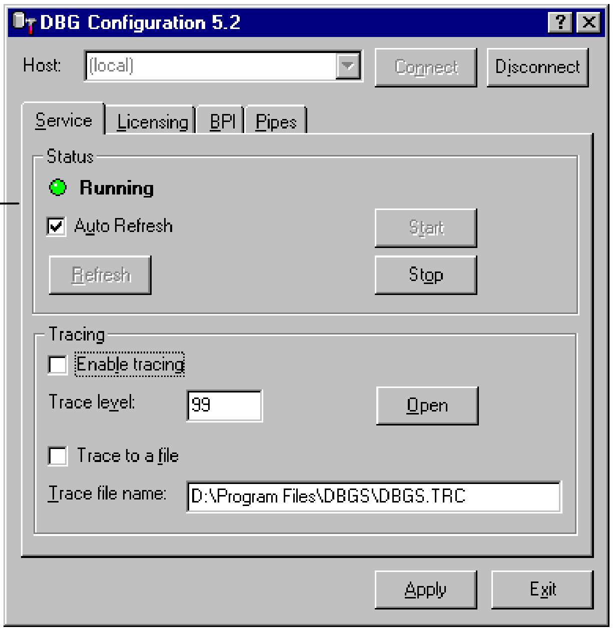
   The Configuration Tool attempts to establish a connection to the host. Once connected, the **Connect** button becomes disabled, and the **Disconnect** button becomes enabled. The **Disconnect** button can be used to disconnect the current host, and allow you to choose a new host. The DBG Configuration dialog box also opens revealing its four tabs:

   - Service tab—used to identify the status of the DBG Server, start, pause, or stop the service, and define tracing preferences. (See Figure 4 on page 21.)
   - Licensing tab—used to enter the License Server name and/or keycode used to ensure the licensing, security, and support for DBG. (See Figure 5 on page 27.)

- Pipes tab—used to define the physical connection between the DBG Server and the database servers to which the service will connect to transmit data. (See Figure 9 on page 34.)
- BPI tab—used to define and manage error-handling instructions for broken pipes encountered by the Server. (See Figure 6. on page 29.)

Working with Service Options

The Service tab opens the Service page, which identifies the operating status of the DBG Server. It also contains options to start, refresh, or stop, and track and debug the service. (See Figure 4.)



DBG Configuration 5.2

Host: (local)

Status

● Running

☑ Auto Refresh

Tracing

□ Enable tracing

Trace level: 99

□ Trace to a file

Trace file name: D:\Program Files\DBGS\DBGS.TRC

LED icon identifying the state of the service.

Figure 4. Service page

In the Status area, the LED icon indicates the state of the service by color and designation:

- Green indicates that the service is currently running.

- Yellow indicates that the service has been paused.
- Red indicates that the service has been stopped.
- Gray indicates an unknown state.

The next few sections describe how to configure settings for the DBG service.

Starting and Stopping the Service

This section describes how to use the Configuration Tool to start and stop the DBG service. Alternatively, you can also start, stop, and even pause the service using the Windows NT Services window or the Windows NT NET command. (See "Alternate Methods of Starting, Stopping, and Pausing the Service" on page 44 for details.)

▶ **To start the service:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. If it's not already open, select the Service tab to display the Service page. (See Figure 4 on page 21.)

3. In the Status area, check the state of the service and act accordingly:

   - If the LED icon is green and indicates "Running" no action is needed; the service has already been started.
   - If the LED icon is red and indicates "Stopped," click **Start**.

     This action starts the service, as indicated by the change in the LED icon, and disables the **Start** button.

- If the LED icon is yellow and indicates "Paused," first click **Stop**, and then wait a few moments and click **Start**.

  This action initially stops, and then starts the service, as indicated by the changes in the LED icon. Also, stopping the service disables the **Stop** button, while starting the service disables the **Start** button.

- If the LED icon is gray and indicates "Unknown," click **Refresh**.

  This action instructs the Configuration Tool to check the most recent state of the service, and update the LED icon. If the LED icon indicates that the service is running, no further action is needed. If the LED icon indicates that the service has stopped, click **Start** to start the service.

4. Click **Exit** to close the Configuration Tool.

Caution:
Stopping the service disconnects all users, including administrators. Before stopping the service, send a network broadcast to alert users or schedule the stop at a more convenient time.

▶ **To stop the DBG service:**

1. Make sure that client applications are not running.
2. Start the Configuration Tool and connect to the DBG Server.
3. If it's not already open, select the Service tab to display the Service page. (See Figure 4 on page 21.)
4. In the Status area, check the state of the service and act accordingly:

   - If the LED icon is green and indicates "Running," or if the LED icon is yellow and indicates "Paused," click **Stop**.

     This action stops the service and disables the **Stop** button.

   - If the LED icon is red and indicates "Stopped," no action is needed; the service has already been stopped.

- If the LED icon is gray and indicates "Unknown," click **Refresh**.

  This action instructs the Configuration Tool to check the most recent state of the service, and update the LED icon. If the LED icon indicates that the service is running, click **Stop** to stop the service. If the LED icon indicates that the service is stopped, no further action is needed.

5. Click **Exit** to close the Configuration Tool.

Refreshing the Service

Refreshing allows you to review the most current state of the service, manually or automatically.

▶ **To refresh the service:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. If it's not already open, select the Service tab to display the Service page. (See Figure 4 on page 21.)

3. Choose one of the following:

   - To manually check the state of the service, click **Refresh**.
   - To set up an automatic check on the state of the service every five seconds, select the **Auto Refresh** box. (Selecting this option automatically disables the **Refresh** button.)

4. Click **Exit** to close the Configuration Tool.

Tracing the Service

Tracing creates logs of informational messages and calls to the DBG Server. With the Configuration Tool, you can set (optional) tracing options, which may be used as an aid in debugging and troubleshooting the service. For example, if the DBG service fails to start, you can use the trace window to learn why the service stopped.

▶ **To set tracing options:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. If it's not already open, select the Service tab to display the Service page. (See Figure 4 on page 21.)

3. Select the **Enable tracing** box (under the Tracing area) to trace the service.

4. Next, define the desired tracing options:

   - For detailed trace information, enter the level of detail you want in the **Trace level** box. The default is 99, which entails all trace messages. Typical trace level values include:

     - 1 for only errors.
     - 2 for both errors and warning messages.
     - 3 for errors, warning and informational messages.
     - 4-99 for varying degrees of errors, warning, informational, debugging, and user-defined messages.

   - If you want to open a trace window and view trace information in real-time, click the **Open** button. (To close a trace window, simply click the **Close** button on the trace window itself, or on the Service page.)

- To write trace information to a file, select the **Trace to a file** box and enter the directory path and file name of the trace file in the **Trace file name** box. The default trace file, DBGS.TRC, is located in the same directory in which the DBG Server executable resides. You can also specify a different trace file by entering its location. (Universal naming conventions (UNC) and remote files are not supported.)

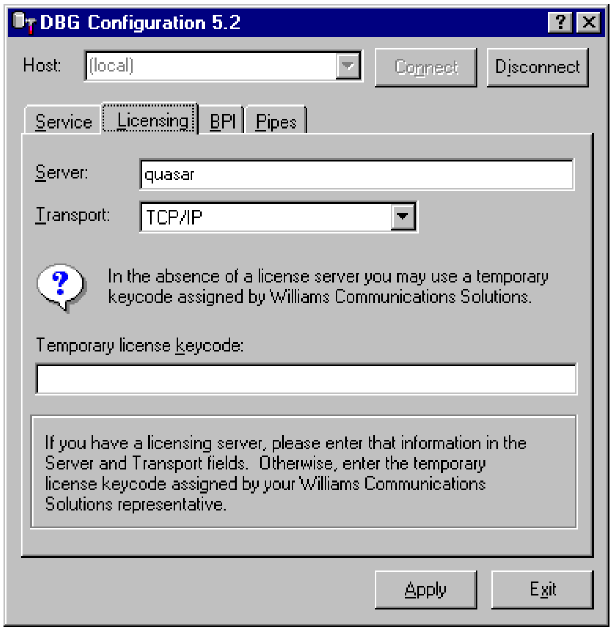
  > **Note:** Enabling tracing with maximum trace level details (e.g., 4-99) or writing trace information to a file may impact system performance.

5. To accept and save your entries, click **Apply**.
6. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Defining Licensing Options

At setup, you were given a choice to enter the location of the License Manager, or enter a keycode. The License Manager provides protection and verification of the individual Server software license; a keycode enables you to evaluate DBG for a limited time period.

To modify the location of the License Manager server, or update a keycode, you need to use the Licensing tab to open the Licensing page. (See Figure 5.)



Server: quasar

Transport: TCP/IP

In the absence of a license server you may use a temporary keycode assigned by Williams Communications Solutions.

Temporary license keycode:

If you have a licensing server, please enter that information in the Server and Transport fields. Otherwise, enter the temporary license keycode assigned by your Williams Communications Solutions representative.

Figure 5. Licensing page

▶ To specify or change your licensing information:

1. Start the DBG Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the Licensing tab to display the Licensing page. (See Figure 5.)

3. If you know the location of your License Manager Server:

   a) Enter the name of the server in the **Server** box.

   b) Pull down the **Transport** drop-down list and select the protocol to communicate with the License Manager Server. TCP/IP is selected by default.

4. If you do not know the location of, or have not installed the License Manager, enter a valid keycode in the **Temporary license keycode** box. The default is (none).

   Make sure that you enter the keycode exactly as it was presented to you. If you do not have a valid keycode, you may obtain a temporary one (limiting the time period that the service is available) from your Williams Communications Support representative.

5. To apply and save the licensing information or keycode, click **Apply**.

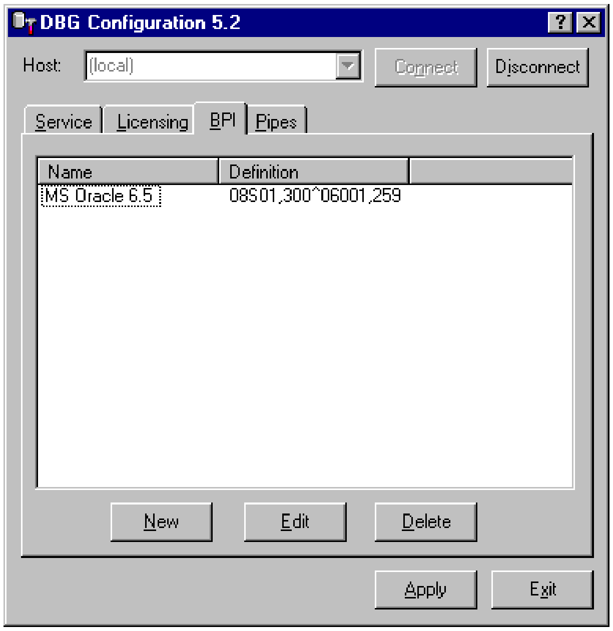
6. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Working with BPI Options

A BPI identifies a broken pipe indicator, which contains error handling instructions for problematic queries through pipes to specific database engines.

DBG relies on a user-specified BPI for resolving errors that may occur when submitting a statement to a database. If a database's ODBC driver returns one or more error codes that match the instructions in a specified BPI, the Server perceives the error as critical and sends an error message back to the calling application. If the ODBC driver returns error codes that do not match any of those specified in the BPI, DBG considers the error as insignificant and simply retries the query.

You can use the BPI tab to open the BPI page to specify and manage BPIs for DBG. (See Figure 6 on page 29.)



Host: (local)

Name	Definition
MS Oracle 6.5	08S01,300^06001,259

Figure 6. BPI page

Configuring a BPI

This section describes how to set up and define broken pipe indicators for error handling to specific database engines. To perform this task, you should be well-versed in ODBC device driver configurations.

▶ To define a BPI:

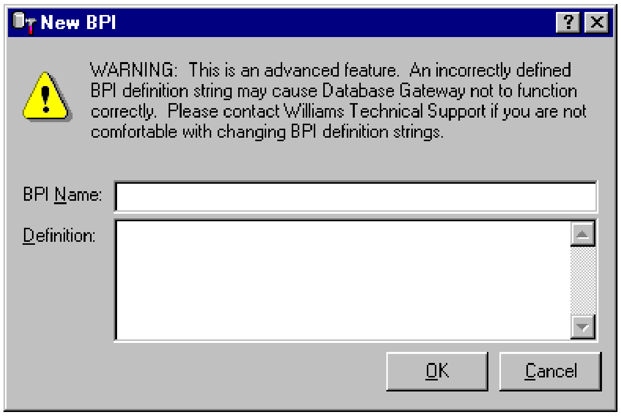
1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the BPI tab to display the BPI page. (See Figure 6.)

3. Click **New**.

   The New BPI dialog box displays. (See Figure 7.)



New BPI

WARNING: This is an advanced feature. An incorrectly defined BPI definition string may cause Database Gateway not to function correctly. Please contact Williams Technical Support if you are not comfortable with changing BPI definition strings.

Figure 7. New BPI dialog box

4. Define the properties for the new BPI:

    - In the **BPI Name** box, enter the name you want to assign to the BPI (required). For example, type: `MS Oracle 6.5`.

      For each BPI you create, each **BPI Name** must be unique.

    - In the **Definition** box, enter a string identifying which ODBC status codes and which native error codes DBG should consider critical (required).

      An ODBC status code typically consists of five or six alphanumeric characters, while a native error code is typically a number.

      Syntax rules for BPI definitions require you to distinguish ODBC status codes from native error codes by separating the two with a comma (,). For example, `04001,134` indicates that the returned error must match both the specified ODBC status code (`04001`) and the specified native error code (`134`). If you prefer, you may specify only one code, provided that you include the comma to distinguish one code from the other. For example, `04001,` states that the returned error must match the specified ODBC status code (`04001`).

In addition, syntax rules require multiple BPI definitions to be separated with a caret (^). For example, `0S003,^,1026` states that the returned error must match either the specified ODBC status code (`0S003`), or the native error code (`1026`). (Notice the use of a comma distinguishing the ODBC status code from the native error code in each instruction.)

5. Click **OK** when you are done.
6. To accept and save the BPI, click **Apply**.
7. To ensure that the new settings take effect, you must stop and then restart the service. For more information, see "Starting and Stopping the Service" on page 22.

Modifying a BPI

Modifying a BPI allows you to reconfigure the properties of an existing BPI.

▶ **To modify an existing BPI:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.
2. Select the BPI tab to display the BPI page. (See Figure 6 on page 29.)
3. Select the broken pipe indicator you want to modify and click **Edit**.

   The Edit BPI dialog box displays the selected BPI's properties. (See Figure 8 on page 32.)

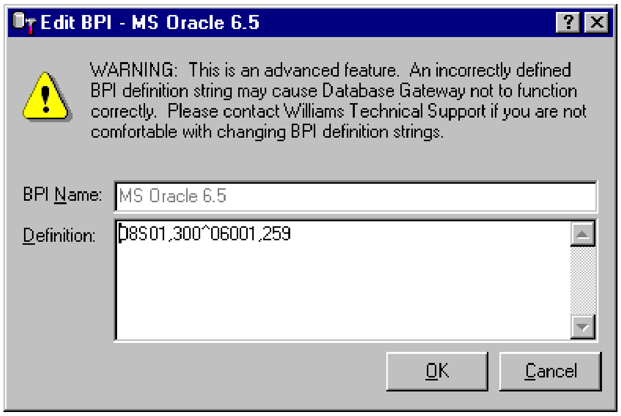

Edit BPI - MS Oracle 6.5
?
WARNING: This is an advanced feature. An incorrectly defined BPI definition string may cause Database Gateway not to function correctly. Please contact Williams Technical Support if you are not comfortable with changing BPI definition strings.
BPI Name:
MS Oracle 6.5
Definition:
D8S01,300^06001,259
OK
Cancel

Figure 8. Edit BPI dialog box

4. To modify the BPI's name or definition, place the cursor in the entry you want to change, and then append or delete the current information by entering the new information.

5. Click **OK** when you are done.

6. To accept and save your changes, click **Apply**.

7. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Deleting a BPI

Deleting a BPI permanently removes an existing BPI. You cannot delete a BPI if it is in use by a pipe; if the pipe is configured by the BPI, it cannot be deleted.

▶ To delete an existing BPI:

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the BPI tab to display the BPI page. (See Figure 6 on page 29.)
3. Select the broken pipe indicator you want to remove and click **Delete**.

   A message prompt displays, asking you to confirm the BPI's removal.
4. Click **Yes** to delete the selected BPI.

   The selected BPI and all of its properties are deleted.
5. To accept and save your changes, click **Apply**.
6. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Working with Pipe Options

To connect to a specific database, DBG requires a defined pipe connection to that database.

A "pipe" is a named entity through which applications using the DBG API may execute SQL (Structured Query Language) or related statements, which can be understood by the underlying database. A pipe opens a link to a specific database, creates a connection, and allows DBG to send and retrieve information from that database back to the calling application.

Use the Pipes tab to open the Pipes page to name and define pipe properties for a specific database. (See Figure 9 on page 34.)

Use the Pipes tab to add, modify, and delete pipes to external databases. To customize the way in which pipes are displayed, refer to the "Changing the View" section.

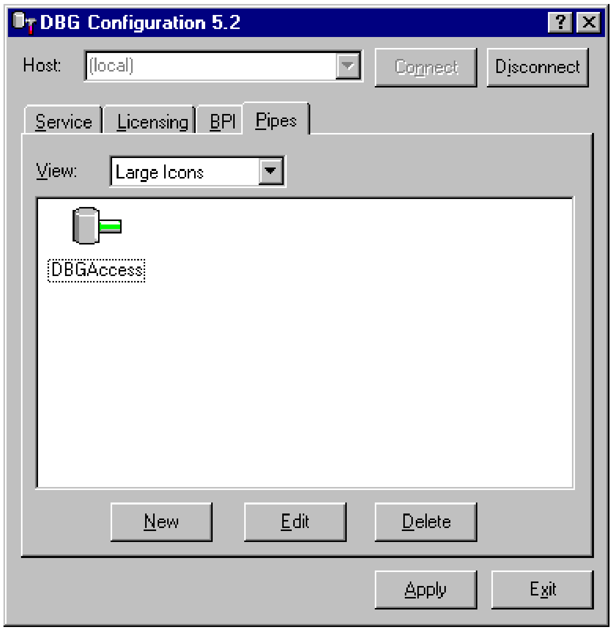


Figure 9. Pipes page

Configuring a Pipe

This section describes how to set up and define pipe connections and SQL (or other) statements directed toward a specific database engine. To perform this task, you should be well-versed in ODBC device driver and system DSN configurations.

▶ **To define a pipe:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the Pipes tab to display the Pipes page. (See Figure 9.)

3. Click **New**.

The New Pipe dialog box displays the Required page. (See Figure 10.)

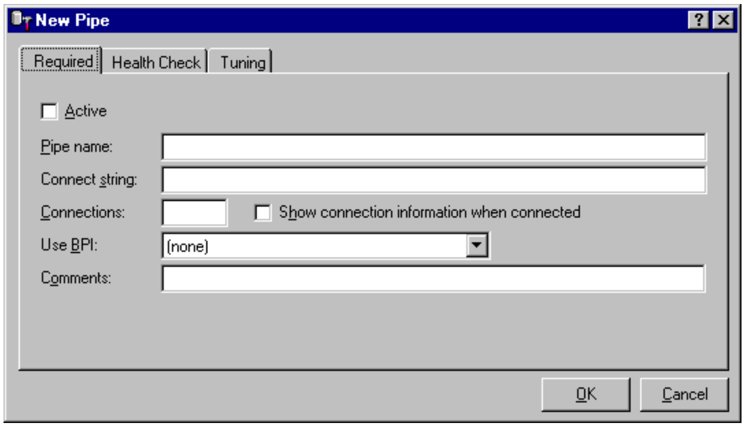


Figure 10. New Pipe dialog box - Required page

4. Use the Required page to define the pipe and its properties:

   a) Select the **Active** box to activate the pipe connection to the database (recommended).

      By default, this option is unchecked, indicating that the pipe is inactive. When a pipe connection is disabled, the Pipes page will overlay an "X" on the inactive pipe. (See Figure 9 on page 34.)

   b) In the **Pipe** name box, specify the name of the pipe (required).

      The maximum length for a pipe name is 33 characters. We recommend that you enter a name relevant to the database you want to access. For example, you may want to name a pipe "HumanRes" because it leads to a human resources database.

   c) In the **Connect** string box, specify a valid ODBC connection string for the database driver (required).

Not all ODBC drivers support the DATABASE parameter; we recommend including it to allow the ODBC driver to reattach to the correct database within an engine in the event of a reconnection.

A connect string typically contains one long string without any spaces, specified as:

```
DSN=<system_dsn_name>;UID=<user_id>;
PWD=<password>;DATABASE=<database name>
```

The DSN parameter (required) defines the DSN (data source network) on which the database resides; it must be a valid system DSN, previously configured using the 32-bit ODBC applet in the Windows NT Control Panel.

The UID and PWD parameters (optional) identify the user name (ID) and the password (PWD) needed to log into the database; both are case-sensitive. Also, these entries must have sufficient rights to allow DBG client applications to access the database.

The DATABASE parameter (optional) identifies the name of the database for those systems that support multiple databases within the same engine, such as Microsoft SQL Server, Oracle, Sybase, and Informix.

d) In the **Connections** box, specify the number of connections (or logins) to the database (required).

   The number of connections allows DBG to simultaneously execute multiple statements along the same pipe, in parallel.

e) Select the **Show connection information when connected** box to include the database connection in the trace.

f) In the **Use BPI** box, identify which broken pipe indicator to use to determine critical errors on the pipe (required).

   You must define a BPI before you can select one; see "Configuring a BPI" on page 29 for more information. Leaving this area blank indicates that DBG should use a generic BPI (default).

g) In the **Comments** box, specify any remarks about the pipe (optional).

5. Click on the Health check tab to display the Health Check page. (See Figure 11.)

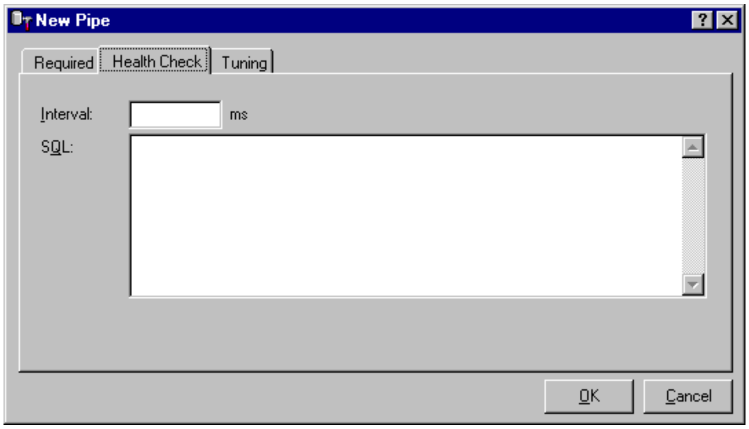


Figure 11. New Pipe dialog box - Health Check page

6. Use the Health Check page to periodically determine the connectivity status of the database and/or network:

Periodically submitting a query statement to the database allows DBG to use the previously specified BPI (in the Required tab) to determine when a network or database error occurs and to react by automatically taking the pipe offline and attempting a reconnection.

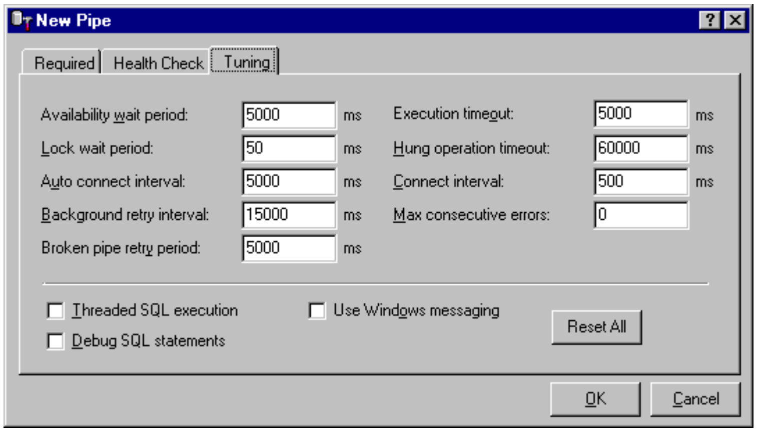
a) In the **Interval** box, specify how many milliseconds to wait before submitting a query to the database.

The default value is 5000 milliseconds (or 5 seconds; recommended).

b) In the **SQL** box, specify one or more query statements.

Each query must be created in a language that the database understands (SQL or otherwise). Refer to the database's documentation for details.

7. Click on the Tuning tab to display the Tuning page. (See Figure 12 on page 38.)



Availability wait period: 5000 ms

Lock wait period: 50 ms

Auto connect interval: 5000 ms

Background retry interval: 15000 ms

Broken pipe retry period: 5000 ms

Execution timeout: 5000 ms

Hung operation timeout: 60000 ms

Connect interval: 500 ms

Max consecutive errors: 0

□ Threaded SQL execution

□ Debug SQL statements

□ Use Windows messaging

Figure 12. New Pipe dialog box - Tuning page

We recommend you leave the Tuning page defaults unless otherwise directed by a member of Williams Technical Support.

8. The Tuning page contains the timing parameters for connecting to a database. The following describes each parameter:

   a) In the **Availability wait period** box, specify the number of milliseconds that DBG should wait for an available pipe connection.

      The default value is 5000 milliseconds (or 5 seconds; recommended). If a connection is not available within the specified time, DBG returns an error code to the calling application.

   b) In the **Lock wait period** box, specify the number of milliseconds that other applications must abide by before using the same pipe connection.

      The default value is 50 milliseconds (recommended). If a connection cannot be locked within the specified time, DBG returns an error code to the calling application.

c) In the **Auto connect interval** box, specify the number of milliseconds that DBG should wait between repeated attempts to reconnect to the database.

   The default value is 5000 milliseconds (or 5 seconds; recommended). The pipe is not available until connections within the pipe are opened to the database.

d) In the **Background retry interval** box, specify the number of milliseconds that DBG should wait between attempts to send SQL statements scheduled for background execution.

   The default value is 15000 milliseconds (or 15 seconds; recommended).

e) In the **Broken pipe retry period** box, specify the number of milliseconds that DBG should wait between attempts to retry SQL statements that failed due to a broken pipe.

   The default value is 5000 milliseconds (or 5 seconds; recommended). If the retry is not executed within the specified time, DBG returns an error code to the calling application.

f) In the **Execution timeout** box, specify the time in milliseconds in which a SQL statement may be executed before DBG cancels the query.

   The default value is 5000 milliseconds (or five seconds; recommended). If the statement is not executed within the specified time, DBG returns an error code to the calling application.

g) In the **Hung operation timeout** box, specify the number of milliseconds that the operation may take before DBG cancels the operation and performs an automatic shutdown.

   The default value is 60000 milliseconds (or one minute; recommended). If the operation exceeds the specified time, DBG returns an error code to the calling application and performs an automatic shutdown.

h) In the **Connect Interval** box, specify the number of milliseconds that DBG should pause in between connection attempts.

   The default value is 500 milliseconds (recommended.) A higher number may result in excessive retries or impact performance.

i) In the **Max consecutive errors** box, specify the maximum number of successive errors that DBG may encounter.

   The default value is zero. If the number of successive errors exceed the specified amount, DBG returns an error code to the calling application.

9. If needed, select the **Threaded SQL Execution** box to specify threaded SQL execution. (We recommend that you select this option only if you are experiencing problems with the database driver.)

   The default is turned off. Selecting this option starts a new thread every time a SQL statement needs to be executed.

10. Select the **Debug SQL statements** box to include executed SQL statements in a trace window.

    The default is turned off.

11. Select the **Use Windows Messaging** box to use Windows messaging to access an ODBC driver.

    The default is turned off.

12. To reset each of the tuning parameters back to their respective defaults, click **Reset All**.

13. Click **OK** when you are done.

14. To accept and save the pipe properties, click **Apply**.

15. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

    > **Tip:** If the service fails to start after configuring a pipe, check the trace file. If the problem is due to a hung operation on a pipe connection, increase the **Hung Operation Timeout** value, and then click **Apply**, reboot the system and restart the service.

Modifying a Pipe

Modifying a pipe allows you to reconfigure the properties for an existing pipe.

▶ **To modify an existing pipe:**

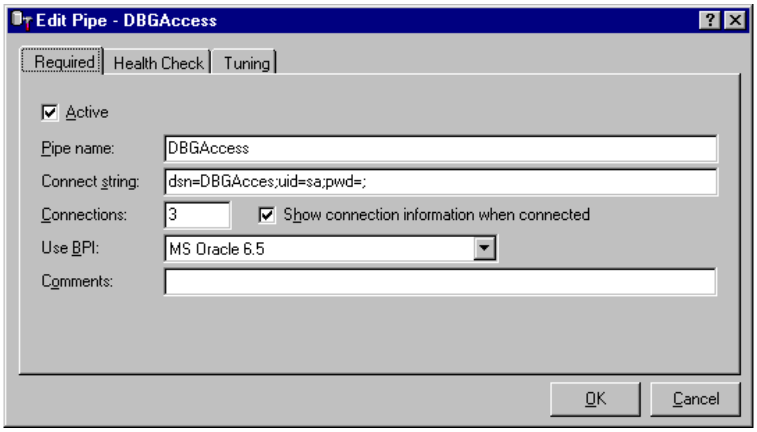
1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the Pipes tab to display the Pipes page. (See Figure 9 on page 34.)

3. Select the pipe you wish to modify and click **Edit**.

   The Edit Pipe dialog box opens to the Required page, which displays the selected pipe's current settings.



Edit Pipe - DBGAccess

Required | Health Check | Tuning

☑ Active

Pipe name: DBGAccess

Connect string: dsn=DBGAcces;uid=sa;pwd=;

Connections: 3 ☑ Show connection information when connected

Use BPI: MS Oracle 6.5

Comments:

Figure 13. Edit Pipe dialog box - Required page

4. To modify the selected pipe:

   a) Select the tab that holds the settings you want to change. Click on either the Required, Health Check or Tuning tab.

   b) Place the cursor in the entry you want to change, and then append or delete the current information by entering new information. To reset each of the tuning parameters back to their respective defaults, click **Reset All**.

   c) Click **OK** when you are done.

5. To accept and save your changes, click **Apply**.

6. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Changing the View

To customize the appearance of existing pipes in the Pipes tab, you can change the way in which pipes are viewed.

▶ **To customize the view for existing pipes:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" and "Establishing a Connection to the Server."

2. Select the Pipes tab to display the Pipes page. (See Figure 9 on page 34.)
3. To specify how pipes are displayed, select one of the following options from the **View** drop-down list:
   - Large Icons displays pipes by using large icons.
   - Small Icons displays pipes by using small icons.
   - List displays pipes in a sequential list.
   - Details displays information about each pipe.
4. To accept and save your changes, click **Apply**.
5. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Deleting a Pipe

Deleting a pipe permanently removes an existing pipe connection from the DBG registry.

▶ **To delete an existing pipe:**

1. Start the Configuration Tool and connect to the DBG Server.

   If necessary, refer to "Starting and Closing the DBG Configuration Tool" on page 18 and "Establishing a Connection to the Server" on page 20.

2. Select the Pipes tab to display the Pipes page. (See Figure 9 on page 34.)
3. Select the pipe you wish to remove and click **Delete**.

   A message prompt displays, asking you to confirm the pipe's removal.
4. Click **Yes** to delete the selected pipe.

   The selected pipe and all of its configurations are deleted.
5. To ensure that the new settings take effect, you must stop and then restart the service. See "Starting and Stopping the Service" on page 22 for more information.

Alternate Methods of Starting, Stopping, and Pausing the Service

In the previous section, you learned how to start and stop the service using the Configuration Tool. This section offers alternate methods for starting, stopping, and even pausing the DBG service using the Windows NT Services window or the Windows NT NET command on the computer in which the service resides.

▶ **To start, stop, or pause the DBG service via the Services window:**

1. On Windows NT 4.0, click the **Start** button, choose **Settings**, and then select **Control Panel**.
2. Double-click on the **Services** icon.

   The Services dialog box (not shown) displays.
3. Select "Dialect Database Gateway Server" and then, depending on the status listed in the Status column, choose which operation you wish to perform:

   a) If the Status column contains no status or "Stopped, click **Start** to start the service.

b) If the Status column contains "Paused*,"* click **Stop** to stop the service or **Continue** to restart the service.

c) If the Status column contains "Started*,"* click **Pause** to suspend the service or **Stop** to stop the service.

4. To change the service's startup settings, consult your Windows NT manual for additional information.

   Service startup settings take effect after the computer reboots and the service has been started. In addition, a running service must be stopped and restarted if settings change. However, before making changes to any service, make sure that you understand startup options for Windows NT services; configuring the DBG service with invalid startup options may render it unusable.

5. To accept and save your entries, click **Close**.

▶ **To start, stop, pause and continue the DBG service via the Windows NT NET command:**

1. Open a Windows NT command prompt window.

   If necessary, consult your Windows NT manual for additional information. Make sure that you use the NT command prompt, not an MS-DOS command prompt. Windows NT's DOS command prompt does not support all the features of the NT command prompt.

2. Depending on which operation you wish to perform, type the following at the command prompt:

   a) To start the service:

   - Type `NET START DBGS`.
   - Press **Enter**.

   b) To stop the service:

   - Type `NET STOP DBGS`.
   - Press **Enter**.

c) To pause the service:

- Type `NET PAUSE DBGS`.
- Press **Enter**.

d) To continue the service:

- Type `NET CONTINUE DBGS`.
- Press **Enter**.

Client API Programming

This chapter describes the API functions that client applications can use to access databases via DBG Servers.

> **Note:** While the DBG Server is Unicode compliant, version 5.4a of the DBG client API does not support Unicode or Double Byte Character Set (DBCS) character data. Character data in Unicode or DBCS client applications must be converted to ANSI format before calling the API functions.

Overview

Architecture

The architecture of the API is based on a set of functions that allow client applications to interact with the DBG system. Most API functions are client based (that is, they do not execute on the Server).

Functions that execute locally (on the client) do so primarily with a "workarea." A workarea is a block of reserved memory where data retrieved from a database, status codes, and other management information is stored. A client application can open as many workareas as it wishes, using each one to communicate with one or more pipes on the Server.

The base functionality of the DBG system is provided through a set of APIs that communicate with the Server using TCP/IP. For 32-bit Windows platforms, the library offers RPC versions of the available functions, allowing client applications to use RPC, rather than base TCP/IP as the communications medium.

Most API functions take a communication handle as a parameter. This handle, assigned when connecting to a Server, maps either to a TCP/IP socket or RPC session (Win32 platforms only) on the client, and is used to communicate with a particular Server.

> **Note:** With the unification of the TCP/IP and RPC versions of the API functions in DBG 5.0 and higher, almost all API functions look the same. Client communication with the server over the network is determined upon the initial connection to the server. Upon establishing communication, the API returns a handle, which dictates the type of communications medium in use. Because each handle has its own communications medium, it is conceivable that a client may have multiple connections using either base TCP/IP or RPC as the communications medium.

The API library supports automatic reconnectivity to the Server. Failover to backup systems is not currently available.

As of this writing, only one function (**DBGCExecute**) communicates with the Server software across the network. All other functions affect the local machine only. Use of network handles and errors in practically all functions (even when not needed) is to allow for future expansion.

Programming Examples

The examples in this chapter are for C/C++ and Visual Basic only.

Compiling

The `dbgcapi.h` header file is required for building a C/C++ application that uses the DBG API. This file will *#include* various support files provided in the DBG SDK on the DBG 5.4a CD-ROM. An SDK is provided for each supported platform.

A Visual Basic module (DBGCAPI.BAS) file is provided for 32-bit versions of Visual Basic 4.0 and higher. This module declares functions exported from the .DLL, as well as constants defined in the API. A special version of the BAS module (DBGCAVB3.BAS) is provided for Visual Basic 3 users.

> **Note:** As of this writing, C/C++ and Visual Basic support is only available for the Intel 80x86 CPU architecture.

Function Declarations

Functions are declared as **DBGCLIENTFUNC**, which is defined differently depending on the platform. The return value from all functions is a signed 32-bit long integer.

Table 2. Function Declarations

Platform	Library Type	Link type
32-bit Windows	Windows DLL (via import library)	long WINAPI
SCO UNIX	Static link	extern long

Linking

For C and C++ applications, linking with the DBG API on various platforms requires the definition of certain constants. You must define the correct constant(s) in your project in order to link correctly with the API library.

Table 3. Linking By Platform

Platform	Library Name	Define
32-bit Windows	DBGAPI32.LIB (DBGAPI32.DLL)	_WINDOWS _WIN32
SCO UNIX	dbgcapi.a	_UNIX _SCO

Support for Variable-Length Columns

Due to the limitations of some database systems, data retrieval for variable-length columns interspersed with fixed length columns is typically restricted. However, DBG provides support for variable-length columns, and can be made to successfully retrieve data from variable-length columns interspersed with fixed length columns, as long as you follow certain rules.

With DBG, the key to ensuring the successful retrieval of variable- and fixed-length column data from any database is twofold. First, SELECT statements (or others that yield data from the database) must be carefully constructed to specify the columns you wish to retrieve, in the order you wish to retrieve them. Second, DBG requires that names of the columns that contain variable-length data must appear AFTER the last column that contains fixed-length data.

For example, say that we have a table named 'people,' which consists of four columns: 'name,' 'notes,' 'age,' and 'comments.' Furthermore, let us assume that the 'name' and 'age' columns are fixed-length columns, while the 'notes' and 'comments' are variable-length columns. To retrieve all fields and all records from the table, your SELECT statement should be as follows:

```
SELECT name, age, notes, comments FROM
people
```

not

```
SELECT name, notes, age, comments FROM
people
```

and certainly not

`SELECT * from people`.

With the `SELECT name, age, notes, comments FROM people` statement, the variable-length columns appear at the end of the column list specification, rather than being interspersed with fixed-length columns. The construction of the SELECT statement also ensures that DBG will be able to retrieve the data after the SELECT statement has executed.

Function Listing

This section lists the individual DBG functions alphabetically, and provides details for each one.

Table 4. Available Functions

Function Call	Description
DBGCCloseWorkarea	Closes a workarea in the client API that is uniquely identified by the handle in the variable *lWAHnd*.
DBGCConnect	Connects to the DBG Server running on *lpszServerName*.
DBGCDisconnect	Relinques the connection to the DBG Server.
DBGCErrorMsg	Retrieves the error message associated with *lErrorCode*, placing it in the buffer pointed to by *lpszBuffer*.
DBGCExecute	Passes the SQL statement pointed to by *lpszSQL* to the DBG Server for execution on the database connected via the pipe *lpszPipeName*.

Function Call	Description
DBGCGetAPIVersion	Returns the version of the Database Gateway API library in use.
DBGCGetColData	Retrieves data in the current record from the column specified by *lpszColName*. Up to *lSize* bytes are copied into the buffer pointed to by *lpvData*.
DBGCGetColName	Returns the name of the column specified by *lIndex*. The name is returned as a zero terminated string into the buffer pointed to by *lpszName*.
DBGCGetColSize	Returns the number of bytes occupied by data in the column specified by *lpszColName*. The size value is returned in the long pointed to by *lplSize*.
DBGCGetColType	Returns the internal DBG data type of the column specified by *lpszColName*. The type value is returned in the long pointed to by *lplType*.
DBGCGetODBCColType	Returns the ODBC data type of the column specified by *lpszColName*. The type value returned in the long pointed to by *lplType*.
DBGCGetODBCErrMsg	Returns the current ODBC error message in the workarea. The zero-terminated message is placed into the variable pointed to by *lpszBuffer*.

Function Call	Description
DBGCGetODBCErrMsgLen	Returns the length of the current ODBC error message in the workarea. The length is placed into the variable pointed to by *lplSize*.
DBGCGetServerVersion	Returns the version of the Database Gateway Server.
DBGCInitialize	Initializes the DBG API library for non-Windows platforms.
DBGCLoadCommTimeouts	Loads timeout values to be used by the native TCP/IP transport
DBGCMoveFirst	Moves to the first record in the workarea identified by *lWAHnd*. This function must always be called first, before any move.
DBGCMoveLast	Moves to the last record in the workarea identified by *lWAHnd*.
DBGCMoveNext	Moves to the next record in the workarea identified by *lWAHnd*.
DBGCMovePrevious	Moves to the previous record in the workarea identified by *lWAHnd*.
DBGCOpenWorkarea	Opens (creates) a new workarea in the client API uniquely identified by the handle returned in the variable pointed to by *lplWAHnd*.
DBGCReset	Resets the workarea clearing all records and other data returned from the server.
DBGCSetCommTimeouts	Specifies timeout values to be used by the native TCP/IP transport.
DBGCShutdown	Relinquishes the DBG API library on non-Windows platforms.

The remainder of this chapter is a detailed listing of all Dialect Database Gateway API functions.

DBGCCloseWorkarea

Declaration DBGCLIENTFUNC **DBGCCloseWorkarea**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks Close a workarea in the client API uniquely identified by the handle in the variable *lWAHnd*. Once a workarea is closed, the handle becomes invalid and may not be used to access workareas again.

Example Create a new workarea, closing it immediately.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCOpenWorkarea(lNetConn, &lNetErr,
   &lWA);
lRet = DBGCCloseWorkarea(lNetConn, &lNetErr,
   lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
```

```
lRet = DBGCOpenWorkarea(lNetConn, lNetErr,
   lWA)
```

```
lRet = DBGCCloseWorkarea(lNetConn, lNetErr,
   lWA)
```

DBGCConnect

Declaration DBGCLIENTFUNC **DBGCConnect**(
LPCSTR *lpszServerName*,
LONG *lConnType*,
LPCSTR *lpszConnParam1*,
LPCSTR *lpszConnParam2*,
LPCSTR *lpszConnParam3*,
LPLONG *lplNetConn*,
LPLONG *lplNetErr*);

Parameters

lpszServerName The name of the host computer running the DBG Server.

For TCP/IP, this parameter may be the DNS name or IP address. For RPC, the value of this parameter depends on the protocol in use: IP protocol uses the DNS name or IP address, while Named Pipes, Local RPC, IPX, and SPX use the computer name. Note: IPX and SPX require a NetWare™ bindery on the network.

lConnType The type of network connection. Specify **DBGNET_CONN_TCP** for TCP/IP on all platforms, **DBGNET_CONN_RPC** on those platforms supporting RPC.

lpszConnParam1 A string parameter, which has different meanings, based on the value of *lConnType*.

For TCP/IP connections, this is the port number the Server is listening on (usually 5200). For RPC connections, this is the name of the RPC protocol sequence to be used:

- `ncacn_np` - Named Pipes
- `ncacn_ip_tcp` - TCP/IP
- `ncalrpc` - local RPC
- `ncadg_ipx` - IPX
- `ncacn_spx` - SPX

lpszConnParam2 Reserved for future use.

lpszConnParam3 Reserved for future use.

lpNetConn Pointer to a long which will hold the handle to the connection.

lplNetErr Pointer to a long to hold the error code reported by the network.

Remarks Connects to the DBG Server running on *lpszServerName*.

The *lConnType* parameter dictates how **DBGCConnect** should connect to the Server, while the *lpszConnParam1* string that follows specifies the parameters for that (Server) connection. TCP/IP connections require the port number(s) on which the Server is listening; RPC connections require the protocol sequence.

> **Note:** By default, the server listens on port 5200 for TCP/IP connections**.**

Example 1 Connect to the DBG Server on *corp_srv* over TCP/IP; disconnect immediately.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;

lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "",
   &lNetConn, &lNetErr);
lRet = DBGCDisconnect(lNetConn, &lNetErr);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long

lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "", lNetConn,
   lNetErr)
lRet = DBGCDisconnect(lNetConn, lNetErr)
```

Example 2 Connect to the DBG Server on *corp_srv* over RPC using Named Pipes as the protocol sequence; disconnect immediately.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;

lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_RPC, "ncacn_np", "", "",
   &lNetConn, &lNetErr);
lRet = DBGCDisconnect(lNetConn, &lNetErr);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
```

```
lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_RPC, "ncacn_np", "", "",
   lNetConn, lNetErr);
lRet = DBGCDisconnect(lNetConn, lNetErr)
```

DBGCDisconnect

Declaration DBGCLIENTFUNC **DBGCDisconnect**(
LONG *lNetConn,*
LPLONG *lplNetErr*);

Parameters *lNetConn* Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

Remarks Disconnects from the current Server.

Example Connect to the DBG Server on *corp_srv*; disconnect immediately.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;

lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "",
   &lNetConn, &lNetErr);
lRet = DBGCDisconnect(lNetConn, &lNetErr);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long

lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "", lNetConn,
   lNetErr)
lRet = DBGCDisconnect(lNetConn, lNetErr)
```

DBGCErrorMsg

Declaration DBGCLIENTFUNC **DBGCErrorMsg**(
LONG *lErrorCode*,
LPSTR *lpszBuffer*,
LONG *lBuffSize*);

Parameters

lErrorCode	An error code returned by DBG.
lpszBuffer	A pointer to a buffer to hold the error message.
lBuffSize	The size of the buffer pointed to by *lpszBuffer*.

Remarks Retrieves the error message associated with *lErrorCode*, which is always zero-terminated, and places it in the buffer pointed to by *lpszBuffer*.

If the buffer is too small to contain the entire error message, the API will truncate the message to a size of one less than the value in *lBuffSize*; the additional byte will be used to zero-terminate the string.

Example Determine the error message associated with a DBG error code.

C/C++

```
LONG   lRet;
LONG   lWA;
LONG   lSize;
char   cBuffer[256];

lRet = ..some DBG operation..
lRet = DBGCGetErrMsg(lRet, cBuffer,
   sizeof(cBuffer));
printf("Error message is %s\n", cBuffer);
```

Visual Basic

```
Dim lRet As Long
Dim lWA As Long
Dim lSize As Long
Dim cBuffer As String * 256

lRet = ..some DBG operation..
lRet = DBGCGetErrMsg(lRet, cBuffer, 256)
Debug.Print "Error message is " & cBuffer
```

DBGCExecute

Declaration DBGCLIENTFUNC **DBGCExecute**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPCSTR *lpszPipeName*,
LPCSTR *lpszSQL*,
DBG_REQ **lpReq*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.
lpszPipeName	The name of a valid pipe on the Server.
lpszSQL	A valid SQL statement to be passed on to the database to which the pipe is connected.
lpReq	Pointer to a **DBG_REQ** structure containing additional information about the execute request.

Remarks Passes the SQL statement pointed to by *lpszSQL* to the DBG Server for execution on the database connected via the pipe *lpszPipeName*.

> **Note:** To ensure a successful retrieval of recordsets, place selections for variable-length columns at the end of each SELECT statement.

Additional information concerning the request, such as number of records to retrieve (valid only for statements yielding records, such as **SELECT**), and whether the statement is to be executed in the background, is contained in the **DBG_REQ** structure pointed to by *lpReq*. This structure is filled in by the API when the function call returns with information from the Server, including number of records and columns returned, error codes, and so forth. (See "Appendix A. Data Structures and Constants" for structure details.)

> **Note:** The workarea specified by *lWAHnd* is reset when this function is called. All data, status codes, etc., are cleared from the workarea prior to the function being executed on the server.

Example

Retrieve and display the names and ages of the top three sales persons in the state of California. This example uses the database connected to via the `corp_sales_pipe` pipe on the Server. The actual database system being used is irrelevant, provided the SQL statement is valid for that database.

This is a complete example demonstrating many DBG API functions. No error checking is included.

> **Note:** The example presents two options for connecting across the network: native TCP/IP or RPC (in a Win32 environment only).
>
> The RPC version uses Named Pipes (`"ncacn_np"`) to ensure a secure login to the Windows NT machine hosting DBG Server.
>
> Regardless of the connection method, subsequent function calls (including the disconnect logic) remain the same.

C/C++

```
LONG      lRet;
LONG      lNetConn;
LONG      lNetErr;
LONG     lWA;
DBG_REQ  tReq;
char     cName[32];
long     lAge;

/* Connect - use only one of the following */
lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "",
   &lNetConn, &lNetErr);
     ----OR----
lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_RPC, "ncacn_np", "", "",
   &lNetConn, &lNetErr);

/* Open a workarea */
lRet = DBGCOpenWorkarea(lNetConn, &lNetErr,
   &lWAHnd);

/* Indicate that we're only interested in the
   first three records and that the statement
   must be executed immediately */
tReq.m_lRecsRequired = 3;
tReq.m_lBackground = 0;

/* Execute the SQL against the database */
lRet = DBGCExecute(lNetConn, &lNetErr,
   lWAHnd, "select name, age from
   sales_people where state = 'CA' order by
   revenue desc", "corp_sales_pipe", &tReq);

/* Display the results */
printf("Top 3 sales people in California\n");
lRet = DBGCMoveFirst(lNetConn, &lNetErr,
   lWAHnd);
while( lRet == ERR_DBG_NONE )
{
  lRet = DBGCGetColData(lNetConn, &lNetErr,
   lWAHnd, "name", cName);
  lRet = DBGCGetColData(lNetConn, &lNetErr,
   lWAHnd, "age", &lAge);
  printf("%s is %ld years old\n", cName,
   lAge);
  lRet = DBGCMoveNext(lNetConn, &lNetErr,
   lWAHnd);
}

/* Clean up and disconnect */
lRet = DBGCCloseWorkarea(lNetConn, &lNetErr,
   lWAHnd);
lRet = DBGCDisconnect(lNetConn, &lNetErr);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim tReq As DBG_REQ
Dim cName As String * 32
Dim lAge As Long

' Connect - use only one of the following
lRet = DBGCConnect("corp_srv",
   DBGNET_CONN_TCP, "5200", "", "", lNetConn,
   lNetErr)
     ----OR----
lRet = DBGCConnectRpc("corp_srv",
   DBGNET_CONN_RPC, "ncacn_np", "", "",
   lNetConn, lNetErr)

' Open a workarea
lRet = DBGCOpenWorkarea(lNetConn, lNetErr,
   lWAHnd)

' Indicate that we're only interested in the
' first three records and that the statement
' must be executed immediately
tReq.m_lRecsRequired = 3
tReq.m_lBackground = 0

' Execute the SQL against the database
lRet = DBGCExecute(lNetConn, lNetErr, lWAHnd,
   "select name, age from sales_people where
   state = 'CA' order by revenue desc",
   "corp_sales_pipe", tReq)

' Display the results
Debug.Print "Top 3 sales people in
   California"
lRet = DBGCMoveFirst(lNetConn, lNetErr,
   lWAHnd)
Do While lRet = ERR_DBG_NONE
  lRet = DBGCGetColData(lNetConn, lNetErr,
   lWAHnd, "name", cName)
  lRet = DBGCGetColData(lNetConn, lNetErr,
   lWAHnd, "age", lAge)
  Debug.Print cName & " is " & lAge & " years
   old"
  lRet = DBGCMoveNext(lNetConn, lNetErr
   lWAHnd)
Loop

' Clean up and disconnect
lRet = DBGCCloseWorkarea(lNetConn, lNetErr,
   lWAHnd)
lRet = DBGCDisconnect(lNetConn, lNetErr)
```

DBGCGetAPIVersion

Declaration DBGCLIENTFUNC **DBGCGetAPIVersion**(**LPSTR** *lpszBuff*);

Parameters *lpszBuff* Pointer to a buffer to hold the returned (zero terminated) API version string.

Remarks

Returns the version of the Database Gateway API in use as an ANSI, zero terminated string.

> **Note:** Although the API version string is currently less than 32 bytes, it is recommended that the buffer not be smaller than 32 bytes, including the zero-terminator, to allow for future version strings that may be larger.

The returned string uses the following format: a.b.c.d, where:

String Position	Represents	Description
a	major version number	Increments each time significant enhancements and features are added to the software.
b	minor version number	Changes when an existing feature is modified or a small feature or enhancement is added, and only when these changes do not affect the existing operations of the software. In addition, this value is reset to zero when the major version number changes.
c	release stage	Indicates where in the release process the software currently is, and uses the ending digit of the release code to signify the actual release stage. In addition, this value is reset when the minor version number changes. For example, 5, 15, 25, 205 all mean the same; the digits prior to the release code are internal tracking numbers used to indicate iterations through the release cycle, shown below: 0 Software is under development,

String Position	Represents	Description
		e.g. 0, 10, 20.
		1 Software has passed primary developer testing, e.g. 1, 11, 21.
		2 Software has passed secondary developer testing, e.g. 2, 12, 22.
		3 Software has passed primary quality assurance, e.g. 3, 13, 23.
		4 Software has passed secondary quality assurance, e.g. 4, 14, 24.
		5 Software has passed field trials, is available for distribution, and will include technical support, e.g. 5, 15, 25. Any modifications to the software at this point require (at minimum) a change in the minor version number. Only one version of the software for a particular major and minor version will ever reach stage 5.
		6 - 9Not currently used
d	build number	Increments for each software build, which is released for primary developer testing. This value is never reset

The following table provides examples of software version numbers and describes how to interpret them:

Version	Explanation
2.4.0.15	The software is at release 2.4 and is currently under development.
2.4.1.16	The software is at release 2.4 and has passed primary developer testing.

Version	Explanation
2.4.30.27	The software is at release 2.4 and is under development. The software is in its third iteration through the release cycle.
2.4.31.42	The software is at release 2.4 and has passed primary developer testing. The software is in its third iteration through the release cycle.
2.4.34.51	The software is at release 2.4 and has passed secondary QA. The software is now ready for field trials. Support is limited to issues related to field trials.
2.4.35.51	The software is at release 2.4 and is ready for distribution. Official support is now available for the product.

Example

Display the version number of the API library.

C/C++

```
LONG lRet;
char cBuff[32];

lRet = DBGCGetAPIVersion(cBuff);
printf("API version is: %s\n", cBuff);
```

Visual Basic

```
Dim lRet As Long
Dim cBuff As String

cBuff = String(32, 0)
lRet = DBGCGetAPIVersion(cBuff)
cBuff = Left(cBuff, InStr(cBuff, Chr(0))-1)
Debug.Print "API version is: " & cBuff
```

DBGCGetColData

Declaration

DBGCLIENTFUNC **DBGCGetColData**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPCSTR *lpszColName*,
LPVOID *lpvBuffer*,
LONG *lSize*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.
lpszColName	The name of the column.
lpvData	A pointer to a buffer to copy the data into. The buffer must be at least *lSize* bytes in length.
lSize	The number of bytes to copy into the buffer.

Remarks

Retrieves data in the current record from the column specified by *lpszColName*. Up to *lSize* bytes are copied into the buffer pointed to by *lpvData*.

If *lSize* is larger than the size of the column, the size of the column dictates how much data will be copied into the buffer.

A binary transfer of the data is made from the workarea to the buffer. No translation is performed on the data as it is copied.

> **Note:** Before calling this function, buffers should be initialized to binary zero to allow for automatically zero-terminating character data and initializing numeric data.

Example Retrieve the "`zipcode`" column from workarea's current record.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
char   cData[32];

memset(cData, 0, sizeof(cData));
lRet = DBGCGetColData(lNetConn, &lNetErr,
   lWA, "zipcode", cData, sizeof(cData));
printf("Zip code is %s.\n", cData);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim cData As String

cData = String(32, 0)
lRet = DBGCGetColData(lNetConn, lNetErr, lWA,
   "zipcode", cData, 32)
Debug.Print "Zip code is " & cData
```

DBGCGetColName

Declaration DBGCLIENTFUNC **DBGCGetColName**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LONG *lIndex*,
LPSTR *lpszName*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

lIndex The zero-based index of the column name to retrieve.

lpszName A pointer to a buffer large enough to hold the name of the column.

Remarks

Returns the name of the column specified by *lIndex*. The name is returned as a zero-terminated string into the buffer pointed to by *lpszName*; the named buffer must be large enough to hold the name of the column.

> **Note:** Column numbers begin at 0.

Example

Retrieve the name of the third column (column 2) in the workarea's recordset.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
char   cColName[256];

lRet = DBGCGetColName(lNetConn, &lNetErr,
   lWA, 2, cColName);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim cColName As String * 256
```

```
lRet = DBGCGetColName(lNetConn, &lNetErr,
   lWA, 2, cColName)
```

DBGCGetColSize

Declaration

DBGCLIENTFUNC **DBGCGetColSize**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPCSTR *lpszColName*,
LPLONG *lplSize*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.
lpszColName	The name of the column.
lplSize	A pointer to a long that will hold the size of the column's data.

Remarks

Returns the number of bytes occupied by data in the column specified by *lpszColName*. The size value is returned in the long pointed to by *lplSize*.

Example

Retrieve the size of the "`zipcode`" column in the workarea's recordset.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
LONG   lColSize;
```

```
lRet = DBGCGetColSize(lNetConn, &lNetErr,
   lWA, "zipcode", &lColSize);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim lColSize As Long

lRet = DBGCGetColSize(lNetConn, lNetErr, lWA,
   "zipcode", lColSize)
```

DBGCGetColType

Declaration DBGCLIENTFUNC **DBGCGetColType**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPCSTR *lpszColName*,
LPLONG *lplType*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

lpszColName The name of the column.

lplType A pointer to a long that will hold the DBG data type of the column.

Remarks Returns the internal DBG data type of the column specified by *lpszColName*. The type value is returned in the long pointed to by *lplType*.

(See "Data Types" on page 100 for a complete list of data types processed by DBG.)

> **Note:** The DBG Server translates ODBC data type values to internal values. Use the **DBGCGet-ODBCColType** function to determine the ODBC data type returned by the database's ODBC driver.

Example Retrieve the DBG data type of the "`zipcode`" column in the workarea's recordset.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
LONG   lColType;

lRet = DBGCGetColType(lNetConn, &lNetErr,
   lWA, "zipcode", &lColType);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim lColType As Long

lRet = DBGCGetColType(lNetConn, lNetErr, lWA,
   "zipcode", lColType)
```

DBGCGetODBCColType

Declaration DBGCLIENTFUNC **DBGCGetODBCColType**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPCSTR *lpszColName*,
LPLONG *lplType*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.
lpszColName	The name of the column.
lplType	A pointer to a long that will hold the ODBC data type of the column.

Remarks Returns the ODBC data type of the column specified by *lpszColName*. The type value returned in the long pointed to by *lplType*.

(See "Data Types" on page 100 for a complete list of data types processed by DBG.)

Example Retrieve the ODBC data type of the "`zipcode`" column in the workarea's recordset.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
LONG   lColType;

lRet = DBGCGetODBCColType(lNetConn, &lNetErr,
   lWA, "zipcode", lColType);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim lColType As Long

lRet = DBGCGetODBCColType(lNetConn, lNetErr,
   lWA, "zipcode", lColType)
```

DBGCGetODBCErrMsg

Declaration DBGCLIENTFUNC **DBGCGetODBCErrMsg**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPSTR *lpszBuffer*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.
lpszBuffer	A pointer to a buffer to hold the ODBC error message. The buffer must be large enough to contain the entire error message including the zero terminator.

Remarks Returns the current ODBC error message in the workarea. The zero-terminated message is placed into the variable pointed to by *lpszBuffer*.

> **Note:** The buffer must be large enough to contain the entire error message including the zero terminator.

Example

Determine the length of the current ODBC error message, size a buffer accordingly, retrieve the message and display it.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
LONG   lSize;
char   *lpszBuffer;

lRet = DBGCGetODBCErrMsgLen(lNetConn,
   &lNetErr, lWA, &lSize);
lpszBuffer = malloc(lSize);
memset(lpszBuffer, 0, lSize);
lRet = DBGCGetODBCErrMsg(lNetConn, &lNetErr,
   lWA, lpszBuffer);
printf("Error message is %s\n", lpszBuffer);
free(lpszBuffer);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim lSize As Long
Dim cBuffer As String

lRet = DBGCGetODBCErrMsgLen(lNetConn,
   lNetErr, lWA, lSize)
cBuffer = String(lSize, 0)
lRet = DBGCGetODBCErrMsg(lNetConn, lNetErr,
   lWA, cBuffer)
Debug.Print "Error message is " & cBuffer
```

DBGCGetODBCErrMsgLen

Declaration

DBGCLIENTFUNC **DBGCGetODBCErrMsgLen**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*,
LPLONG *lSize*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

lplSize A pointer to a long integer to receive the length of the ODBC error message (including zero terminator).

Remarks

Returns the length of the current ODBC error message in the workarea. The length is placed into the variable pointed to by *lplSize*. The length returned by the API includes the zero terminating byte.

Example

Retrieve the length of the current ODBC error message.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;
LONG   lSize;

lRet = DBGCGetODBCErrMsgLen(lNetConn,
   &lNetErr, lWA, &lSize);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long
Dim lSize As Long

lRet = DBGCGetODBCErrMsgLen(lNetConn,
   lNetErr, lWA, lSize)
```

DBGCGetServerVersion

Declaration DBGCLIENTFUNC **DBGCGetServerVersion**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LPSTR *lpszBuff*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lpszBuff Pointer to a buffer to hold the returned (zero terminated) Server version string.

Remarks Returns the version of the Database Gateway Server as an ANSI, zero terminated string.

> **Note:** Although the server version string is currently less than 32 bytes, it is recommended that the buffer not be smaller than 32 bytes, including the zero-terminator, to allow for future version strings that may be larger.

The returned string uses the following format: a.b.c.d, where:

String Position	Represents	Description
a	major version number	Increments each time significant enhancements and features are added to the software.
b	minor version	Changes when an existing feature is modified or a small feature or

String Position	Represents	Description
	number	enhancement is added, and only when these changes do not affect the existing operations of the software. In addition, this value is reset to zero when the major version number changes.
c	release stage	Indicates where in the release process the software currently is, and uses the ending digit of the release code to signify the actual release stage. In addition, this value is reset when the minor version number changes. For example, 5, 15, 25, 205 all mean the same; the digits prior to the release code are internal tracking numbers used to indicate iterations through the release cycle, shown below: 0 Software is under development, e.g. 0, 10, 20. 1 Software has passed primary developer testing, e.g. 1, 11, 21. 2 Software has passed secondary developer testing, e.g. 2, 12, 22. 3 Software has passed primary quality assurance, e.g. 3, 13, 23. 4 Software has passed secondary quality assurance, e.g. 4, 14, 24. 5 Software has passed field trials, is available for distribution, and will include technical support, e.g. 5, 15, 25. Any modifications to the software at this point

String Position	Represents	Description
		require (at minimum) a change in the minor version number. *Only one version of the software for a particular major and minor version will ever reach stage 5.*
		6 – 9 Not currently used
d	build number	Increments for each software build, which is released for primary developer testing. This value is never reset.

The following table provides examples of software version numbers and describes how to interpret them:

Version	Explanation
2.4.0.15	The software is at release 2.4 and is currently under development.
2.4.1.16	The software is at release 2.4 and has passed primary developer testing.
2.4.30.27	The software is at release 2.4 and is under development. The software is in its third iteration through the release cycle.
2.4.31.42	The software is at release 2.4 and has passed primary developer testing. The software is in its third iteration through the release cycle.
2.4.34.51	The software is at release 2.4 and has passed secondary QA. The software is now ready for field trials. Support is limited to issues related to field trials.
2.4.35.51	The software is at release 2.4 and is ready for distribution. Official support is now available for the product.

Example Display the version number of the Database Gateway Server.

C/C++

```
LONG  lRet;
LONG  lNetConn;
LONG  lNetErr;
char  cBuff[32];

lRet = DBGCGetServerVersion(lNetConn,
   &lNetErr, cBuff);
printf("Server version is: %s\n", cBuff);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim cBuff As String

cBuff = String(32, 0)
lRet = DBGCGetServerVersion(lNetConn,
   lNetErr, cBuff)
cBuff = Left(cBuff, InStr(cBuff, Chr(0))-1)
Debug.Print "Server version is: " & cBuff
```

DBGCInitialize

Declaration DBGCLIENTFUNC **DBGCInitialize**()

Parameters None

Remarks Initializes the DBG API library for non-Windows platforms.

This function must be called once, at the beginning of your application, and before calling any other DBG API functions on non-Windows platforms. There should be a corresponding call to **DBGCShutdown** at the end of your program.

> **Caution:** Invoking this function in a Windows environment may cause unexpected results. The Windows implementation of the client API automatically initializes and destroys internal data structures when loading and unloading.

Example Initialize the API library in a non-Windows application.

C/C++

```
LONG   lRet;
lRet = DBGCInitialize();
```

Visual Basic

N/A – non-Windows environments only.

DBGCLoadCommTimeouts

Declaration DBGCLIENTFUNC **DBGCLoadCommTimeouts**(
LPCSTR *lpszFileName*);

Parameters *lpszFileName* Name of the file containing TCP/IP communications timeout parameters.

Remarks Loads timeout values which are to be used by the base TCP/IP transport. Values are loaded from the file name specified by *lpszFileName*. The file itself is an ANSI text file with a value on each line. Each value specifies a different timeout parameter (see Table 5 that follows) and must be followed by the new line sequence for the particular operating system.

Values are divided into send and receive operational timeouts specified in seconds. When using the TCP/IP base transport, a typical API function call consists of a send operation, followed by a receive operation. The send operation sends instructions over the network to the DBG Server, while the receive operation waits on a response, also from the Server. If the send or receive operation times out, the function is aborted and an error is returned to the calling application. RPC connectivity is not affected by these timeouts

DBGCLoadCommTimeouts can be called at any time during the execution of a program, but it is recommend that it be called once, at the beginning of the program (after **DBGCInitialize**, if applicable).

RPC functions are not affected by these timeout values.

Table 5. Timeout Values

Timeout	Description
SendWaitMax	Maximum time in seconds that the network send routines wait for the socket to become available for writing.
SendOpMax	Maximum time in seconds that the network send routines may take to execute an entire send operation.
ReceiveWaitMax	Maximum time in seconds that the network receive routines wait for the socket to become available for reading.
ReceiveOpMax	Maximum time in seconds that the network receive routines may take to execute an entire receive operation.

> **Note:** Timeout values are global to the API.
>
> A value less than one second will use the default values built into the API. These values are five seconds for waits and 30 seconds for operations.

Example

The following text file, TIMEOUT.DBG, created using a standard text editor, contains timeout values. Each line is followed by a newline character, including the last line.

Send operations will wait for up to 5 seconds for the socket to become available, and send operations may not exceed 10 seconds. Receive operations will wait up to 6 seconds for the socket to become available and will timeout after 25 seconds.

```
5
10
6
25
```

Following are source code examples to illustrate loading this file from the current directory in a Windows environment. Note that other operating systems may have different rules for naming files and directories.

C/C++

```
LONG   lRet;

lRet = DBGCLoadTimeouts(".\\TIMEOUT.DBG");
```

Visual Basic

```
Dim lRet As Long

lRet = DBGCLoadTimeouts(".\TIMEOUT.DBG");
```

DBGCMoveFirst

Declaration DBGCLIENTFUNC **DBGCMoveFirst**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks Moves to the first record in the workarea identified by *lWAHnd*, and must always be called first, before any move.

An error is returned if no records exist in the workarea's recordset.

> **Note:** This function must be preceded by a successful database query yielding one or more records.

Example Move to the first record in the workarea.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCMoveFirst(lNetConn, &lNetErr,
   lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCMoveFirst(lNetConn, lNetErr, lWA)
```

DBGCMoveLast

Declaration DBGCLIENTFUNC **DBGCMoveLast**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks

Moves to the last record in the workarea identified by *lWAHnd*.

An error is returned if no records exist in the workarea's recordset.

> **Note:** This function must be preceded by a successful database query yielding one or more records.

Example

Move to the last record in the workarea.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCMoveLast(lNetConn, &lNetErr, lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCMoveLast(lNetConn, lNetErr, lWA)
```

DBGCMoveNext

Declaration

DBGCLIENTFUNC **DBGCMoveNext**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks

Moves to the next record in the workarea identified by *lWAHnd*.

If an attempt to move beyond the end of the workarea's recordset is made, an error message is returned. Also, the last record in the recordset (if any) becomes the current record.

> **Note:** This function must be preceded by a successful database query yielding one or more records.

Example

Move to the next record in the workarea.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCMoveNext(lNetConn, &lNetErr, lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCMoveNext(lNetConn, lNetErr, lWA)
```

DBGCMovePrevious

Declaration DBGCLIENTFUNC **DBGCMovePrevious**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn Handle specifying the connection to the Server.

lplNetErr Pointer to a long to hold the error code reported by the network.

lWAHnd A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks Moves to the previous record in the workarea identified by *lWAHnd*.

If an attempt to move beyond the end of the workarea's recordset is made, an error message is returned. Also, the last record in the recordset (if any) becomes the current record.

> **Note:** This function must be preceded by a successful database query yielding one or more records.

Example Move to the previous record in the workarea.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCMovePrevious(lNetConn, &lNetErr,
   lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCMovePrevious(lNetConn, lNetErr,
   lWA)
```

DBGCOpenWorkarea

Declaration DBGCLIENTFUNC **DBGCOpenWorkarea**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LPLONG *lplWAHnd*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lplWAHnd	Pointer to a long integer to receive the handle of the newly opened workarea.

Remarks Opens (creates) a new workarea in the client API uniquely identified by the handle returned in the variable pointed to by *lplWAHnd*. This workarea is used on subsequent calls to the API to submit SQL statements to the pipe on the Server and to manipulate data retrieved from the database.

> **Note:** Although not required with version 5.0, it is recommend that a connection is established with the server before opening a workarea. Likewise, it is recommended that the workarea is closed before the connection with the server is terminated.

Example Create a new workarea, closing it immediately.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCOpenWorkarea(lNetConn, &lNetErr,
   &lWA);
lRet = DBGCCloseWorkarea(lNetConn, &lNetErr,
   lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCOpenWorkarea(lNetConn, lNetErr,
   lWA)
lRet = DBGCCloseWorkarea(lNetConn, &lNetErr,
   lWA)
```

DBGCReset

Declaration

DBGCLIENTFUNC **DBGCReset**(
LONG *lNetConn*,
LPLONG *lplNetErr*,
LONG *lWAHnd*);

Parameters

lNetConn	Handle specifying the connection to the Server.
lplNetErr	Pointer to a long to hold the error code reported by the network.
lWAHnd	A workarea handle returned on a previous call to **DBGCOpenWorkarea**.

Remarks

Resets the workarea clearing all records and other data returned from the Server.

> **Note:** Calling this function results in losing all data in the recordset.

Example Clear the workarea.

C/C++

```
LONG   lRet;
LONG   lNetConn;
LONG   lNetErr;
LONG   lWA;

lRet = DBGCReset(lNetConn, &lNetErr, lWA);
```

Visual Basic

```
Dim lRet As Long
Dim lNetConn As Long
Dim lNetErr As Long
Dim lWA As Long

lRet = DBGCReset(lNetConn, lNetErr, lWA)
```

DBGCSetCommTimeouts

Declaration DBGCLIENTFUNC **DBGCSetCommTimeouts**(
LONG *lSendWaitMax*,
LONG *lSendOpMax*,
LONG *lRecvWaitMax*,
LONG *lRecOpMax*);

Parameters

lSendWaitMax Maximum time in seconds that the network send routines wait for the socket to become available for writing

lSendOpMax Maximum time in seconds that the network send routines may take to execute an entire send operation.

lRecWaitMax Maximum time in seconds that the network receive routines wait for the socket to become available for reading

lRecvOpMax Maximum time in seconds that the network receive routines may take to execute an entire receive operation.

Remarks

Specifies the timeout values that are to be used by the native TCP/IP transport.

Values are divided into send and receive operational timeouts specified in seconds. When using the TCP/IP base transport, a typical API function call consists of a send operation, followed by a receive operation. The send operation sends instructions over the network to the DBG Server, while the receive operation waits on a response, also from the Server. If a send or receive operation times out, the function is aborted and an error is returned to the calling application. RPC connectivity is not affected by these timeouts

DBGCSetCommTimeouts can be called at any time during the execution of a program, but it is recommend that it be called once, at the beginning of the program (after **DBGCInitialize**, if applicable).

> **Note:** Timeout values are global to the API.
>
> A value less than one second will use the default values built into the API. These values are five seconds for waits and 30 seconds for operations.

Example

Specify timeouts for the TCP/IP transport. Send operations will wait for up to 5 seconds for the socket to become available, and send operations may not exceed 10 seconds. Receive operations will wait up to 6 seconds for the socket to become available and will timeout after 25 seconds.

C/C++

```
LONG lRet;

lRet = DBGCSetCommTimeouts(5, 10, 6, 25);
```

Visual Basic

```
Dim lRet As Long
```

```
lRet = DBGCSetCommTimeouts(5, 10, 6, 25)
```

DBGCShutdown

Declaration DBGCLIENTFUNC **DBGCShutdown**()

Parameters None

Remarks Relinquishes the DBG API library on non-Windows platforms.

This function must be called once, before exiting your application, to unload the DBG API library.

> **Caution:** Invoking this function in a Windows environment may cause unexpected results. The Windows implementation of the client API automatically initializes and destroys internal data structures when loading and unloading respectively.

Example Shutdown the API library in a non-Windows application.

C/C++

```
LONG   lRet;
lRet = DBGCShutdown();
```

Visual Basic

N/A – non-Windows environments only.

Appendix A. Data Structures and Constants

This appendix lists data structures and constants specific to Dialect Database Gateway.

Data Structures

DBG uses the following data structures to fill in date and time information returned by the database, or communicate additional execute request information:

- DBG_DATE
- DBG_REQ
- DBG_TIME
- DBG_TIMESTAMP

The next few pages provide details for each of these types of data structures.

DBG_DATE

Remarks

A structure filled in by the DBG Server for date fields returned by the database.

C/C++

```
typedef struct _tagDBG_DATE
{
      signed short      m_sYear;
      unsigned short    m_usMonth;
      unsigned short    m_usDay;
} DBG_DATE, FAR *LPDBG_DATE;
```

Visual Basic

```
Type DBG_DATE
      m_sYear As Integer
      m_usMonth As Integer
      m_usDay As Integer
End Type
```

DBG_REQ

Remarks

A structure used to communicate additional execute request information to **DBGCExecute**. Certain fields are filled in by the function call when it returns.

C/C++

```
typedef struct _tagDBG_REQ
{
      /* Filled in by caller */
      LONG  m_lRecsRequired;
      LONG  m_lBackground;

      /* Fill in by function */
      LONG  m_lODBCErrorCode;
      char  m_cODBCStatus[6];
      LONG  m_lNativeErrorCode;
      LONG  m_lRecsReturned;
      LONG  m_lColsReturned;
} DBG_REQ, FAR *LPDBG_REQ;
```

Visual Basic

```
Type DBG_REQ
      ' Filled in by caller
      m_lRecsRequired As Long
      m_lBackground As Long
      ' Fill in by function
      m_lODBCErrorCode As Long
      m_cODBCStatus As String * 6
      m_lNativeErrorCode As Long
      m_lRecsReturned As Long
      m_lColsReturned As Long
End Type
```

DBG_TIME

Remarks A structure filled in by the DBG Server for time fields returned by the database.

C/C++

```
typedef struct _tagDBG_TIME
{
      unsigned short    m_usHour;
      unsigned short    m_usMinute;
      unsigned short    m_usSecond;
} DBG_TIME, FAR *LPDBG_TIME;
```

Visual Basic

```
Type DBG_TIME
      m_usHour As Integer
      m_usMinute As Integer
      m_usSecond As Integer
End Type
```

DBG_TIMESTAMP

Remarks A structure filled in by the DBG Server for timestamp fields returned by the database.

C/C++

```
typedef struct _tagDBG_TIMESTAMP
{
      signed short      m_sYear;
      unsigned short    m_usMonth;
      unsigned short    m_usDay;
      unsigned short    m_usHour;
      unsigned short    m_usMinute;
      unsigned short    m_usSecond;
} DBG_TIMESTAMP, FAR *LPDBG_TIMESTAMP;
```

Visual Basic

```
Type DBG_TIMESTAMP
      m_sYear As Integer
      m_usMonth As Integer
      m_usDay As Integer
      m_usHour As Integer
      m_usMinute As Integer
      m_usSecond As Integer
End Type
```

Constants

DBG uses the following constants to define data names and sizes, specify default timeout values and network settings for TCP/IP functions, and define data types:

- Sizes
- Timeout values
- Miscellaneous network settings
- Data types.

The next few pages provide details for each of these types of constants.

Sizes

Table 6 that follows lists constants that define the maximum sizes for names and data.

Table 6. Constants and Data Sizes

Constant	Value	Description
DBGMAX_COLNAME_SIZE	33	Maximum size of a column name, including zero terminator.
DBGMAX_PIPENAME_SIZE	33	Maximum size of a pipe name, including zero terminator.

Network Timeouts

Table 7 that follows lists constants that define default timeout values for TCP/IP function operations. These timeouts may be changed using the **DBGCSetCommTimeouts** and **DBGCLoadCommTimeouts** API functions.

Table 7. Timeout Value Constants

Constant	Value	Description
MAXSENDWAITSECS	5	Maximum time, in seconds, that network send routines will wait for a socket to become available.
MAXSENDOPSECS	30	Maximum time, in seconds, that network send routines will take to complete an operation.
MAXRECVWAITSECS	5	Maximum time, in seconds, that network receive routines will wait for a socket to become available.
MAXRECVOPSECS	30	Maximum time, in seconds, that network receive routines will take to complete an operation.

Network Settings

Table 8 that follows lists constants that define various network settings and values that affect only TCP/IP functions. These values cannot be changed and are provided for informational purposes only.

Table 8. Constants and Network Settings

Constant	Value	Description
SLEEPFOR_PERIOD	1000	Period, in milliseconds, that network routines pause in between attempts to acquire a socket.
MAX_SR_RETRIES	5	Maximum number of transmission (sending or receiving) retries.

Data Types

Table 9 that follows lists constants that define ODBC data types as translated by the DBG Server software.

Table 9. Data Type Constants

Constant	Value	Description
DBG_DT_CHAR	1	Character data.
DBG_DT_DOUBLE	8	IEEE double-precision floating point.
DBG_DT_BIT	-7	A bit flag (usually used to represent Boolean values).
DBG_DT_STINYINT	-26	Signed tiny integer (8 bits).
DBG_DT_UTINYINT	-28	Unsigned tiny integer (8 bits).
DBG_DT_SSHORT	-15	Signed short integer (16 bits).
DBG_DT_USHORT	-17	Unsigned short integer (16 bits).
DBG_DT_SLONG	-16	Signed long integer (32 bits).

Constant	Value	Description
DBG_DT_ULONG	-18	Unsigned long integer (32 bits).
DBG_DT_FLOAT	7	IEEE single-precision floating point.
DBG_DT_BINARY	-2	Raw binary.
DBG_DT_DATE	9	Database date translated into a DBG_DATE structure.
DBG_DT_TIME	10	Database time translated into a DBG_TIME structure.
DBG_DT_TIMESTAMP	11	Database timestamp translated into a DBG_TIMESTAMP structure.

Appendix B. Condition Values

This appendix lists common condition values returned by the DBG Server or by the client API. Table 10 lists each condition by its (numeric) value, along with its description.

Table 10. Condition Values

Value	Condition	Description
0	ERR_DBG_NONE	No error.
1	ERR_DBG_INVALID_ARGS	Invalid arguments.
2	ERR_DBG_UNDEFINED_FUNC	Undefined function.
3	ERR_DBG_NET_STARTUP	Error initializing network transport.
4	ERR_DBG_NET_INVALID_NAME	Cannot find host server name.
5	ERR_DBG_NET_INVALID_SOCKET	Invalid network socket.
6	ERR_DBG_NET_CONNECT	Cannot connect to server.
7	ERR_DBG_NET_NOT_CONNECTED	Not currently connected to a server.
8	ERR_DBG_NET_RECV	Error receiving data over network.
9	ERR_DBG_NET_SEND	Error sending data over network.
10	ERR_DBG_ALREADY_EXISTS	The API or server object already exists.

Value	Condition	Description
11	ERR_DBG_NOMEM_SRV	The server cannot allocate memory.
12	ERR_DBG_NOMEM_CLT	The client cannot allocate memory.
13	ERR_DBG_BAD_CONNHND	Invalid connection handle.
14	ERR_DBG_NODATA	No records or columns.
15	ERR_DBG_NOPREV	No previous record or column.
16	ERR_DBG_NONEXT	No next record or column.
17	ERR_DBG_INVALIDBGEC	Invalid current record.
18	ERR_DBG_INVALIDCOL	Invalid column name.
19	ERR_DBG_NOESTABCONN	The connection is not established.
20	ERR_DBG_ODBC_ALLOCSTMT	An error was encountered during SQLAllocStmt.
21	ERR_DBG_ODBC_EXECDIRECT	An error was encountered during SQLExecDirect.
22	ERR_DBG_ODBC_BINDCOL	An error was encountered during SQLBindCol.
23	ERR_DBG_ODBC_NUM-RESULTCOLS	An error was encountered during SQLNumResultCols.
24	ERR_DBG_ODBC_COLAT-TRTYPE	An error was encountered during SQLColAttributes(SQL_COLUMN_TYPE).
25	ERR_DBG_ODBC_COLATTRLEN	An error was encountered during SQLColAttributes(SQL_COLUMN_LENGTH).

Value	Condition	Description
26	ERR_DBG_ODBC_COLATTRNAME	An error was encountered during SQLColAttributes(SQL_COLUMN_NAME).
27	ERR_DBG_ODBC_FETCH	An error was encountered during SQLFetch.
28	ERR_DBG_INVALIDCOLTYPE	SQLColAttributes(SQL_COLUMN_TYPE) returned an invalid type.
29	ERR_DBG_INVALIDPIPENAME	The pipe name is invalid.
30	ERR_DBG_UNSUPPORTEDTYPE	The datatype is not supported by the Server.
31	ERR_DBG_CONNAVAILTIMEOUT	A timeout occurred while attempting to find an available connection.
32	ERR_DBG_BROKENPIPE	A broken pipe was detected.
33	ERR_DBG_NOTFOUND	The object was not found.
34	ERR_DBG_BLOCK_SIGNAL	The blocking handle was signaled.
35	ERR_DBG_BLOCK_TIMEOUT	The blocking handle timed out.
36	ERR_DBG_ODBC_PREPARE	An error was encountered during SQLPrepare.
37	ERR_DBG_ODBC_BINDPARAM	An error was encountered during SQLBindParam.
38	ERR_DBG_ODBC_EXECUTE	An error was encountered during SQLExecute.
39	ERR_DBG_ODBC_SETSTMTOPTION	An error was encountered during SQLSetStmtOption.
40	ERR_DBG_ODBC_GETDATA	An error was encountered during SQLGetData.

Value	Condition	Description
41	ERR_DBG_INVALID_CONN-TYPE	Invalid connection type.
42	ERR_DBG_INVALID_WORK-AREA	Invalid workarea.
996	ERR_SVC_INACTIVE	Service is not in an active state.
997	ERR_DBG_EXCEPT	The connection handle for an RPC connection is invalid or a general network exception occurred.
998	ERR_DBG_INVALID_CONN_-HANDLE	The connection handle is invalid.
999	ERR_DBG_SIGNON	Cannot sign on to the Server.

Appendix C. The mivrdbg User Function

This appendix describes the mivrdbg User Function for the Meridian Integrated IVR 2.0, Meridian OPEN IVR 2.0, and Symposium OPEN IVR 4.0.

Overview

The mivrdbg process is an interface to the DBG API for SCO UNIX, which is implemented as an IVR User Function. Using the interface, an IVR script can gain access to the most commonly used DBG API functions, including the ability to communicate with any database supporting the ODBC standard.

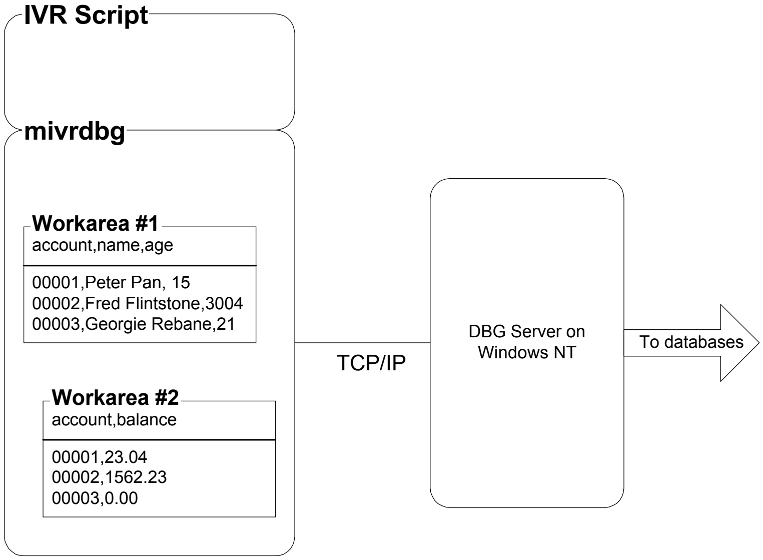
SQL Support

DBG does not perform any special processing on SQL statements, and acts as a pass-through system for SQL requests. A successful execution will occur as long as the SQL statement submitted to DBG (and passed on to the database) is valid for that database.

Workareas

The DBG API introduces the concept of a workarea where all data and status information regarding a SQL session is stored. After opening a workarea (using the **OpenWorkarea** user function described on page 112), a script may execute SQL statements, retrieving data returned by the server. A script may open as many workareas as needed, combining data from multiple back-end databases.

> **Note:** When a SQL statement is executed, the workarea is cleared of both data and status information.



```
account,name,age
00001,Peter Pan, 15
00002,Fred Flintstone,3004
00003,Georgie Rebane,21
```

```
account,balance
00001,23.04
00002,1562.23
00003,0.00
```

Figure 14. Concept of a Workarea

Background Processing

DBG also supports background processing, which enables a client to submit a SQL for processing even when the Server is idle. This functionality is especially useful in those circumstances where the outcome of a SQL statement is not important to the client. For example, a typical example of background processing is a client that wishes to perform an operation that may take some time to complete, such as a bulk update. In this case, the client can request that the Server process the statement during an idle period; the function call into the Server immediately returns as the system will not wait for the statement to complete.

Configuration

The mivrdbg process requires an initialization file (mivrdbg.ini) that specifies the name of the Windows NT computer hosting the DBG Server. This initialization file must be located in the same directory as the mivrdbg executable. For Symposium and Meridian OPEN IVRs, this is typically the `/u/nortel/exe` directory, while for both Symposium and Meridian Integrated IVRs, the directory is typically `/u/ivr/exe`.

The first line of the file must contain the DNS name of the host or its IP address for the DBG Server.

The second, optional, line of the initialization file specifies whether the type of connection the User Function should maintain between the IVR and the DBG Server. Leave this line blank or specify "**dynamic connections**" (the default) to have the User Function close the previous connection, if any, and establish a new one each time an interface function is called. Alternatively, you can specify "**static connections**" to maintain static connections.

Following are two examples of the mivrdbg.ini file. Both use the DBG Server hosted on the **corp_srv** Windows NT computer. The first initialization file specifies that the User Function should maintain static connections, the second indicates that a new connection is to be established each time an interface function is called (the previous connection, if any, will be closed before a new connection is opened).

Example 1

```
corp_srv
static connections
```

Example 2

```
corp_srv
dynamic connections
```

Communication

Communication between the IVR and the DBG Server is handled over TCP/IP with a new connection made each time a SQL statement is executed.

If you plan to use the DNS name of the DBG Server host computer, you must also define that computer's name in a DNS Server that the IVR has been configured to use. Alternatively, you may place the name of the host and its IP address in the `/etc/hosts` file of the IVR system if no DNS Server is available or configured.

Refer to your SCO UNIX documentation for information on configuring for DNS Servers and/or maintenance of the `/etc/hosts` file.

Interface Functions

In all, there are nine interface functions:

- OpenWorkarea
- CloseWorkarea
- ExecuteSQL
- MoveFirst
- MoveLast
- MoveNext
- MovePrevious
- GetColumnData
- ExecuteSQLFromFile

Details for each of these interface functions follow, after an explanation for return values and data types an sizes.

Return Values

All mivrdbg functions return two sets of return values that may be used by the IVR script for branching purposes:

- A numeric return code of 0 or 1 indicating success or failure respectively.
- A string in buffer 1 containing a more detailed description of the error encountered (if any).

If a function call is successful, buffer 1 will contain the word `SUCCESS`. If the function call fails, buffer 1 will contain the word `FAIL_xxxxx` where `xxxxx` is a short description of the error.

The primary reason for the string return code in buffer 1 is to ease debugging of IVR scripts using a tool such as the IVR's `sam` utility.

Data Types and Sizes

The current release of mivrdbg only supports data strings. The length of data stored and retrieved is limited to the buffer size allowed by the IVR Application Editor (`xae`).

OpenWorkarea

Function Code 51

Inputs

Buffer	Description
none	None

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code
2	Workarea number (WA)

Return Codes

Value	Description
0	Success
1	Failure

Remarks Open a local workarea.

CloseWorkarea

Function Code 52

Inputs

Buffer	Description
0	Workarea number (WA)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Return Codes

Value	Description
0	Success
1	Failure

Remarks Close a previously opened local workarea.

ExecuteSQL

Function Code 53

Inputs

Buffer	Description
0	Workarea number (WA)
1	Pipe name
2	Legal SQL statement
3	Max records required for return
4	Background operation (Y=Yes; N=No); No records returned if Yes.

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code
2	ODBC error code (Consult your ODBC documentation.)
3	ODBC status (Consult your ODBC documentation.)
4	Native error code (Consult your driver documentation.)
5	Records returned
6	Columns returned

Return Codes

Value	Description
0	Success
1	Failure

Remarks Execute a SQL statement on the server.

MoveFirst

Function Code 54

Inputs

Buffer	Description
0	Workarea number (WA)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Return Codes

Value	Description
0	Success
1	Failure

Remarks Move to the first record in the workarea.

MoveLast

Function Code 55

Inputs

Buffer	Description
0	Workarea number (WA)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Return Codes

Value	Description
0	Success
1	Failure

Remarks Move to the last record in the workarea.

MoveNext

Function Code 56

Inputs

Buffer	Description
0	Workarea number (WA)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Return Codes

Value	Description
0	Success
1	Failure

Remarks Move to the next record in the workarea.

MovePrevious

Function Code 57

Inputs

Buffer	Description
0	Workarea number (WA)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Return Codes

Value	Description
0	Success
1	Failure

Remarks Move to the previous record in the workarea.

GetColumnData

Function Code 58

Inputs

Buffer	Description
0	Workarea number (WA)
1	Column name
2	C-style format string (optional)

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code
2	Column data (up to 30 characters)
3	Remainder of data up to MAX_BUFS

Return Codes

Value	Description
0	Success
1	Failure

Remarks

Retrieve a column's data from the current record.

If a third parameter (#2) is provided, mivrdbg will format the data using the C-style format string specified. You must ensure that the format string you provide conforms to C-style `printf()` format strings and that it is valid for the data type which is being returned by the database. See below for a discussion of how the function formats data.

> **Note:** If parameter #2 is not specified, mivrdbg will retrieve the raw data from the workarea. In this case, the data must be a string data type or unpredictable results will occur.

Data Formatting

The software routing that is responsible for formatting data in mivrdbg ultimately uses the C runtime library functions for formatting of data. While all data is retrieved from the DBG record buffer in binary format, it is converted to the proper type before being passed to the C runtime library routines along with the format string specifying how the data is to be formatted in the buffer returned to Generations.

Format	Description
Strings	String data returned from the database is null terminated, and can therefore be directly processed by the C runtime library without conversion. Format strings related to string data (the "**%s**" family) may be used to format string data. For example, the string "**Hello world**" may be formatted using "**%s**" resulting in "**Hello world**"
Single precision floating point	Single precision floating numbers are processed using the format strings for floating point numbers. For example, the binary double value **881.2210000** may be formatted using "**%8.3f**" resulting in "**881.221**".
Double precision floating point	Double precision floating numbers are processed using the format strings for floating point numbers. For example, the binary double value **123.4500000** may be formatted using "**%7.2f**" resulting in "**123.45**".

Format	Description
Bit fields	Bit fields are usually used to represent Boolean values such as TRUE and FALSE. The mivrdbg process converts bit fields to a single character representing of each ("**Y**" and "**N**"). A value of 0 is translated as "**N**"; all other values are translated as "**Y**". Format bit fields using string type format strings. For example, A bit field of 0 may be formatted using "**%s**" resulting in "**N".** A bit field of 1 may be formatted using "**%s**" resulting in "**Y**".
Signed tinyint	Signed tinyint values are converted to signed shorts. See that discussion below.
Unsigned tinyint	Unsigned tinyint values are converted to unsigned shorts. See that discussion below.
Signed short	Signed short numbers are represented as signed 16-bit integers by DBG. Format strings related to signed short values (such as "**%d**") may be used. For example, the signed short value **-62** may be formatted using "**%d**" resulting in "**-62**."
Unsigned short	Unsigned short numbers are represented as unsigned 16-bit integers by DBG. Format strings related to unsigned short values (such as "**%u**") may be used. For example, the unsigned short value **37** may be formatted using "**%u**" resulting in "**37**."

Format	**Description**
Signed long	Signed long numbers are represented as signed 32-bit integers by DBG. Format strings related to signed long values (such as "**%ld**") may be used. For example, the signed long value **-78291** may be formatted using "**%ld**" resulting in "**-78291**."
Unsigned long	Unsigned long numbers are represented as unsigned 32-bit integers by DBG. Format strings related to unsigned long values (such as "**%lu**") may be used. For example, the unsigned long value **561281** may be formatted using "**%lu**" resulting in "**561281**."
Dates	Dates retrieved from a database are converted to a database independent. A date consists of a month, day, and year (inclusive of century), each being a 16-bit unsigned integer. The formatting routing for a date passes the month, date, and year portions of the date in that order to the runtime library for processing - your format string must process all three these parameters For example, assuming a date of Dec 1, 1997, the following parameters are passed "**12**", "**1**", "**1997**". A format string such as "**%02u/%02u/%04u**" will result in "**12/01/1997**."

Format	**Description**
Times	Times retrieved from a database are converted to a database independent. Time, which is represented in military time, consists of a hour, minute, and second, each being a 16-bit unsigned integer. The formatting routing for a date passes the hour, minute, and second portions of the time in that order to the runtime library for processing: your format string must process all three these parameters. For example, assuming a time of 25 minutes and 3 seconds after 1PM, the following parameters are passed "**13**", "**25**", "**3**". A format string such as "**%02u:%02u:%02u**" will result in "**13:25:03**."

Format	Description
Timestamps	Timestamps fields consist of a date and time portion, the time portion having an additional field of *fraction* indicating the number of milliseconds (thousandths of a second) between seconds. Fractions range from 0 to 999. Time is represented in military time. Each field is represented as 16-bit unsigned integer with the exception of the fraction field, which is represented as a 32-bit unsigned long. The formatting routing for a timestamp passes the month, day, year, hour, minute, second, and fraction portions of the timestamp in that order to the runtime library for processing - your format string must process all seven these parameters. For example, assuming a timestamp of 25 minutes, 3 seconds, and 260 milliseconds after 1:00 PM on Christmas day, 1997, the following parameters are passed "**12**", "**25**", "**1997**", "**13**", "**25**", "**3**", "**260**". A format string such as "**%02u:%02u:%04u-%02u:%02u:%02u:%03u**" will result in "**12/25/1997-13:25:03:260**."

> **Note:** The user function will split a column's data across the output buffers if the data cannot fit into a single buffer. The number of buffers the data is split across and the size of each buffer is dependent on the release of the IVR.

ExecuteSQLFromFile

Function Code 59

Inputs

Buffer	Description
0	Workarea number (WA)
1	File name. Full path specification with file structure as follows: • xxxxxx – pipe name • 999999 – max records to return • Y or N – background operation (Y=Yes; N=No) • ssss… - SQL line 1 • ssss… - SQL line 2 • ssss… - SQL line n (up to max of 8192 bytes)
2	Param1 (optional) - @1@ in SQL statement
3	Param2 (optional) - @2@ in SQL statement
4	Param3 (optional) - @3@ in SQL statement
5	Param4 (optional) - @4@ in SQL statement
6	Param5 (optional) - @5@ in SQL statement
7	Param6 (optional) - @6@ in SQL statement
8	Param7 (optional) - @7@ in SQL statement
9	Param8 (optional) - @8@ in SQL statement

Outputs

Buffer	Description
0	SUCCESS or FAIL_xxxx
1	DBG error code

Buffer	Description
2	ODBC error code (Consult your ODBC documentation.)
3	ODBC status (Consult your ODBC documentation.)
4	Native error code (Consult your driver documentation.)
5	Records returned
6	Columns returned

Return Codes

Value	Description
0	Success
1	Failure

Remarks

Similar to function 53 (**ExecuteSQL**); the difference being that parameters are loaded from a file rather than passed via the buffers.

> **Note:** The SQL statement in the file may consist of multiple lines, each up to a maximum of 256 characters per line. The user function will concatenate successive lines from the file into a single SQL statement until the end of the file is reached.

Example SQL File

Example showing a statement that retrieves a single caller record from the customer table in the *MyDatabasePipe* based on either entered social security number or home telephone number.

```
MyDatabasePipe

1

N
```

```
select name, addr1, addr2, city, state, zip
from cust

where ssn = '@1@' or home_tel = '@2@'
```

Appendix D. DBG UserDLL for InterVoice IVR

This appendix describes the DLL for the InterVoice IVR.

Overview

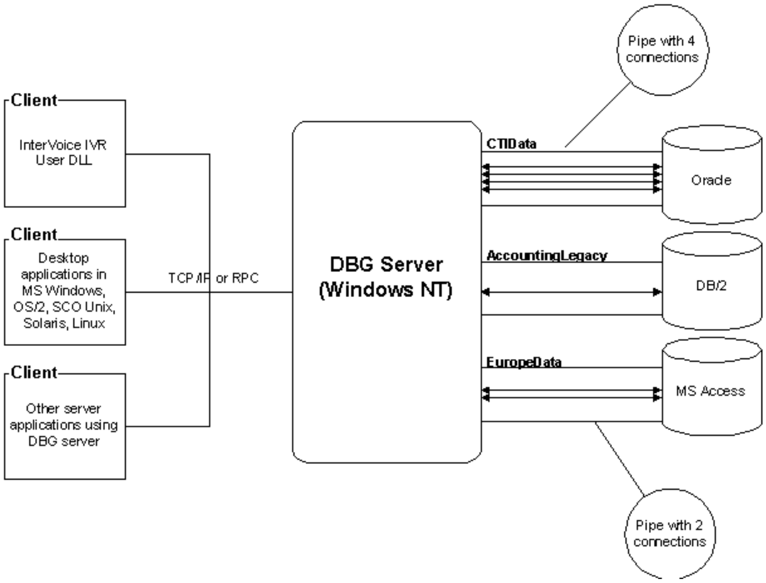
DBG now includes a UserDLL that that facilitates communication between Database Gateway (DBG) and InterVoice's Interactive Voice Response (IVR) platform.

InterVoice IVR scripts can use this UserDLL to access most Database Gateway functions and its ability to any database that supports the ODBC standard.

Database support

DBG supports any database system that provides a 32-bit ODBC driver for Windows NT. This list includes (but is not limited to) Sybase, Oracle, Informix, DB/2, Microsoft SQL Server, and Microsoft Access, Microsoft FoxPro, dBase, Microsoft Excel, IBM AS/400, text files, Sybase SQL Anywhere, Paradox, and RUMBA DRDA-32.

The following figure depicts a system in which the DBG API accesses three database systems from a variety of clients. Although this example includes only Oracle, DB/2 and Microsoft Access, DBG can handle any database that includes an ODBC interface for Windows NT.

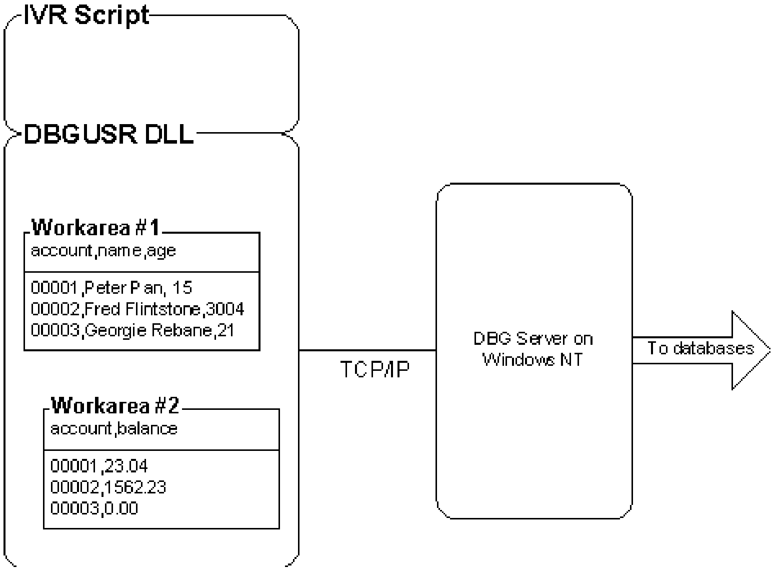


SQL support

DBG acts as a "pass-through" system for SQL requests; in other words, it performs no special processing on SQL statements. As long as the SQL statement submitted to DBG (and passed on to the database) is valid for that database, execution will succeed.

Workareas

The DBG API uses a virtual "workarea" to store all data and status information about an SQL session. After using the *OpenWorkarea* user function to open a workarea, a script may execute SQL statements to retrieve data returned by the server. A script may open as many workareas as needed, combining data from multiple back-end databases.



Background Processing

Background processing is a feature of DBG that allows a client to submit a SQL statement for processing when the server is idle. This functionality is especially useful in those situations when the outcome of a SQL statement is not important to the client. For example, consider a client that wants to perform a lengthy operation like a bulk update. In this case, the client can ask that the server process the statement during an idle period. The function call into the server returns immediately; the system does not wait for the statement to complete.

Communication

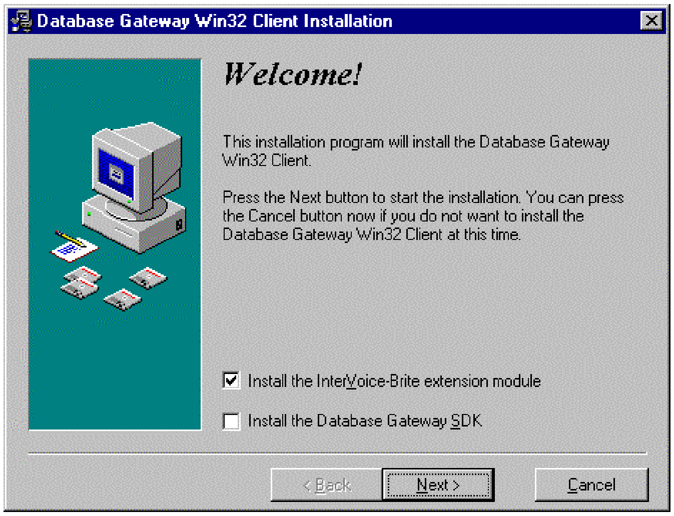
A global connection between the InterVoice IVR and the DBG server is maintained over RPC with TCP/IP as the transportation layer. A reconnection procedure begins automatically after any network error that indicates a broken connection between the IVR and the DBG server.

Installation and Configuration

If you chose not to install the DBG UserDLL when you first installed the Win 32-bit DBG client, you can install it now by following these steps:

1. Launch the Win 32-bit DBG client installation program setup.exe.

   "The Welcome window" appears (Figure 15).



Database Gateway Win32 Client Installation

Welcome!

This installation program will install the Database Gateway Win32 Client.

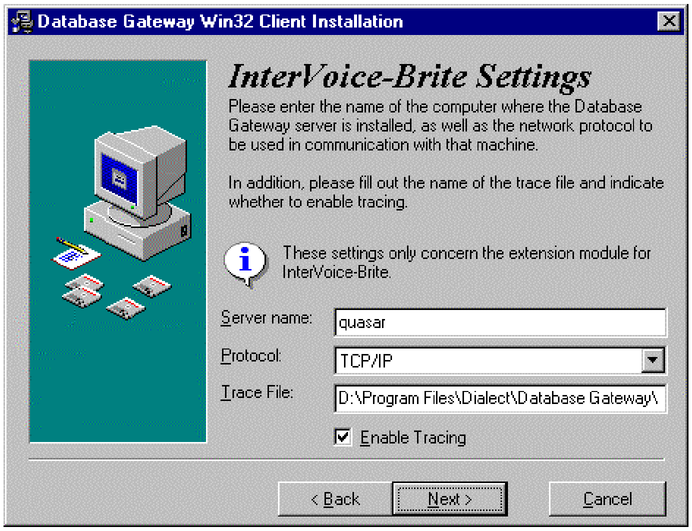
Press the Next button to start the installation. You can press the Cancel button now if you do not want to install the Database Gateway Win32 Client at this time.

☑ Install the InterVoice-Brite extension module

☐ Install the Database Gateway SDK

Figure 15: The Welcome window

2. Choose the option "Install the InterVoice-Brite extension module" and click **Next >**.
3. After you agree to the license agreement and choose a destination directory for the client files, the "The Configuration window" appears (Figure 16).



Database Gateway Win32 Client Installation

InterVoice-Brite Settings

Please enter the name of the computer where the Database Gateway server is installed, as well as the network protocol to be used in communication with that machine.

In addition, please fill out the name of the trace file and indicate whether to enable tracing.

These settings only concern the extension module for InterVoice-Brite.

Server name: quasar

Protocol: TCP/IP

Trace File: D:\Program Files\Dialect\Database Gateway\

☑ Enable Tracing

Figure 16: The Configuration window

4. Complete the following fields:

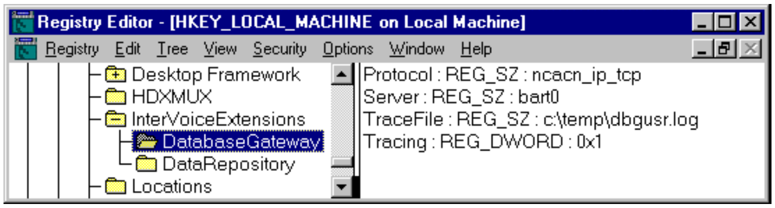
   Server name: the name of the NT host machine operating the DBG server.

   > **Note**: This host may be the same as the IVR.

   Protocol: The transfer protocol required for communication between the InterVoice-Brite extension and the DBG server. Valid choices include Local RPC (if both DBG and the extension reside on the same computer), TCP/IP, Named Pipes, IPX, and SPX.

   Trace File Name: the location and name of the log file to create.

   As illustrated in Figure 17, the setup program will modify the registry settings under *HKEY_LOCAL_MACHINE SOFTWARE\Williams Telecommunications\ InterVoiceExtensions\DatabaseGateway*.


Registry Editor - [HKEY_LOCAL_MACHINE on Local Machine]
Registry Edit Tree View Security Options Window Help
Desktop Framework
HDXMUX
InterVoiceExtensions
DatabaseGateway
DataRepository
Locations

Protocol : REG_SZ : ncacn_ip_tcp
Server : REG_SZ : bart0
TraceFile : REG_SZ : c:\temp\dbgusr.log
Tracing : REG_DWORD : 0x1

Figure 17: dbgusr Registry settings

Troubleshooting

The log file can help you troubleshoot any problems that occur. By analyzing its timestamped records of API accesses, you can easily determine when the DBG UserDLL was loaded and unloaded, what particular functions were called and when, and the results of these function calls.

You can choose to "enable tracing" during installation or you can enable tracing later by changing the registry key *HKEY_LOCAL_MACHINE SOFTWARE\Williams Telecommunications\InterVoiceExtensions\DatabaseGateway \Tracing* to 1. The name of the log file is stored under *HKEY_LOCAL_MACHINE SOFTWARE\Williams Telecommunications\InterVoiceExtensions\DatabaseGateway \TraceFile.*

Developing UserDLL forms in InterVoice InVision

You can add DBG UserDLL functions to your IVR applications by dragging-and-dropping UserDLL forms into the InterVoice InVision development workspace and then typing values in the appropriate parameter fields. Located in the %WINNT%/system32 directory, the DBG UserDLL is named "dbgusr.dll," but you need only to enter *dbgusr* in the DLL name field.

You must specify variable names for the output of DBG UserDLL functions. Be sure to prefix each name with a "greater than" sign (>), to identify it as a "pass-by-reference" parameter. Failure to include this prefix can cause unexpected results.

Figure 18 contains a simple application.

> **Note**: To avoid memory leaks, close the workarea as soon as you no longer need it.

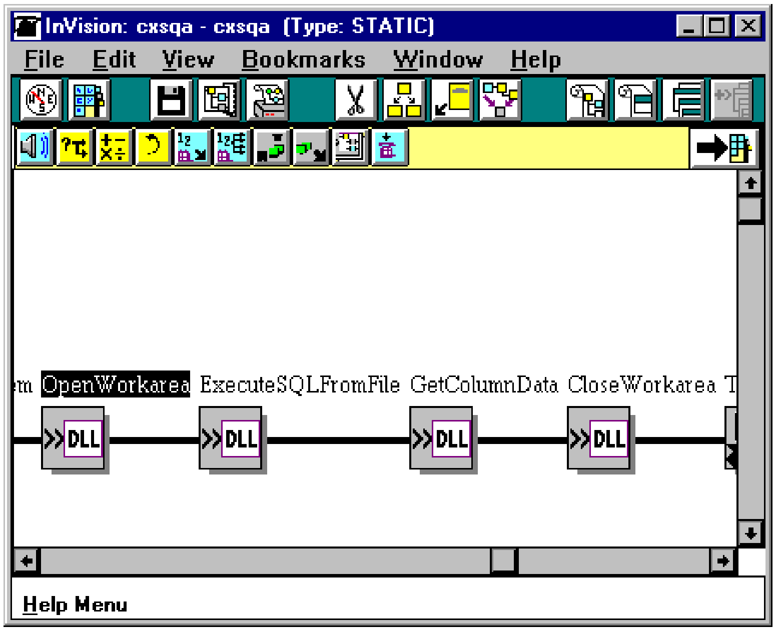

InVision: cxsqa - cxsqa (Type: STATIC)
File Edit View Bookmarks Window Help
m OpenWorkarea ExecuteSQLFromFile GetColumnData CloseWorkarea T
DLL
DLL
DLL
DLL
Help Menu

Figure 18: Sample InVision application

Interface functions

OpenWorkarea

Remarks

- Open a local workarea

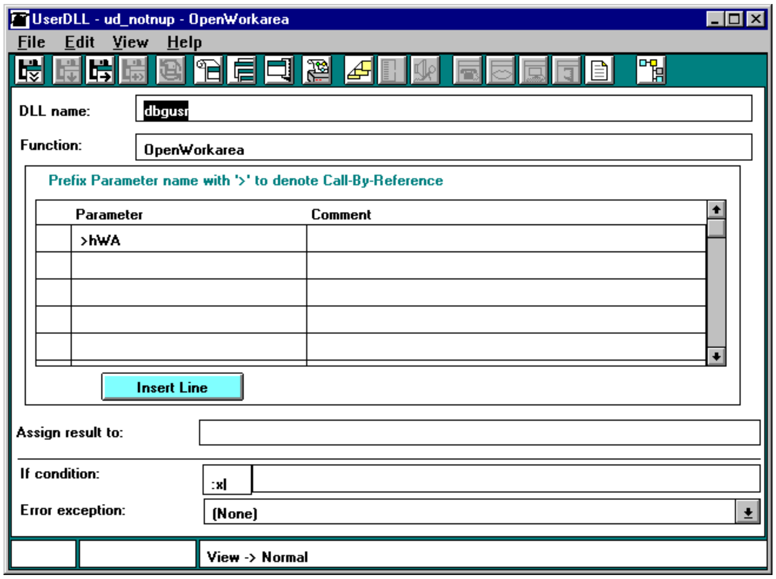
Parameters

Name	Type	In/Out	Description
hWa	Integer	Output	The handle of workarea

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

Example



DLL name: dbgusr

Function: OpenWorkarea

Prefix Parameter name with '>' to denote Call-By-Reference

Parameter	Comment
>hWA	

Assign result to:

If condition: :xl

Error exception: (None)

View -> Normal

CloseWorkarea

Remarks

- Closes a previously opened local workarea

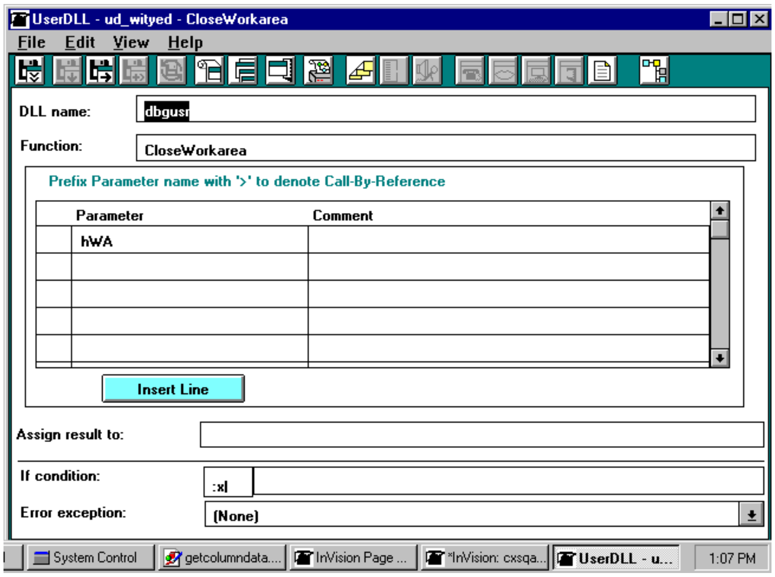
Parameters

Name	Type	In/Out	Description
hWa	Integer	Input	The handle of previously opened workarea

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

Example



DLL name: dbgusr

Function: CloseWorkarea

Prefix Parameter name with '>' to denote Call-By-Reference

Parameter	Comment
hWA	

Assign result to:

If condition: :x|

Error exception: (None)

ExecuteSQL

Remarks

- Executes a SQL statement on the server

Parameters

Name	Type	In/Out	Description
hWa	Integer	Input	Workarea handle obtained from OpenWorkarea function.
pipeName	String	Input	Pipe name.

sql	String	Input	Legal SQL statement.
maxRecs	String	Input	Max records required for return.
backgrd	Integer	Input	Background operation (1 = Yes, 0 = No).
recsRtn	Integer	Output	Records returned for the SQL query.
colRtn	Integer	Output	Columns returned for the SQL query.

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

MoveFirst

Remarks

- Moves to the first record in the workarea

Parameters

Name	**Type**	**In/Out**	**Description**
HWa	Integer	Input	Workarea handle obtained from OpenWorkarea function

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504

ERR_CANNOT_OPEN_REGISTRY_KEY	505

MoveLast

Remarks

- Moves to the last record in the workarea

Parameters

Name	Type	In/Out	Description
HWa	Integer	Input	Workarea handle obtained from OpenWorkarea function

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

MoveNext

Remarks

- Moves to the next record in the workarea

Parameters

Name	Type	In/Out	Description
HWa	Integer	Input	Workarea handle obtained from OpenWorkarea function

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504

ERR_CANNOT_OPEN_REGISTRY_KEY	505

MovePrevious

Remarks

- Moves to the previous record in the workarea

Parameters

Name	Type	In/Out	Description
HWa	Integer	Input	Workarea handle obtained from OpenWorkarea function

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

GetColumnData

Remarks

- Retrieves a column's data from the current record

  If a third parameter ***format*** is provided, DBG UserDLL will format the data using the C-style format string specified. You must ensure that the format string you provide conforms to C-style printf() format strings and that it is valid for the data type which is being returned by the database.

  > **Note**: If parameter **format** is not specified, DBG UserDLL will retrieve the raw data from the workarea. In this case the data must be a string data type or unpredictable results will occur.

Parameters

Name	Type	In/Out	Description
HWa	Integer	Input	Workarea handle obtained from OpenWorkarea function
ColName	String	Input	Column name
Format	String	Input	**(OPTIONAL)** C-style format string, discussed in detail in "Data Formatting" below.
Data	String	Output	Data returned

Data Formatting

Strings

String data returned from the database is null terminated, and can therefore be directly processed by the C runtime library without conversion. Format strings related to string data (the "**%s**" family) may be used to format string data.

For example: The string "**Hello world**" may be formatted using "**%s**" resulting in "**Hello world**"

Single precision floating point

Single precision floating numbers are processed using the format strings for floating point numbers.

For example: The binary double value **881.2210000** may be formatted using "**%8.3f**" resulting in "**881.221**".

Double precision floating point

Double precision floating numbers are processed using the format strings for floating point numbers.

For example: The binary double value **123.4500000** may be formatted using "**%7.2f**" resulting in "**123.45**".

Bit fields

Bit fields are usually used to represent boolean values such as TRUE and FALSE. DBG UserDLL converts bit fields to a single character representing of each ("**Y**" and "**N**"). A value of 0 is translated as "**N**", all other values are translated as "**Y**". Format bit fields using string type format strings.

For example: A bit field of 0 may be formatted using "**%s**" resulting in "**N**". A bit field of 1 may be formatted using "**%s**" resulting in "**Y**".

Signed tinyint

Signed tinyint values are converted to signed shorts. See below.

Unsigned tinyint

Unsigned tinyint values are converted to unsigned shorts. See that discussion below.

Signed short

Signed short numbers are represented as signed 16-bit integers by DBG. Format strings related to signed short values (such as "**%d**") may be used.

For example: The signed short value **-62** may be formatted using "**%d**" resulting in "**-62**".

Unsigned short

Unsigned short numbers are represented as unsigned 16-bit integers by DBG. Format strings related to unsigned short values (such as "**%u**") may be used.

For example: The unsigned short value **37** may be formatted using "**%u**" resulting in "**37**".

Signed long

Signed long numbers are represented as signed 32-bit integers by DBG. Format strings related to signed long values (such as "**%ld**") may be used.

For example: The signed long value **-78291** may be formatted using "**%ld**" resulting in "**-78291**".

Unsigned long

Unsigned long numbers are represented as unsigned 32-bit integers by DBG. Format strings related to unsigned long values (such as "**%lu**") may be used.

For example: The unsigned long value **561281** may be formatted using "**%lu**" resulting in "**561281**".

Dates

Dates retrieved from a database are converted to a database independent. A date consists of a month, day, and year (inclusive of century), each being a 16-bit unsigned integer. The formatting routing for a date passes the month, date, and year portions of the date in that order to the runtime library for processing - your format string must process all three these parameters

For example: Assuming a date of Dec 1, 1997, the following parameters are passed "**12**", "**1**", "**1997**". A format string such as "**%02u/%02u/%04u**" will result in "**12/01/1997**".

Times

Times retrieved from a database are converted to a database independent. A time consists of a hour, minute, and second, each being a 16-bit unsigned integer. The formatting routing for a date passes the hour, minute, and second portions of the time in that order to the runtime library for processing - your format string must process all three these parameters. *Note that time is represented in military time.*

For example: Assuming a time of 25 minutes and 3 seconds after 1pm, the following parameters are passed "**13**", "**25**", "**3**". A format string such as "**%02u:%02u:%02u**" will result in "**13:25:03**".

Timestamps

Timestamps fields consist of a date and time portion, the time portion having an additional field of *fraction* indicating the number of milliseconds (thousandths of a second) between seconds. Fractions range from 0 to 999.

Each field is represented as 16-bit unsigned integer with the exception of the fraction field which is represented as a 32-bit unsigned long.

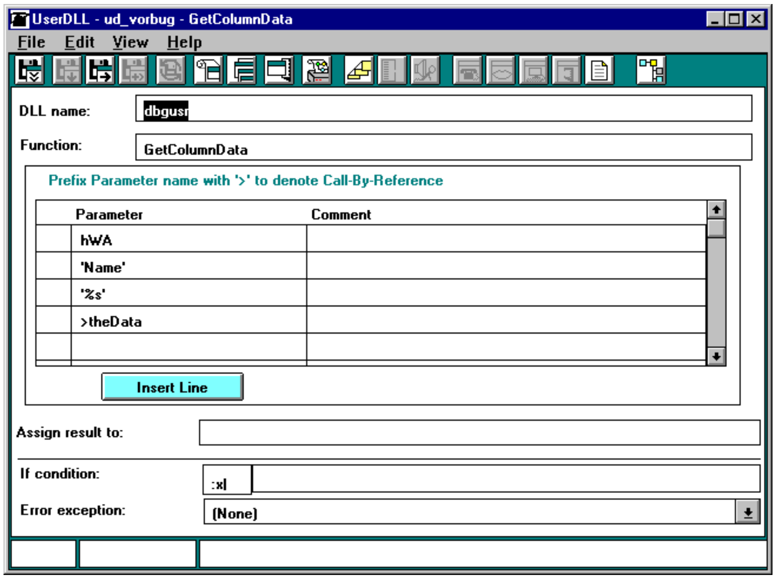
The formatting routing for a timestamp passes the month, day, year, hour, minute, second, and fraction portions of the timestamp in that order to the runtime library for processing - your format string must process all seven these parameters. *Note that time is represented in military time.*

For example: Assuming a timestamp of 25 minutes, 3 seconds, and 260 milliseconds after 1pm on Christmas day, 1997, the following parameters are passed "**12**", "**25**", "**1997**", "**13**", "**25**", "**3**", "**260**". A format string such as "**%02u:%02u:%04u-%02u:%02u:%02u:%03u**" will result in "**12/25/1997-13:25:03:260**"

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

Example



DLL name: dbgusr

Function: GetColumnData

Prefix Parameter name with '>' to denote Call-By-Reference

Parameter	Comment
hWA	
'Name'	
'%s'	
>theData	

Assign result to:

If condition: :x|

Error exception: (None)

ExecuteSQLFromFile

Remarks

- Similar to ExecuteSQL function, with the exception that parameters are loaded from a file rather than passed via the buffers.

Parameters

Name	Type	In/Out	Description
hWa	Integer	Input	Workarea handle obtained from OpenWorkarea function
fileName	String	Input	File name (full path specification). The file contains text, structured as follows:
		xxxxxx	Pipe name
		999999	Max records to return

		Y or N	Background operation (Y = Yes, N = No)
		ssss1...	SQL line 1
		ssss2...	SQL line 2
			SQL line n (up to total of 8192 bytes)
recsRtn	Integer	Output	Number of record returned for the executed SQL
colsRtn	Integer	Output	Number of columns returned for the executed SQL
param1	String	Input	[optional] - @1@ in SQL statement.
Param2	String	Input	[optional] - @2@ in SQL statement
Param3	String	Input	[optional] - @3@ in SQL statement
Param4	String	Input	[optional] - @4@ in SQL statement
Param5	String	Input	[optional] - @5@ in SQL statement
Param6	String	Input	[optional] - @6@ in SQL statement

Returns

SUCCESS	0
ERR_FAILED	501
ERR_INVALID_VARIABLE_TYPE	502
ERR_INVALID_FORMAT	503
ERR_FAILED_TO_CONNECT	504
ERR_CANNOT_OPEN_REGISTRY_KEY	505

Example

The following example file shows a statement that retrieves a single caller record from the customer table in the *MyDatabasePipe* based on either entered social security number or home telephone number.

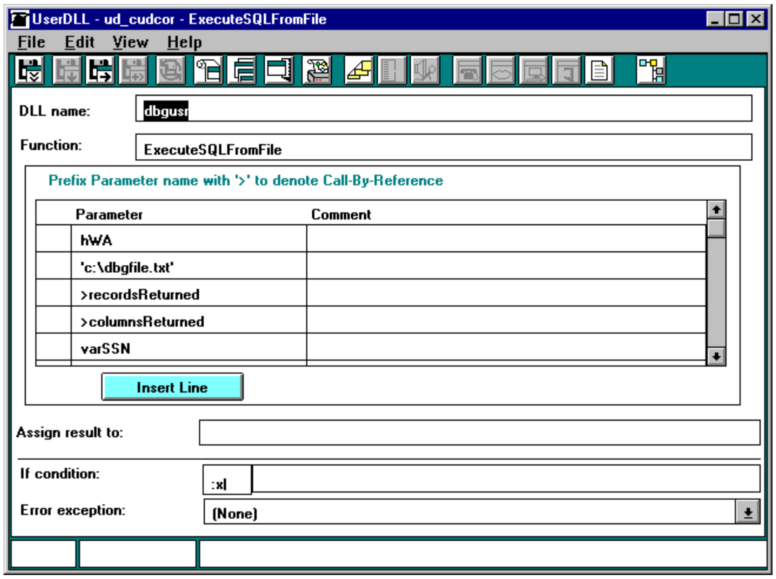
```
MyDatabasePipe

1

N

select name, addr1, addr2, city, state, zip
from cust

where ssn = '@1@' or home_tel = '@2@'
```



DLL name: dbgusr

Function: ExecuteSQLFromFile

Prefix Parameter name with '>' to denote Call-By-Reference

Parameter	Comment
hWA	
'c:\dbgfile.txt'	
>recordsReturned	
>columnsReturned	
varSSN	

Assign result to:

If condition: :xl

Error exception: (None)

Appendix E. Licensing

The Williams License Manager (License Manager) is a multi-purpose system that works in conjunction with the hardware license key to provide licensing services to the Dialect family of products, including Database Gateway. Together, the License Manager and the hardware license key:

- Ensure that you have the legal right to use your purchased product.
- Enable you to run one copy of the DBG Server for each server license you acquire.
- Provide you with additional benefits of owning legal software, such as ongoing technical support.

To facilitate the terms of your purchase agreement and your licensing requests, the License Manager offers "per instance" server licenses, which are particularly beneficial for maintaining a standby or backup system for the DBG Server.

With a "per instance" server license, you may install more than one copy of the DBG Server software on more than one Server. However, under those same terms, you must acquire one server license for each DBG Server that you run. For example, if one Database Gateway service stops, it can return its license to the License Manager, so that an alternate DBG Server may acquire a license and start its service.

In addition, even if the License Manager Server itself stops, Database Gateway's built-in "grace period" enables the service to continue its operation for 30 minutes more. If the License Manager Server returns during this period, Database Gateway will continue to function as normal. However, if the License Manager does not return online within the grace period, Database Gateway will cease to operate, and must be restarted when the License Manager Server returns online.

Separately, if you only want to evaluate Database Gateway, Williams offers an alternative to the License Manager and the hardware license key with a temporary license keycode. A keycode limits the time period in which you can use Database Gateway, and may be acquired on an interim basis from your Williams representative.

For additional information about licensing, contact your Williams representative. For additional information about the License Manager, see the *License Manager Installation Guide*.

Index

Notes

Notes